



دانشگاه صنعتی جندی شاپور دزفول
دانشکده برق و کامپیوتر

گزارش پروژه‌ی پایانی
رشته مهندسی کامپیوتر – سخت افزار

**بررسی معماری پردازنده‌های گرافیکی (GPU)
و مقایسه آنها با پردازنده‌های عمومی**
(نسخه غیرنهایی)

دانشجو:

احمد لشگر

استاد راهنمای پروژه:

مهندس مجتبی سلطانی

تیر ماه ۱۳۸۹

بسم الله الرحمن الرحيم

فهرست مطالب:

۱	 مقدمه
۱	 فصل اول: تکامل معماری GPU ها
۲	 ۱-۱ خطلوله گرافیک
۳	 ۱-۱-۱ ورود رئوس به حافظه GPU
۳	 ۱-۱-۲ سایه‌زن راس
۶	 ۱-۱-۳ سایه‌زن هندسی
۷	 ۱-۱-۴ رسترایزر
۸	 ۱-۱-۵ سایه‌زن پیکسل
۹	 ۱-۱-۶ ترکیب نهایی
۱۰	 ۱-۱-۷ بافت نگاری
۱۴	 ۱-۱-۸ روش‌های ضدناصافی
۱۶	 ۲-۱ رویکرد معماری GPU ها
۲۱	 ۳-۱ نسل‌های GPU
۲۱	 ۱-۳-۱ نسل اول (۱۹۹۶) - رسترایزر
۲۲	 ۲-۳-۱ نسل دوم (۱۹۹۸) - ثابت‌کاره
۲۲	 ۳-۳-۱ نسل سوم (۲۰۰۱) - سایه‌زن راس برنامه‌پذیر
۲۳	 ۴-۳-۱ نسل چهارم (۲۰۰۳) - سایه‌زن راس و سایه‌زن پیکسل برنامه‌پذیر
۲۳	 ۵-۳-۱ نسل پنجم (۲۰۰۵) - سایه‌زن یکپارچه
۲۴	 ۶-۳-۱ نسل ششم (۲۰۰۶) - سایه‌زن راس، سایه‌زن هندسی و سایه‌زن پیکسل برنامه‌پذیر
۲۷	 فصل دوم: معماری NV40
۳۰	 ۱-۲ پردازنده رئوس

۳۲	۲-۲ تنظیم اولیه و رسترایزر
۳۴	۳-۲ پردازنده فرگمنت
۳۸	۴-۲ واحد ترکیب پیکسل
۳۹	۵-۲ سنجش کارایی
۴۱	فصل سوم: معماری G80
۴۵	۱-۳ آرایه‌ی پردازنده جریان
۴۶	۲-۳ خوشه بافت/پردازنده
۴۷	۳-۳ چندپردازنده جریان
۴۸	۱-۳-۳ سیستم چندنخی در SM
۴۹	۲-۳-۳ مدیریت و زمان‌بندی نخ‌ها
۵۳	۳-۳-۳ تحمل تاخیر
۵۴	۴-۳-۳ دستورالعمل‌های SM
۵۶	۵-۳-۳ پردازنده جریان
۵۷	۶-۳-۳ واحد توابع خاص
۵۸	۴-۳ واحد بافت
۵۸	۵-۳ رسترایزر
۵۹	۶-۳ پردازنده عملیات رستر
۶۱	فصل چهارم: مقایسه معماری CPU و GPU
۶۱	۱-۴ حجم نخ
۶۱	۲-۴ وابستگی نهان
۶۲	۳-۴ مقیاس‌پذیری
۶۲	۴-۴ کلاکینگ و فرکانس
۶۲	۵-۴ سطح موازی‌سازی
۶۳	۶-۴ زمان‌بندی

۶۳.....	۷-۴ مصرف ترانزیستور
۶۳.....	۸-۴ سلسله مراتب حافظه
۶۴.....	منابع
۶۶.....	اندیس ها

مقدمه

ایده ساخت پردازنده‌ای کمکی برای پردازش‌های گرافیکی وقتی به ذهن طراحان رسید که بار گرافیکی روی دوش CPU سنگینی می‌کرد. تا پیش از برخورد به این محدودیت، تمامی الگوریتم‌ها گرافیکی روی CPU پیاده‌سازی می‌شدند.

معماری‌های اولیه GPUها ساخته شده از واحدهایی بودند که هرکدام از این واحدها، مرحله‌ای از خطلوله را تشکیل می‌دادند. این خطلوله که موسوم به خطلوله گرافیکی است، در ساده‌ترین شکل شامل مراحل زیر است:

۱. ابتدا رئوس شی از حافظه اصلی به حافظه GPU وارد می‌شوند.
 ۲. انتخاب رئوسی که از زاویه دید فعلی قابل رویت هستند و حذف باقی رئوس. سپس تبدیل‌ها و نورپردازی‌ها روی این رئوس انجام می‌شود.
 ۳. رئوس قابل رویت به مثلث‌هایی مجموعه بندی می‌شوند. این مثلث‌ها تنها با سه راس معرفی می‌شوند. هرکدام از رئوس این مثلث دارای خصوصیت‌هایی هستند.
 ۴. هرکدام از این مثلث‌ها به پیکسل‌هایی تبدیل می‌شوند و خصوصیت‌های این پیکسل‌ها از روی خصوصیت‌های سه راس مثلث درونیابی می‌شود.
 ۵. پیکسل‌ها در این مرحله دارای عمق و رنگ هستند و براساس مشخصه‌ی عمق، پیکسل‌هایی که قابل رویت نیستند حذف می‌شوند.
- عملیات‌های حذف رئوس غیرقابل رویت، تبدیل مثلث‌ها به پیکسل‌های متناظرشان، حذف پیکسل‌های غیرقابل رویت و رنگ‌آمیزی پیکسل‌ها، با الگوریتم‌های استاندارد که تماماً به کمک سخت‌افزار پیاده‌سازی شده بودند، انجام می‌شدند. به دلیل این پیاده‌سازی سخت‌افزاری، خطلوله گرافیکی انعطاف‌پذیری بسیار کمی داشت. با توسعه الگوریتم‌های جدید نورپردازی و سایه‌زنی رئوس و پیکسل‌ها که افکت‌های پیچیده‌ای را معرفی می‌کردند، انعطاف‌پذیرتر و قابل برنامه‌ریزی‌تر شدن GPUها احساس شد.

مشکل دیگر معماری‌های اولیه GPU، اتلاف سخت‌افزار بود. باری که روی واحدهای مختلف خطلوله وارد می‌شد برای برنامه‌های مختلف متفاوت بود؛ مثلاً برای تحلیل‌های گرافیکی‌ای که تعداد مثلث‌ها زیاد بود، واحد سایه‌زن رئوس بسیار مشغول می‌شد درحالی که واحدهای دیگر کم‌کار بودند و برای تحلیل‌های گرافیکی که چند مثلث بزرگ داشتند، واحد سایه‌زن پیکسل‌ها بسیار مشغول می‌شد. اگر

معماری GPU طوری بود که واحدهای شبیه به هم، تاحد امکان یکی بودند، واحدهای بیکار کمتر می شدند و کارایی بالاتر می رفت.

اواخر سال ۲۰۰۰ بود که برای نخستین بار صحبت از سایه زنی های یکپارچه به میان آمد. رفته رفته سایه زنی های یکپارچه معماری GPU ها را متحول کردند و مشکل اتلاف سخت افزار را هم برطرف کردند. در معماری مدرن GPU ها، یک واحد کاملاً قابل برنامه ریزی وجود دارد که قادر به انجام چندین مرحله از خطلوله است. این واحد عموماً Stream Processor نامیده می شود زیرا ورودی های گرافیکی (خطوط، مثلث ها، اشیا و...)، جریانی از داده ها را تشکیل می دهند که مستقل از هم هستند و باید پردازش موازی یکسانی روی تمام آنها انجام شود. مجموعه دستورالعمل های SP، تمامی دستورهای لازم برای سایه زنی راس و پیکسل را دارد. باساخته شدن چنین واحد یکپارچه ای که دستورات پرش، ذخیره و بازیابی از حافظه را نیز داشت، GPU شبیه به پردازنده ای چندین هسته ای شد. معماری امروزه GPU ها، متشکل از این پردازنده های چند هسته هستند، به همراه برخی از واحدهای «ثابت کاره» که از ابتدای ساخت اولین GPU تا به امروز، کم تغییر بوده اند. از جمله این واحدها رسترایزرها هستند.

نمی توان معماری GPU ها را بررسی کرد بدون اینکه از گرافیک کامپیوتری صحبت نکرد. برای انجام سریع محاسبات گرافیکی، رئوس، مثلث ها، خط ها و پیکسل ها به گونه خاصی مدیریت می شوند که بسیار ظریف و زیرکانه ابداع شده اند. دستگاه های مختصاتی غیر از دکارتی برای نمایش اشیا و سرعت دادن به تغییر زاویه دید، عملیات ها تشخیص عضویت نقطه در مثلث، درونیابی رنگ پیکسل، حذف پیکسل های رویت ناپذیر و ... استفاده می شوند که برای درک معماری GPU ها باید آنها را دانست.

دراین مقاله، ابتدا مروری بر گرافیک کامپیوتری داریم و بیشتر روی خطلوله گرافیکی تاکید می کنیم. سپس جهت آشنایی اولیه با معماری GPU، سیر تکاملی و نسل های مختلف معماری آنها را از اولین تا سال ۲۰۰۶ مرور می کنیم. سپس به معرفی عمیق تر برخی از این معماری ها می پردازیم. از آنجایی که معماری مدرن GPU بسیار شبیه به معماری پردازنده های چندین هسته ای است، مقایسه ای بین معماری GPU و CPU انجام می دهیم و این دو پردازنده را از جنبه های گوناگون بررسی می کنیم.

فصل اول: تکامل معماری GPUها

از آنجایی که هدف اصلی از ساخت GPUها، شتاب دادن به پردازش‌های گرافیکی بوده است، برای آشنایی با معماری‌های قدیمی و مدرن GPU لازم است که نگاهی به ساختار گرافیک کامپیوتری داشته باشیم. الگوریتم‌ها و پیچیدگی‌های این مبحث خارج از محدوده‌ی این نوشتار هستند و در این بخش تنها مفاهیم مقدماتی معرفی می‌شوند.

یک صحنه‌ی گرافیکی پُر جزئیات، از اشیاء سه‌بعدی متعددی ساخته شده است. اشیاء سه‌بعدی همواره با مجموعه‌ای از رئوس^۱ معرفی می‌شوند. اتصال این رئوس گسسته به هم (تشکیل خط یا مثلث)، یک شی می‌سازد که پیوسته به نظر می‌رسد. برای نمایش یک شی هرچقدر تعداد این رئوس بیشتر باشد، سطوح شی نرم‌تر^۲ و واقعی‌تر خواهند بود. در تصویر زیر شی‌ای را مشاهده می‌کنید که با مجموعه‌ای از رئوس معرفی شده است:



تصویر ۱

^۱ Vertex

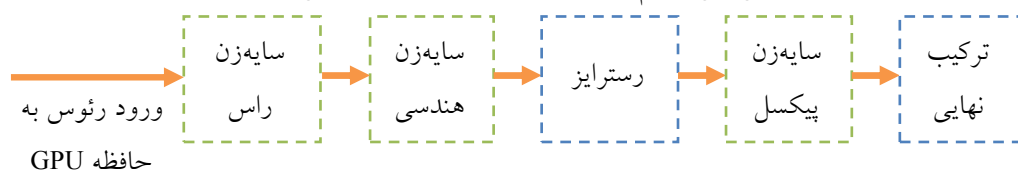
^۲ Smoother

تمامی اشیای موجود در صحنه بوسیله مجموعه‌ای از رئوس معرفی می‌شوند و برای پردازش‌های گرافیکی و نمایش یک صحنه تولید شده با کامپیوتر^۱، فرآیند پردازش از این رئوس شروع می‌شود. کلیه رئوس خطلوله‌ای را طی می‌کنند و در انتها به آرایه‌ای دوبعدی از پیکسل^۲هایی تبدیل می‌شوند که هرکدام یک رنگ مشخص دارند (به کل این فرآیند رندر^۳ می‌گویند). این آرایه دوبعدی در قسمتی به نام بافر فریم^۴ یا بافر تصویر موجود در GPU ذخیره می‌شود. این بافر فریم با نرخ مشخصی به مانیتور فرستاده می‌شود و همان است که در صفحه نمایش مشاهده می‌کنیم.

۱-۱ خطلوله گرافیک^۵

خطلوله گرافیکی دارای چندین مرحله^۶ است که طی سال‌ها تکمیل شده و مراحل به آن اضافه شده است. خروجی هر مرحله ورودی مرحله بعد است. برخی از مراحل قابلیت برنامه‌ریزی دارند و برخی دیگر بدون قابلیت برنامه‌ریزی‌اند و توسط سخت‌افزار انجام می‌شوند و خودکار، خروجی را به مرحله‌ی بعد می‌دهند.

امروزه خطلوله گرافیکی در هنگام کدنویسی گرافیکی بسیار حائز اهمیت است. زبان‌های برنامه‌نویسی گرافیکی Direct3D و OpenGL هر کدام خطلوله مخصوص به خود را دارند ولی کلیت هر دو یکسان است. اگرچه سازندگان GPU حلالامکان معماری خود را نزدیک به این خطلوله‌ها می‌سازند تا کدها با بیشترین کارایی اجرا شوند اما آنچه در GPUها می‌گذرد مستقل از این زبان‌ها است و این است که کارایی گرافیکی GPUها باهم متفاوت است. اگر بخواهیم خطلوله‌ای نوعی^۷ ارائه دهیم که مستقل از سازنده‌ای خاص است به مراحل می‌رسیم که در تصویر زیر مشاهده می‌کنید.



¹ Computer Generated Scene

² Pixel (Picture Cell)

³ Render

⁴ Frame Buffer

⁵ Graphics Pipeline

⁶ Stage

⁷ Typical

در ادامه هریک از این مراحل خطلوله را آنچنان که برای درک سخت‌افزار این کار کافی است، معرفی می‌کنیم.

۱-۱-۱ ورود رئوس به حافظه GPU

وقتی پردازنده به دستوری می‌رسد که باید بار محاسباتی را به پردازنده کمکی انتقال دهد، GPU را مطلع می‌کند و از طریق درایور کارت گرافیک، کد برنامه و رئوس و باقی داده‌ها لازم به حافظه GPU منتقل می‌کند. این رئوس مختصات مطلق شی در محیط دنیا را بیان می‌کنند.

۱-۱-۲ سایه‌زن راس^۱

سایه‌زن راس با بسیاری از مختصات‌ها سر و کار دارد. مختصات محلی شی که در فضای شی^۲ تعریف می‌شود برای ویرایش و تعریف یک شی بکار می‌رود. مختصات سراسری شی در فضای دنیا^۳ نیز تعریف می‌شود که در این دستگاه تمام اشیا در کنار هم قرار می‌گیرند. برای رندر، باید اشیا را از زاویه دوربین یا چشم باید دید. فضای دوربین^۴ مختصات شی را در زاویه دید دوربین تعریف می‌کند. در این مختصات رئوس اشیا، مختصات x,y,z می‌گیرند که محور z در امتداد زاویه‌ی دوربین قرار می‌گیرد. تبدیل از فضای شی به فضای دنیا و از فضای دنیا به فضای دوربین باید انجام شود، این کار عموماً با ترکیب ماتریس تبدیل مراحل، یکباره انجام می‌شود که به آن نمایش مدل^۵ می‌گویند.

پس از اینکه رئوس تبدیل به فضای دوربین شدند، باید تحت تبدیلی قرار گیرند تا تاثیر پرسپکتیو در آنها دیده شود؛ هرچه از زاویه‌ی دید دورتر می‌شویم، اشیای کوچکتر شوند. به این عملیات پروجکشن^۶ می‌گویند و معمولاً در مختصات همگن^۷ 4^y بعدی انجام می‌شود. به فضایی که پس از پروجکشن ایجاد

¹ Vertex Shader

² Object Space

³ World Space

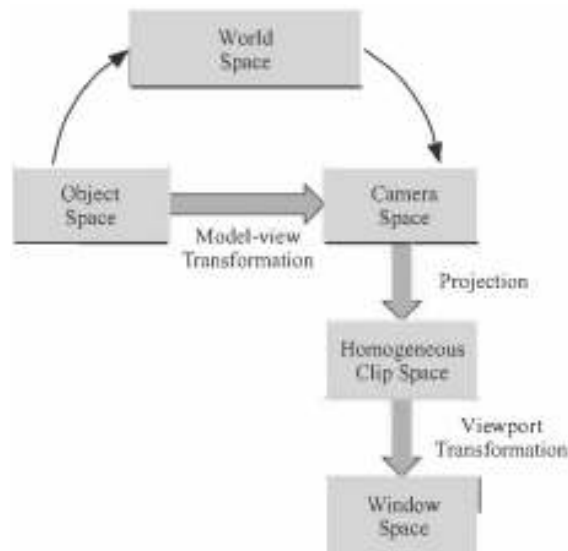
⁴ Camera Space

⁵ Model-view

⁶ Projection

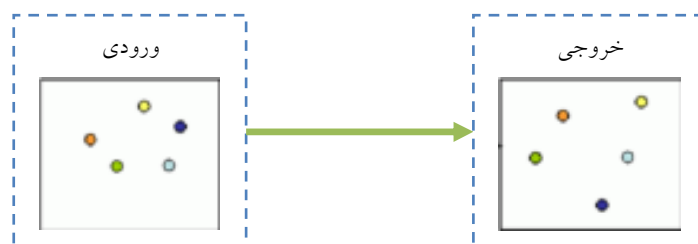
⁷ Homogeneous Coordinate

می‌شود، فضای همگن برش^۱ می‌گویند. در این فضا، رئوس در هر سه محور X, Y, Z مختصات نرمال شده در بازه $[-1, 1]$ دارند.



تصویر ۲- مراحل تغییر مختصات رئوس

آخرین تبدیلی که باید رئوس در این مرحله بگذرانند، تبدیل زاویه‌ی دید^۲ است. زاویه‌ی دید اغلب ناحیه‌ای از نمایش است که شامل تصویر رندر شده است، البته ممکن است زیر ناحیه‌ای از پنجره، کل پنجره یا کل نمایش باشد. این تبدیل، مختصات نرمال شده را به محدوده واقعی مختصات پیکسل‌هایی که زاویه‌ی دید پوشش داده می‌شوند نگاشت می‌کند. مختصات Z معمولاً به محدوده $[0, 1]$ نگاشت می‌شود. پس از پایان این تبدیل رئوس در فضای پنجره^۳ قرار می‌گیرند [21]. در تصویر زیر ورودی و خروجی یک سایه‌زن راس را مشاهده می‌کنید.



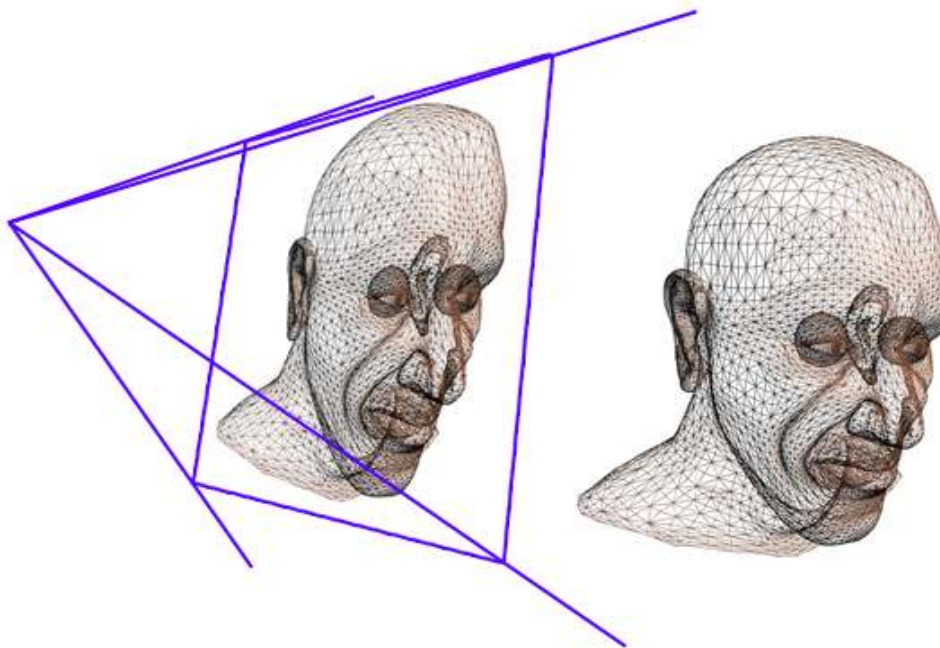
تغییر دستگاه مختصاتی رئوس و محاسبه مختصات جدید از زاویه‌ی دید موردنظر (تصویر از منبع [14])

^۱ Homogeneous Clip Space

^۲ Viewport Transformations

^۳ Window Space

محاسبات نورپردازی رئوس و انتخاب^۱ رئوس قابل رویت از زاویه‌ی دید جاری و حذف باقی رئوس شی، در این مرحله انجام می‌شود. رئوس در این مرحله دارای خصوصیت‌های مثل مختصات، رنگ و آلفا^۲ (درجه شفافیت) هستند.



تصویر ۳

واحدی از GPU که کلیه‌ی این عملیات‌ها را انجام می‌دهد «سایه‌زن راس» نامیده می‌شود. هر راس می‌تواند مختصات یک یا چند مجموعه از مختصات بافت^۳ را به عنوان خصوصیت در برداشته باشد. مختصات بافت می‌تواند توسط برنامه یا توسط سایه‌زن راس به صورت خودکار تعیین شوند (در ادامه در رابطه با بافت‌ها بیشتر بحث می‌کنیم). در معماری‌های اولیه GPUها، سایه‌زن راس قابل برنامه‌نویسی و کنترل کامل نبود ولی در معماری‌های مدرن این عملیات کاملاً قابل برنامه‌ریزی‌اند. یکی از محدودیت‌ها و نکات حائز اهمیت این مرحله آن است که هیچ راس جدیدی ساخته نمی‌شود و تنها تبدیل و انتخاب صورت می‌گیرد. به علت اینکه رئوس مستقل از یکدیگرند، پردازش این بخش می‌تواند کاملاً موازی انجام می‌شود.

^۱ Culling

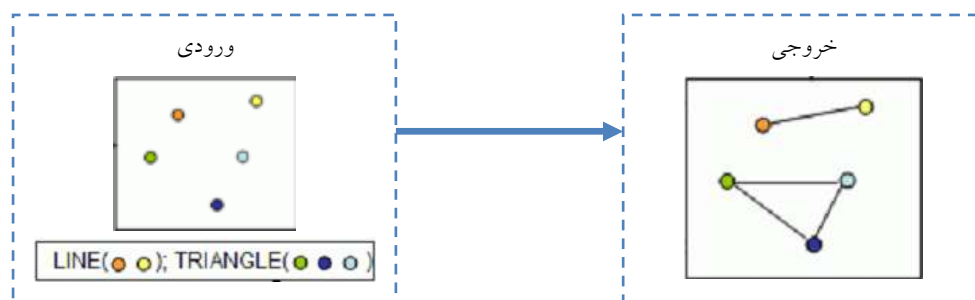
^۲ Red Green Blue Alpha (RGBA)

^۳ Texture Coordinate

۳-۱-۱ سایه‌زن هندسی^۱

در خط‌لوله‌های GPUهای قدیمی، این مرحله منحصر به ساخت اولیه^۲ بود. ساخت اولیه عبارت است از تشکیل نقطه، خط یا مثلث از روی رئوسی که توسط سایه‌زن راس پردازش شده‌اند. به هرکدام از این نقطه، خط یا مثلث‌ها در علم گرافیک کامپیوتری، اولیه^۳ می‌گویند.

ساخت اولیه درواقع وظیفه اتصال دادن رئوس را برعهده دارد که این اتصال دادن توسط اطلاعات دیگری که نحوه اتصال (خط یا مثلث) را مشخص می‌کند، انجام می‌شود. راسی که با هیچ راس دیگری اتصال ندارد، به اولیه‌ی نقطه تبدیل می‌شود. هر راسی که توسط سایه‌زن راس پردازش شده، در قالب یک اولیه (نقطه، خط یا مثلث) از این مرحله خارج می‌شود و یا اینکه حذف می‌شود. البته ممکن است که راس در بیش از یک اولیه هم ظاهر شود (معمولاً هر راس در تشکیل چند مثلث شرکت دارد). در تصویر زیر ورودی و خروجی نوعی این مرحله را مشاهده می‌کنید.



ایجاد اولیه متناظر با رئوس، با استفاده از اطلاعات اتصال (تصویر از منبع [14])

در GPUهای مدرن، ساخت اولیه جزو واحدی بزرگتر است که سایه‌زن هندسی نام دارد. این واحد پس از انجام عملیات ساخت اولیه‌ها، با افزودن رئوس جدید (که مشخصاتشان از باقی رئوس درونیابی می‌شود)، اولیه‌های جدیدی می‌سازد. افزایش تعداد اولیه‌ها باعث ارتقاء جزئیات تصویر نهایی می‌شود.

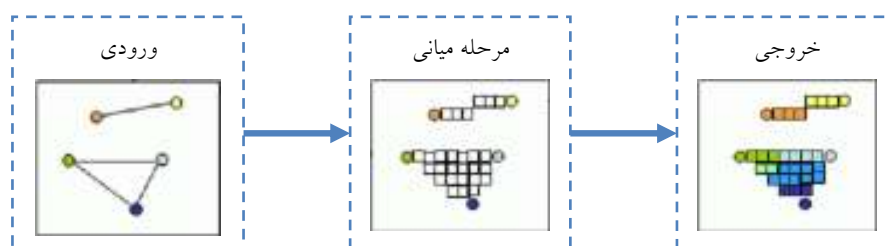
^۱ Geometry Shader

^۲ Primitive Assembly

^۳ Primitive

۱-۴ رسترایزر^۱

تا اینجا سطوح و اشیا با مجموعه‌ای از مثلث‌ها و خط‌ها و نقطه‌ها توصیف شده‌اند. برای اینکه این اولیه‌ها روی مانیتور قابل نمایش باشند، باید آنها را به پیکسل‌هایی تجزیه کرد (در برخی منابع به این پیکسل، فرگمنت^۲ می‌گویند). در این تعریف پیکسل آن چیزی است که روی صفحه مانیتور نمایش داده می‌شود و تنها خصوصیت رنگ دارد ولی فرگمنت دارای خصوصیت دیگری (از جمله شفافیت و عمق) نیز هست. در این مقاله ما هر دو را به یک معنی بکار می‌بریم و به موجودیتی اشاره می‌کنیم که دارای خصوصیت رنگ، شفافیت و عمق است و هر جا منظورمان پیکسل صفحه نمایش باشد، مستقیماً ذکر می‌کنیم). بسته به اندازه‌ی این اولیه‌ها، ممکن است شامل یک تا چندصد پیکسل باشد. واحدی به نام «رسترایزر» وظیفه این گسسته سازی را دارد و مختصات دوبعدی، عمق و رنگ هر پیکسل را تعیین می‌کند. واحد رسترایزر در اغلب معماری‌ها، ثابت‌کاره^۳ است و قابل برنامه‌ریزی نیست. در تصویر زیر نمونه‌ای از ورودی و خروجی رسترایزر را مشاهده می‌کنید.



ابتدا عملیات گسسته‌سازی (رستر) انجام می‌شود و سپس خصوصیات (رنگ) باتوجه به خصوصیات رئوس برای هر پیکسل درونیابی می‌شود. (تصویر از منبع [14])

پدیده‌ی اره‌ای شدن یا ناصافی^۴، از رایج‌ترین اتفاقات پس از رسترایز کردن است. برای رفع این پدیده، عموماً عملیات ضدناصافی^۵ هم انجام می‌شود. این کار با تغییر شفافیت پیکسل‌های مرزی انجام می‌شود. (شفافیت از خصوصیت‌های پیکسل است که در مرحله سایه‌زنی پیکسل و برای ترکیب لایه‌ها لحاظ می‌شود [8]). این عملیات در تئوری بلافاصله پس از رسترایزر باید انجام شود تا تاثیر آن را از بین

¹ Rasterizer

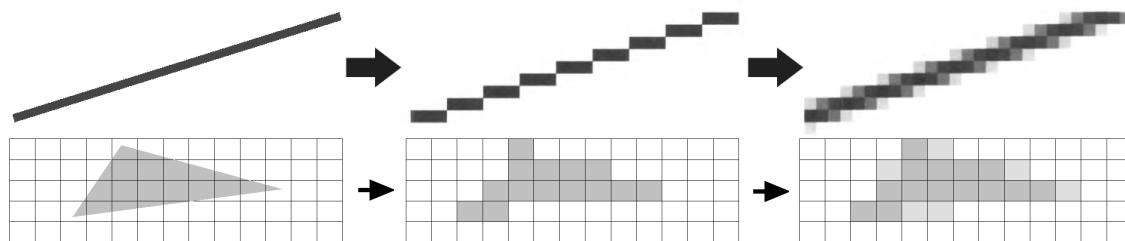
² Fragment

³ Fixed-Function

⁴ Aliasing

⁵ Anti-Aliasing (AA)

ببرد ولی در پیاده‌سازی عملی در GPUها، این عملیات در مرحله ترکیب نهایی انجام می‌شود. در تصویر ۴ نمونه‌هایی از عملیات کامل‌تر و گام‌به‌گام واحد رسترایزر را مشاهده می‌کنید.



تصویر ۴

در ۱-۱-۸ روش‌های ضدناصافی را معرفی می‌کنیم.

۱-۱-۵ سایه‌زن پیکسل^۱

در این مرحله رنگ و خصوصیت پیکسل‌های درونیابی شده از رسترایزر را نهایی می‌کنیم. عملیات این مرحله مستقلاً روی تک‌تک پیکسل‌ها انجام می‌شود. گستره‌ی این عملیات که کاملاً قابل برنامه‌ریزی‌اند، از بی‌تغییر گذاشتن مشخصات پیکسل، تا تغییر عمق پیکسل، نورپردازی، نگاشت بافت^۲ (بافت نگاری) و نگاشت فرورفتگی^۳ هستند.

به علت تعداد پیکسل‌ها، این مرحله یکی از محاسباتی‌ترین مراحل خط لوله است و البته مشابه سایه‌زن راس، محاسبات این بخش نیز به علت ماهیت پیکسل‌ها کاملاً موازی می‌تواند انجام می‌شود. یکی از عملیات‌های بسیار پرکاربرد و البته اختیاری این مرحله بافت‌نگاری است. در بخش (۱-۱-۷ بافت نگاری) در این باره بحث شده است.

^۱ Pixel Shader

^۲ Texture Mapping

^۳ Bump mapping

۱-۱-۶ ترکیب نهایی^۱

تا اینجا لایه‌هایی از پیکسل‌ها داریم که عمق، رنگ، شفافیت و بافت منحصر به فردی دارند. مرحله آخر خطلوله وظیفه ترکیب خصوصیت‌های پیکسل، حذف پیکسل‌های غیرقابل رویت و ترکیب پیکسل‌های شفاف را دارد. پیش از وارد شدن به مرحله ترکیب، هر پیکسل باید از چندین سنجش عبور کنند.

اول سنجش مالکیت پیکسل^۲ انجام می‌شود. خیلی ساده بررسی می‌شود که آیا پیکسل از دید دوربین قابل رویت است یا خیر. پیکسل وقتی از این مرحله عبور نمی‌کند که پنجره‌ای در مقابل دوربین مانع رویت آن باشد.

دوم سنجش قیچی^۳ اجرا می‌شود. برنامه گرافیکی ممکن است ناحیه‌ای در معین کرده باشد (که به آن مستطیل قیچی^۴ می‌گویند) که رندر به آن محدود شود. هر پیکسلی که خارج از این ناحیه قرار گیرد باید نادیده گرفته شود. یکی از کاربردهای مستطیل قیچی، پردازش ناحیه درون سایه است.

سوم سنجش آلفا^۵ است. آلفا درجه شفافیت رنگ پیکسل را مشخص می‌کند. در سنجش، مقدار آلفا پیکسل با مقدار مخصوصی که برنامه‌ریزی شده مقایسه می‌شود. اگر رابطه برقرار نبود، پیکسل نادیده گرفته می‌شود.

چهارم سنجش استنسیل^۶ است. بافری به نام بافر استنسیل وجود دارد که یک ماسک برای پیکسل‌ها است. این بافر برای هر پیکسل عددی را ذخیره می‌کند که بوسیله آن می‌توان پیکسل‌ها را گروه‌بندی کرد. عدد استنسیل هر پیکسل از بافر خوانده می‌شود و با مقدار برنامه‌ریزی شده مقایسه می‌شود. اگر شرط برقرار بود پیکسل به مرحله بعدی می‌رود (می‌تواند بافر استنسیل را برای نشانه‌گذاری تغییر دهد) و در غیر این صورت نادیده گرفته می‌شود.

پنجمین و آخرین سنجش بررسی عمق^۷ است که حذف پیکسل‌های غیرقابل رویت از طریق الگوریتم z-buffer (که نام دیگر آن depth buffering است) را پیاده می‌کند. این الگوریتم اساس بسیار ساده‌ای دارد؛ برای هر مکان پیکسلی انتخاب می‌شود که کمترین عمق را دارد.

¹ Final Composition (Blending)

² Pixel Ownership Test

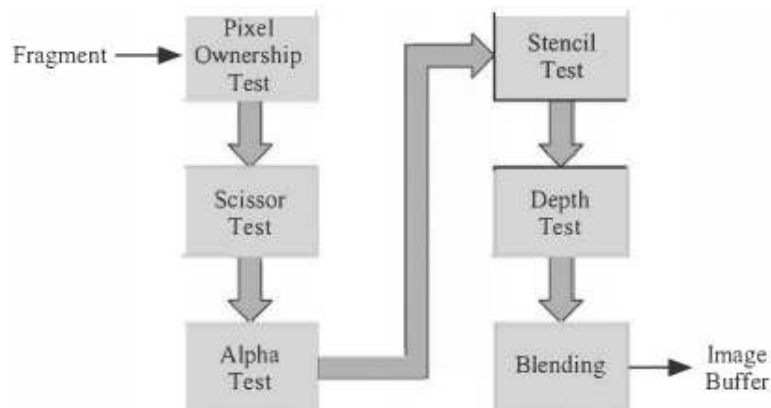
³ Scissor Test

⁴ Scissor rectangle

⁵ Alpha Test

⁶ Stencil Test

⁷ Depth Test



تصویر ۵- عملیات‌های ماقبل از ترکیب نهایی

پس از اینکه کلیه این سنجش‌ها انجام شد، ترکیب نهایی رنگ پیکسل‌ها با پیکسل درون بافر فریم انجام می‌شود. عملیات ترکیب ممکن است خیلی ساده با یک جابه‌جایی انجام شود و یا اینکه جلوه‌های ویژه‌ای مثل شفافیت نیز تولید شوند.

عملیات این مرحله خطلوله گرافیکی در GPUها توسط واحدی به نام «واحد عملیات رستر»^۱ انجام می‌شود [13].

۷-۱-۱ بافت نگاری

برای اینکه دیواری آجری را نشان دهید، لازم نیست تک‌تک آجرها را به عنوان شی تعریف کنید و کلیه مراحل خطلوله را روی آنها اجرا کنید. بافت‌نگاری راهی ساده برای این کار پیشنهاد می‌کند. بافت‌نگاری پوسته‌ای با طرح آجر را روی سطح شی گسترده می‌کند.

بافت‌الگویی با جزئیات است که چندین بار تکرار می‌شود تا سطح را بپوشاند؛ در تعریفی عمومی‌تر، تصویری چند بعدی است که به فضایی چند بعدی نگاشت می‌شود. بافت‌نگاری یعنی نگاشت با یک تابع به سطحی در فضای سه‌بعدی. دامنه این تابع می‌تواند یک، دو یا سه بعدی باشد [20]. نگاشت فرورفتگی که در سایه‌زنی پیکسل به آن اشاره شد، درجه‌ای بالاتر از نگاشت بافت است. در این حالت بافت تنها تعدادی پیکسل نیست بلکه از بردارهایی تشکیل شده است که شیب‌هایی در سطح شی ایجاد می‌کند.

¹ Raster Operation Unit

سه روش رایج برای ساخت سطح بافت‌نگاری شده وجود دارد؛ پویش در فضای تصویر^۱، پویش در فضای بافت^۲ و دوگذر^۳. الگوریتم این سه روش در زیر آمده است: (x,y) اندیس‌های فضای تصویر هستند و u,v اندیس‌های فضای بافت)

SCREEN SCANNING:

```
for y
  for x
    compute  $u(x,y)$  and  $v(x,y)$ 
    copy  $TEX[u,v]$  to  $SCR[x,y]$ 
```

TEXTURE SCANNING:

```
for v
  for u
    compute  $x(u,v)$  and  $y(u,v)$ 
    copy  $TEX[u,v]$  to  $SCR[x,y]$ 
```

TWO-PASS:

```
for v
  for u
    compute  $x(u,v)$ 
    copy  $TEX[u,v]$  to  $TEMP[x,v]$ 
  for x
    for v
      compute  $y(x,v)$ 
      copy  $TEMP[x,v]$  to  $SCR[x,y]$ 
```

که TEX آرایه‌ی بافت، SCR آرایه‌ی تصویر (مقصد) و $TEMP$ آرایه‌ی موقتی است.

پویش تصویر، که نگاشت معکوس^۴ هم نامیده می‌شود، رایج‌ترین روش است. برای هر پیکسل در فضای تصویر، پیکسلی در فضای بافت پیدا و این ناحیه فیلتر می‌گردد (در ادامه درباره متودهای فیلترینگ بحث می‌شود). این روش وقتی که تصویر باید به‌ترتیب^۵ نوشته شود، نگاشت به‌سادگی معکوس می‌شود و بافت قابل دسترسی تصادفی^۶ است، مناسب است.

پویش بافت ساده‌تر از پویش تصویر است زیرا نیازی به معکوس کردن نگاشت نیست ولی ظرافت خاصی می‌خواهد. نمونه‌برداری یکنواخت از بافت، نمونه‌برداری یکنواخت از تصویر را تضمین نمی‌کند،

¹ Scan in screen space

² Scan in texture space

³ Two-pass

⁴ Inverse Mapping

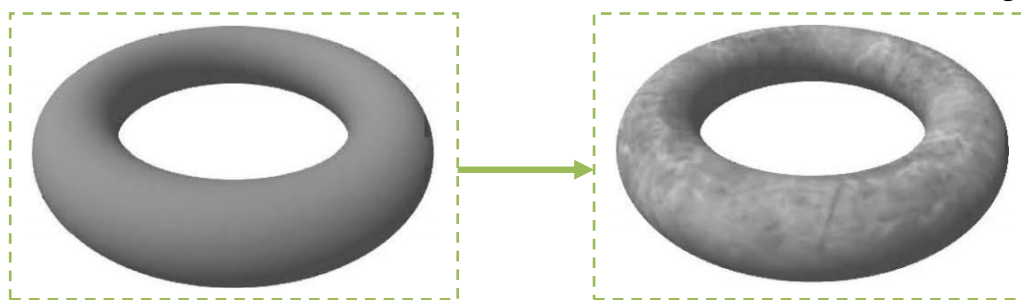
⁵ Sequential

⁶ Random Access

مگر برای نگاشت خطی^۱. از این رو برای نگاشت‌های غیرخطی، اغلب تقسیم بافت باید انجام شود؛ که در غیر این صورت فاصله یا همپوشانی در تصویر ایجاد می‌شود. این روش وقتی که پوشش تصویر دشوار باشد و یا بافت باید به‌ترتیب خوانده شود و یا در حافظه دسترسی تصادفی نمی‌گنجد مناسب است.

روش دوگذر، نگاشت دوبعدی را به دونگاشت تک بعدی ساده می‌کند. در گذر اول روی ردیف‌های تصویر اعمال می‌شود و گذر دوم روی ستون‌ها. این روش وقتی که بافت قابل دسترسی تصادفی نیست ولی دسترسی به ردیف یا ستون سریع است و البته بافری برای ذخیره تصویر میانی (TEMP) وجود دارد، مناسب است.

بافت‌نگاری تنها خصوصیت‌های رنگ و شفافیت (RGBA) را تغییر می‌دهد. (شایان ذکر است که بافت‌نگاری در مرحله سایه‌زنی راس ممکن است انجام شود[8]). در تصویر زیر عمل بافت‌نگاری را مشاهده می‌کنید.



بکاربردن بافت‌نگاری به مدل جزئیات اضافه می‌کند (تصویر از منبع [21])

پس از محاسبات نگاشت و کشیدن بافت روی سطح، فیلترکردن بافت^۲ برای جلوگیری از ناصافی ضروری است. فیلترینگ نمونه‌برداری مجدد از بافت برای تصویر است.

کم‌هزینه‌ترین روش فیلترکردن بافت، نمونه‌برداری نقطه^۳ (فیلترکردن نزدیک‌ترین همسایه^۴) است که نزدیک‌ترین پیکسل به نقطه نمونه‌برداری شده را استفاده می‌کند. این روش برای تصاویری که هم مقیاس بافت هستند، بسیار مناسب است ولی برای تصاویر بزرگ، پیکسل‌های بافت بلوک‌های بزرگی دیده می‌شوند و برای تصاویر کوچک، مشکل موج‌دار شدن و انحنا پدید می‌آید.

¹ Affine (linear) mapping

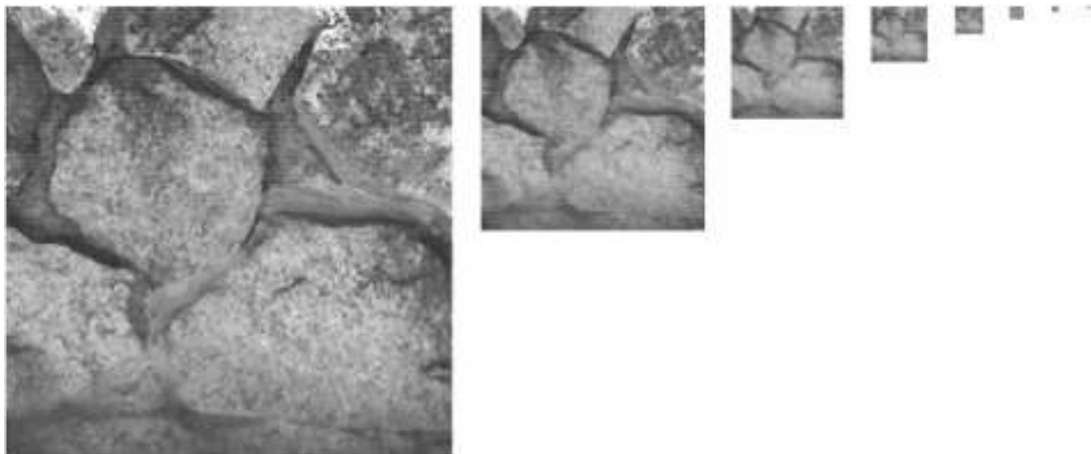
² Texture Filtering

³ Point Sampling

⁴ Nearest-neighbor Filtering

روش دوم فیلترکردن دوطقی^۱ نام دارد که برای هر پیکسل تصویر چهار پیکسل از بافت واکشی می‌کند و با هم ترکیب می‌کند. این چهار نمونه با میانگین وزنی باهم ترکیب می‌شوند و هر وزن بستگی به رابطه مختصات پیکسل بافت با پیکسل تصویر دارد [21]. وقتی مدل از دوربین دور می‌شود یا ناحیه‌ی دید بسیار کوچکتر از بافت می‌شود، ناصافی در این روش شدت می‌گیرد.

روش سوم که فیلترکردن سه‌خطی^۲ نام دارد؛ گونه‌ای از فیلترکردن دوطقی است و درونیابی خطی بین میپ‌مپس^۳ انجام می‌دهد. میپ‌مپس مجموعه‌ای از تصاویر است که از روی بافت اصلی بدست آمده‌اند. این تصاویر قبل از عملیات بافت نگاری محاسبه می‌شوند و معمولاً در اندازه‌های کوچکتر از تصویر اصلی ساخته می‌شوند.



تصویر ۶- میپ‌مپس‌های یک تصویر مشخص (تصویر از منبع [21])

فیلترکردن سه‌خطی، بسته به اندازه مدل، دو تصویر متوالی از میپ‌مپس را انتخاب می‌کند (مثلاً اگر تصاویر میپ‌مپس در ابعاد 8×8 ، 16×16 ، 32×32 و 64×64 باشند و مدل 25×25 باشد دو تصویر 32×32 و 16×16 انتخاب می‌شوند). مطابق فیلترکردن دوطقی از هریک از دو تصاویر ۴ نمونه برداشته می‌شود و این ۸ نمونه باهم ترکیب می‌شوند و پیکسل نمونه‌برداری نهایی محاسبه می‌شود.

چهارمین روش، فیلترکردن ناهمجهت^۴ نام دارد و مخصوص زمانی است که سطح با راستای دید زاویه می‌سازد و برآن عمود نیست. فیلترکردن ناهمجهت نمونه‌هایی حول پیکسل مرکزی از میپ‌مپس

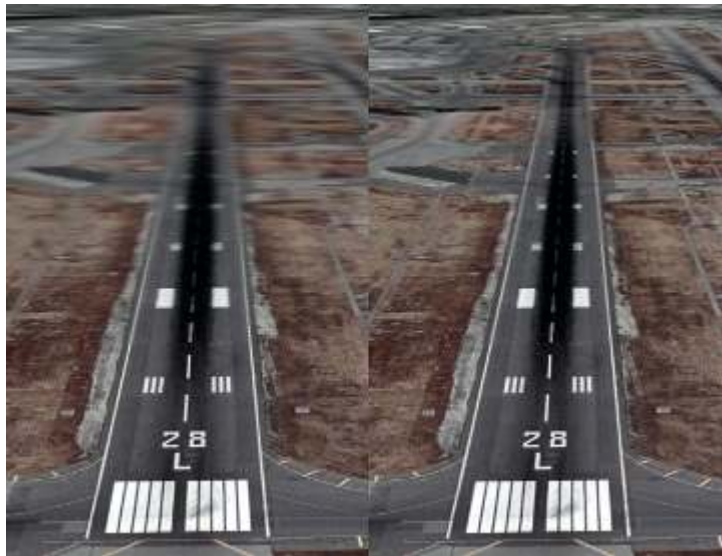
¹ Bilinear Filtering

² Trilinear filtering

³ Mipmaps

⁴ Anisotropic filtering

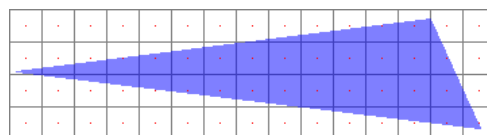
برمی دارد. تعداد نمونه‌ها با پارامتری مشخص می‌شود که نسبت نام دارد. اگرچه این روش کیفیت بسیار خوبی دارد ولی به محاسبات و دسترسی به حافظه زیادی احتیاج دارد و معمولاً با نسبت‌های 2:1، 4:1، 8:1 یا 16:1 انجام می‌شود. هرچه این نسبت بالاتر رود تصویر کمتر مات به نظر می‌آید و البته محاسبات بیشتری هم می‌طلبد زیرا نمونه‌های بیشتری برداشته می‌شود.



تصویر ۷-سطحی که با راستای دید زاویه می‌سازد و بر آن عمود نیست. بافت‌نگاری با فیلترکردن سه‌خطی سمت چپ و فیلترکردن ناهمجهت سمت راست (تصویر از ویکیپدیا)

۸-۱-۱ روش‌های ضدناصافی

دو روش متداول برای مقابله با ناصافی ناشی از رسترایز کردن وجود دارد: آبرنمونه‌برداری^۱ و چندنمونه‌برداری^۲. این دو روش مبتنی بر رسترایز کردن تصویری در ابعاد بزرگتر هستند. تصویر ۸ اولیه‌ای را نشان می‌دهد که می‌خواهیم پیکسل‌هایی که سطح اولیه را پوشش می‌دهند، بدست آوریم.

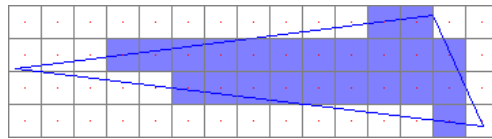


تصویر ۸-اولیه‌ای که آماده رسترایز شدن است (تصویر از منبع [30])

اگر عملیات رسترایز را روی این اولیه انجام دهیم پیکسل‌های تصویر ۹ را بدست می‌آوریم:

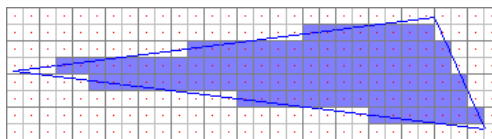
¹ Supersampling

² Multisampling



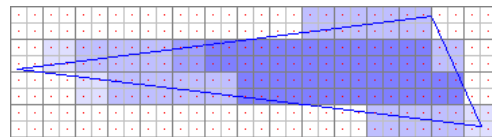
تصویر ۹-پیکسل‌های حاصل از رسترایزر (تصویر از منبع [30])

حال اگر اولیه فوق را در ابعادی ۴ برابر اندازه فعلی رسترایز کنیم، پوشش پیکسلی تصویر ۱۰ بدست می‌آید:



تصویر ۱۰-رسترایزر اولیه در ابعادی بزرگتر (تصویر از منبع [30])

هر ۴ پیکسل در تصویر ۱۰ معادل یک پیکسل در تصویر ۸ هستند. حال با میانگین‌گیری از رنگ‌های این چهار پیکسل، رنگ هر پیکسل در تصویر ۸ را پیدا می‌کنیم. نتیجه این کار را در تصویر ۱۱ می‌بینید:



تصویر ۱۱-حاصل نهایی رسترایز کردن به کمک تصویری با ابعاد بزرگتر (تصویر از منبع [30])

هر دو روش ابرنمونه‌برداری و چندنمونه‌برداری با میانگین‌گیری از پیکسل‌های تصویر بزرگتر کار می‌کنند. اگر برای هر پیکسل نهایی، از n پیکسل تصویر بزرگتر میانگین گرفته شود، مقیاس روش nx است. مقیاس‌های متداول $2x$ ، $4x$ ، $8x$ و $16x$ هستند. چندنمونه‌برداری روشی بهبود یافته (از نظر سرعت) ابرنمونه‌برداری است. ابرنمونه‌برداری سایه‌زن پیکسل را برای تمام پیکسل‌های تصویر بزرگتر اجرا می‌کند و برای هر کدام خصوصیات را محاسبه می‌کند درحالی که چندنمونه‌برداری، سایه‌زن پیکسل را تنها یکبار اجرا می‌کند و نتیجه را بین همه پیکسل‌ها اشتراک می‌گذارد. در هر دو روش عمق تمام پیکسل‌ها حساب می‌شود.

۱-۲ رویکرد معماری GPUها

کار^۱ی (سایه‌زنی راس، پیکسل و غیره) که باید در خطلوله گرافیکی پردازش شود را در نظر بگیرید، ورودی از تعداد زیادی عنصر (راس، اولیه یا پیکسل) تشکیل شده‌است. در همچنین خطلوله‌ای، خروجی هر کار، ورودی کار دیگری است. خطلوله، موازی‌سازی کار^۲ انجام می‌دهد زیرا مراحل مختلف خطلوله می‌توانند هم‌زمان پردازش‌های خود را انجام دهند. در هر مرحله نیز بیش از یک عنصر می‌تواند پردازش شود که موازی‌سازی داده^۳ است. اگر CPU این خطلوله را پیاده‌سازی کند، یک (یا چند اگر SIMD باشد) عنصر را خواهد گرفت و اولین مرحله خطلوله را انجام می‌دهد و نتیجه را ذخیره می‌کند، سپس مرحله بعد و الی آخر. CPU خطلوله را در زمان تقسیم می‌کند و تمام منابع پردازنده را به ترتیب به مراحل تخصیص می‌دهد. اما GPUها روش متفاوتی دارند؛ GPU منابع خود را به گونه‌ای بین مراحل مختلف تقسیم می‌کند که خطلوله در مکان تقسیم می‌شود نه در زمان. بخشی از پردازنده که روی یک مرحله کار می‌کند، خروجی خود را مستقیم به بخش متفاوتی می‌فرستد که روی مرحله بعدی کار می‌کند.

به دو دلیل ساختار این ماشین در حالت ثابت‌کاره بسیار موفق بود. اول آنکه سخت‌افزارهای هر مرحله می‌توانند از مزیت مستقل بودن داده‌های هر مرحله بهره ببرند و داده‌ها را موازی پردازش کنند. البته موازی‌سازی کار نیز امکان‌پذیر است و GPU را قادر می‌سازد که نیاز پردازش‌های عظیم خطلوله گرافیکی را برآورده کند و چندین مرحله را موازی انجام دهد. دوم آنکه سخت‌افزار هر مرحله می‌تواند با سخت‌افزار ثابت‌کاره‌ی مخصوص مرحله ساخته شود که موجب افزایش شدید سرعت پردازش می‌شود. مثال شاخص، واحد رسترایزر است که پوشش پیکسلی مثلث‌ها را بررسی می‌کند.

آنچه تا بدین جا بیان کردیم، مربوط به GPUهای پیش از معماری سایه‌زن یکپارچه است. در معماری‌های قدیمی GPUها، معادل هریک از مراحل خطلوله، واحدی در GPU وجود داشت که عملیات نظیر آن را انجام می‌داد. واحدهای سایه‌زن راس و سایه‌زن پیکسل تاحدودی قابل کنترل بودند ولی عملیات واحدهای دیگر کاملاً سخت‌افزاری انجام می‌شدند و ثابت‌کاره بودند. ازطرفی دیگر بار

¹ Task

² Task Parallelism

³ Data Parallelism

محاسباتی روی مراحل مختلف خطلوله یکسان نبود؛ بسته به ماهیت اشیای صحنه گرافیکی گاهی سایه‌زن راس، گلوگاه^۱ پردازش می‌شد و گاهی سایه‌زن پیکسل.

این مشکل با پیشرفت گرافیک کامپیوتری نمود بیشتری پیدا کرد؛ منابع سخت‌افزاری که در اختیار برنامه سایه‌زن راس و سایه‌زن پیکسل قرار می‌گرفت راضی‌کننده‌ی نیازها نبود و این ناشی از موازی‌سازی کار در GPU بود. این مشکل که با نام تعدیل بار^۲ شناخته می‌شود، وقتی به چشم می‌آمد که برای برنامه‌ای خاص، کد سایه‌زن راس بسیار پیچیده بود و کد سایه‌زن پیکسل بسیار ساده و یا بالعکس. در این حالت ALUهای یکی از سایه‌زن‌ها درگیر است و ALUهای سایه‌زن دیگر کم‌کار است و نمی‌تواند داده‌ی جدیدی پردازش کند و باید منتظر اتمام کار سایه‌زن مشغول شود. از آنجایی که مشابه تمامی خطلوله‌ها، کارایی خطلوله را کندترین مرحله آن تعیین می‌کند، تعدیل ناصحیح بار کارایی GPU را پایین می‌آورد.



تصویر ۱۲- بار گرافیکی نابرابر روی مراحل مختلف خطلوله باعث بیکار ماندن برخی از واحدها می‌شود. (تصویر از منبع

[11])

این دلایل طراحان را برآن داشت تا مجموعه دستورالعمل‌های معماری^۳ دو سایه‌زن را تا حد امکان یکی کنند و یک سایه‌زن کاملاً قابل برنامه‌ریزی توسعه دهند تا توزیع منابع خوبی انجام شود. همچنین این معماری جدید، از آنجایی که کاملاً قابل برنامه‌ریزی است، قابلیت پیاده‌سازی الگوریتم‌های دلخواه برای راس و پیکسل را دارد. این مفهوم با نام معماری سایه‌زن یکپارچه^۴ شناخته می‌شود و اولین

¹ Bottleneck

² Load Balancing

³ Instruction Set Architecture

⁴ Unified Shader

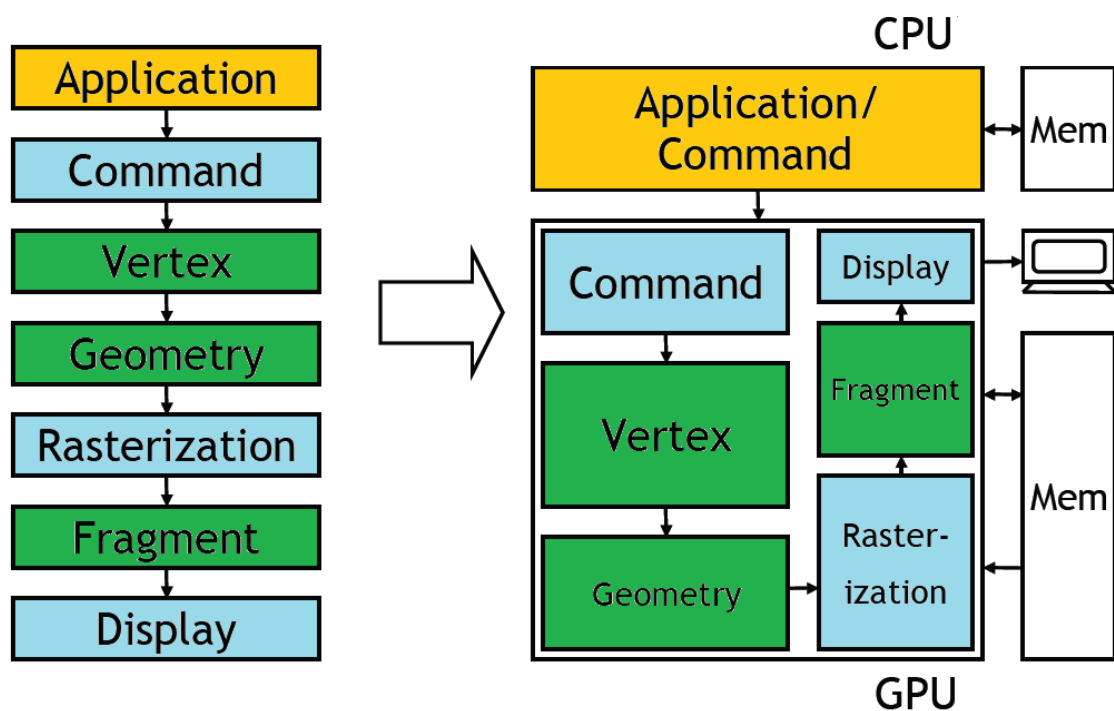
GPU ای که این معماری را پیاده‌سازی کرد، Xenos محصول AMD بود که در Xbox 360 استفاده شد. سایه‌زن یکپارچه از واحدهای قابل برنامه‌ریزی پرتعدادی تشکیل شده‌اند که هرکدام مجموعه دستورالعمل‌های لازم تمام سایه‌زن‌ها را دارند و قادر به اجرای کدهای آنها هستند.



تصویر ۱۳- معماری سایه‌زن یکپارچه، صرف نظر از برنامه سایه‌زنی راس و سایه‌زنی پیکسل، اتلاف سخت‌افزار را به حداقل می‌رساند. برنامه سایه‌زنی راس و سایه‌زنی پیکسل روی واحدهای مشابه اجرا می‌شوند و سخت‌افزار اختصاصی برای آنها وجود ندارد. (تصویر از منبع [11])

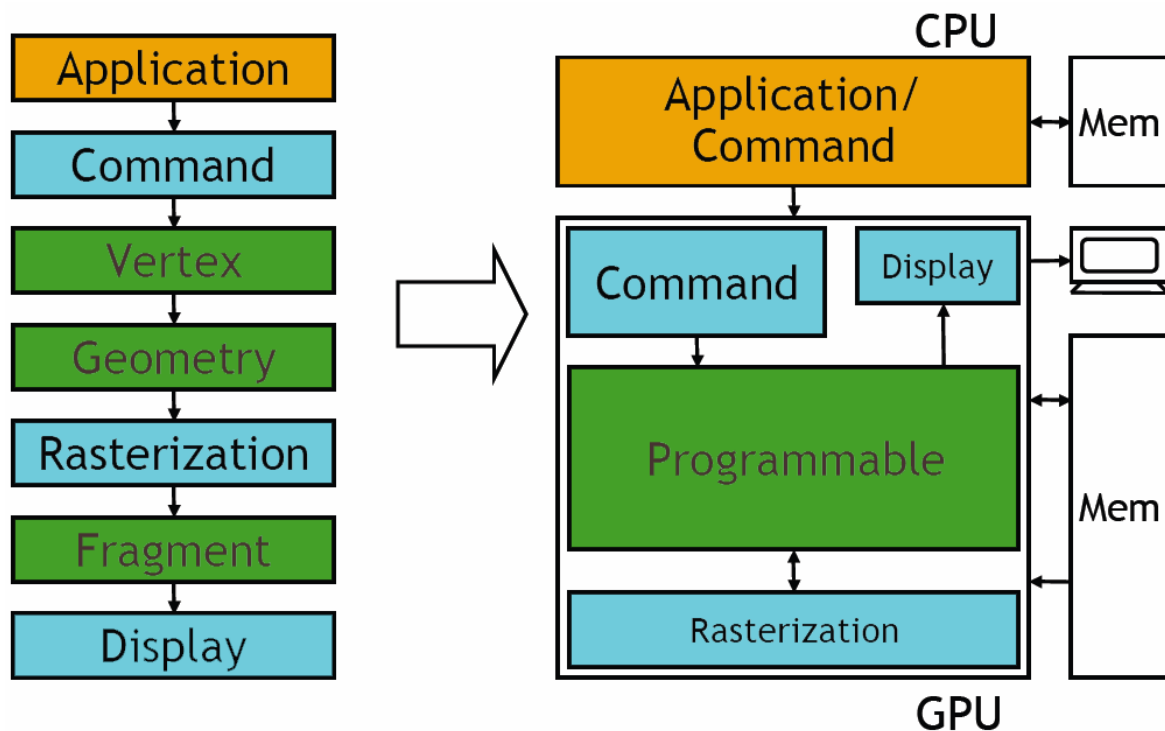
هنگامی که مراحل قابل برنامه‌ریزی جانشین مراحل ثابت‌کاره شدند، واحدهای تک‌منظوره‌ی ثابت‌کاره به‌سادگی جای خود را به واحدهای قابل برنامه‌ریزی دادند، اما موازی‌سازی کار و موازی‌سازی داده همواره وجود داشته است. این واحدهای قابل برنامه‌ریزی زمان خود را بین اجرای کار راس، کار پیکسل و کار هندسی (که بعدها اضافه شد) تقسیم می‌کنند. به علت اینکه بخش قابل برنامه‌ریزی خطلوله، عامل افزایش محاسبات خطلوله گرافیک است، معماری GPU درحال مهاجرت از «خطلوله‌ی صرف با موازی‌سازی کار» به یک «واحد یکپارچه برنامه‌پذیر با موازی‌سازی داده» است [9].

بهترین توصیف نسل‌های گذشته GPU ها، مقداری برنامه‌پذیری در خطلوله ثابت‌کاره است. در تصویر ۱۴ معماری متناظر خطلوله این GPU ها را مشاهده می‌کنید.



تصویر ۱۴-چپ مراحل خطلوله گرافیکی. راست سخت‌افزار اجرا کننده خطلوله. آبی: غیرقابل برنامه‌ریزی و ثابت کاره. سبز: قابل برنامه‌ریزی. در این تصویر حافظه GPU و CPU به‌خوبی با تفکیک از یکدیگر نمایش داده شده‌اند. (تصویر از منبع [۱۰])

طراحی GPU برای محاسبات موازی است که افزایش برون‌دهی مطرح است و برخلاف CPU، کاهش تاخیر هدف نیست. براین اساس معماری آنها به گونه متفاوتی پیشرفت کرد. بهترین توصیف از GPUهای امروزی آن است که بگوئیم موتوری برنامه‌پذیر است که با واحدهای ثابت‌کاره احاطه شده است. تصویر ۱۵ معماری این GPUها را نمایش می‌دهد.



تصویر ۱۵- معماری GPUهای مدرن که از یک سایه‌زن یکپارچه (که در تصویر Programmable نامیده شده) استفاده می‌کنند. این سایه‌زن قادر به اجرای دستورات سایه‌زن راس، سایه‌زن هندسی و سایه‌زن پیکسل است. شایان ذکر است که اساس واحد ثابت‌کاره رسترایزر همان است که در گذشته بود.

لازم است تعریفی از واحد ثابت‌کاره که پیش از این به آن اشاره کردیم را ارائه کنیم. این واحد قابل پیکربندی^۱ است ولی قابل برنامه‌ریزی نیست. بدین معنی که قادر به پیاده‌سازی الگوریتم‌های ثابتی، با پارامترهای معینی است و تنها می‌توان این واحد را برای انجام همین عملیات تنظیم کرد. قابل پیکربندی بودن بسیار دور از قابل برنامه‌ریزی بودن است که قابل برنامه‌ریزی بودن به معنای انعطاف‌پذیری کامل است. در معماری‌های بسیار قدیمی GPUها، تقریباً تمام مراحل خط‌لوله تنها قابل پیکربندی بودند؛ حتی سایه‌زن راس و پیکسل.

¹ Configurable

۳-۱ نسل‌های GPU

عمر کارت‌های گرافیکی تقریباً برابر عمر رایانه است. اولین کارت گرافیک در سال ۱۹۸۱ توسط IBM PC معرفی شد و ^۱CGA نام داشت. این اولین کارت گرافیکی رنگی و اولین کامپیوتر رنگی IBM بود. این کارت 16Kbyte حافظه و چندین وضعیت گرافیکی و متنی مختلف داشت. بالاترین اندازه تصویر ممکن این کارت 640*200 و حداکثر عمق رنگ آن ۴ بیت (۱۶ رنگ) است. این کارت تنها شامل یک ^۲RAMDAC و حافظه بود و تمامی پردازش‌ها (حتی رسترایز) توسط CPU انجام می‌شد و نتیجه از حافظه اصلی به حافظه صفحه نمایش کپی می‌شد. در سال ۱۹۸۴، ^۳EGA محصول جدیدی بود که با IBM AT معرفی شد و قابلیت نمایش ۱۶ رنگ و حداکثر اندازه تصویر 640*350 را داشت. در سال ۱۹۸۷ این کارت جای خود را به آخرین کارت گرافیک IBM تا به امروز داد که VGA نام داشت. این کارت با قبلی‌ها متفاوت بود و دارای واحد رسترایز داخلی بود. حافظه داخلی آن 256Kbyte و در دو وضعیت ۱۶ و ۲۵۶ رنگ کار می‌کرد.

اولین کارتی که قابلیت پردازش ^۳بعدی داشت، Matrox Impression بود که مخصوص بازار طراحی به کمک رایانه (PC CAD) ساخته شده بود. تا این زمان کارت گرافیک مخصوص سیستم‌های گران قیمت بود مشابه کارت‌های SGI که با زبان OpenGL قابل برنامه‌نویسی بودند. تا اینکه کارت Voodoo ساخته شد. وودو بنیان‌گذار نسلی بود که تحول اساسی در دنیای کارت گرافیک ایجاد کرد و مفهوم واحد پردازنده گرافیکی شکل گرفت. در این بخش معماری کلی و انتزاعی از نسل‌های مختلف GPUها ارائه می‌کنیم و هدف از این بخش آشنایی با تحولات معماری GPUها از وودو تا به امروز است.

۳-۱-۱ نسل اول (۱۹۹۶) - رسترایزر

یکی از اولین کارت‌های مستقلی که برای پردازش‌های گرافیکی ساخته شده بود، وودو نام داشت که محصول 3dfx بود. این کارت تنها مراحل رسترایز و ترکیب نهایی (در شکل با نام عملیات رستر^۴ مشخص شده است) را انجام می‌داد. مرحله سایه‌زنی راس بوسیله CPU انجام می‌شد و نتیجه آن از

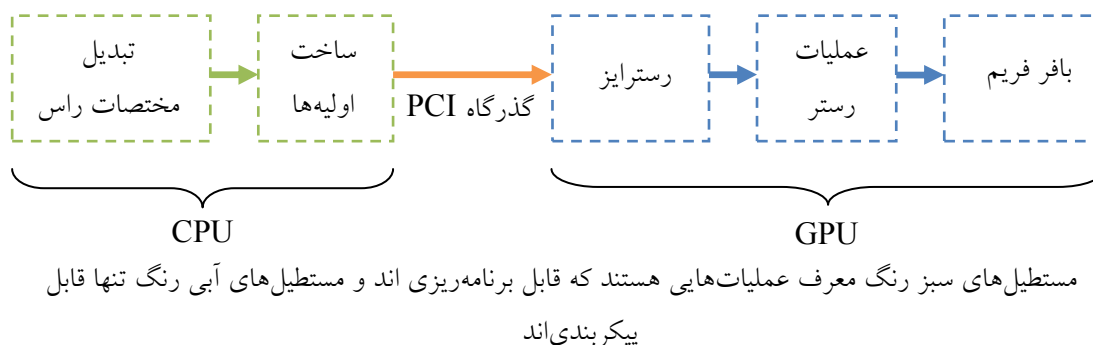
^۱ Color Graphics Adapter

^۲ Random Access Memory Digital-to-Analog Converter

^۳ Enhanced Graphic Adapter

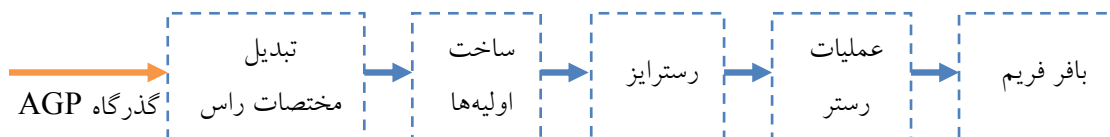
^۴ Raster Operation

طریق گذرگاه PCI به وودو فرستاده می‌شد تا باقی مراحل را GPU انجام دهد. حذف پیکسل‌های غیرقابل رویت با الگوریتم z-buffer انجام می‌شد. همان‌طوری که در تصویر زیر مشاهده می‌کنید، سایه‌زنی راس تنها محدود به تبدیل مختصات و تغییر دستگاه مختصاتی است و عملاً سایه‌زنی پیکسل ممکن نبوده است. تغییر مختصات و چرخاندن هر راس با ضرب ماتریسی انجام می‌شود.



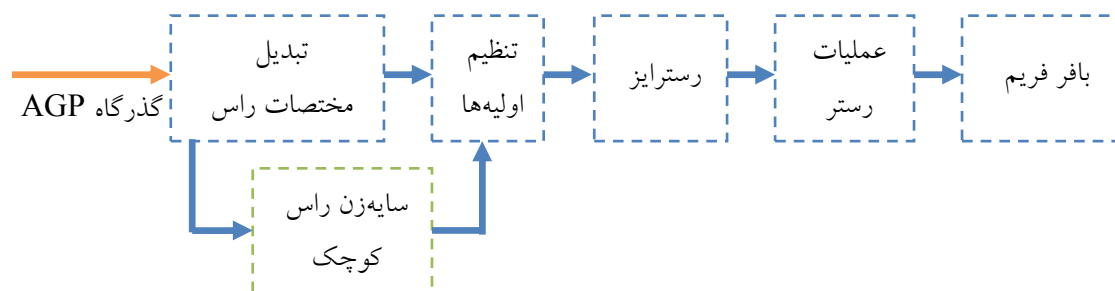
۱-۳-۲ نسل دوم (۱۹۹۸) - ثابت‌کاره

تفاوت اصلی با نسل قبلی، انتقال تبدیل‌کننده مختصات راس به داخل GPU بود که کار CPU را سبک می‌کرد. علاوه بر این GPUهای این نسل، بافت‌نگاری‌های چندسطحی را هم انجام می‌دادند. گذرگاه واسطه CPU و GPU با AGP جایگزین شد که سرعت بیشتری نسبت به PCI داشت. کارت‌های GeForce و Radeon7500 از این نسل هستند.



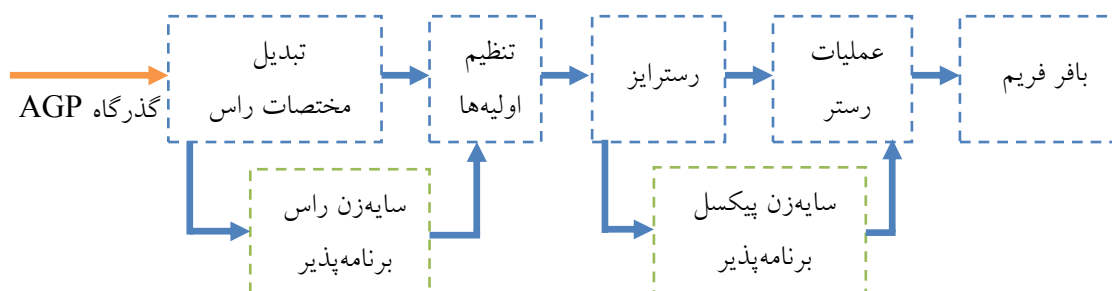
۱-۳-۳ نسل سوم (۲۰۰۱) - سایه‌زن راس برنامه‌پذیر

برای اولین بار، قابلیت برنامه‌نویسی در مرحله سایه‌زنی راس ممکن شد. عملیات ضد ناصافی نیز برای اولین بار پیاده‌سازی شد و در واحد رسترایزر قرار گرفت و پس از گسسته‌سازی (تبدیل مثلث‌ها به پیکسل) انجام می‌شد. کارت‌های GeForce3 و Radeon8500 از این نسل هستند.



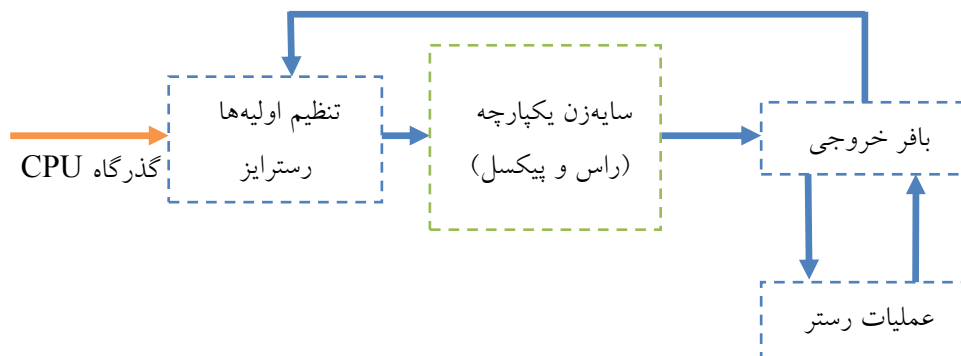
۱-۳-۴ نسل چهارم (۲۰۰۳) - سایه‌زن راس و سایه‌زن پیکسل برنامه‌پذیر

قابلیت برنامه‌نویسی پیکسل نیز به GPU اضافه شد و آن را بیش از پیش قابل برنامه‌ریزی کرد. در این سال‌ها GPUهای مختلف منابع و واحدها متفاوتی برای پردازش راس و پیکسل داشتند و این بود که کارایی آنها از هم فاصله گرفت. کارت‌های سری GeForce FX 5800 و Radeon 9800 از این نسل هستند. پشتیبانی از DirectX9 از ویژگی‌های GPUهای این نسل است.



۱-۳-۵ نسل پنجم (۲۰۰۵) - سایه‌زن یکپارچه

اواخر سال ۲۰۰۵ تحول عظیمی در معماری‌های GPU شکل گرفت و آن پیاده‌سازی سایه‌زن یکپارچه توسط GPU کنسول بازی Xbox360 بود. این GPU که Xenoss نام داشت، محصول ATI بود و بعدها معماری R600 را از روی آن توسعه دادند. زنوس بیان‌گذار سایه‌زن یکپارچه و نسل اول آن است که در آن برنامه‌های سایه‌زنی راس و پیکسل از واحدی یکسان استفاده می‌کنند. دیاگرام زیر معماری زنوس (با حذف جزئیات) را نمایش می‌دهد:

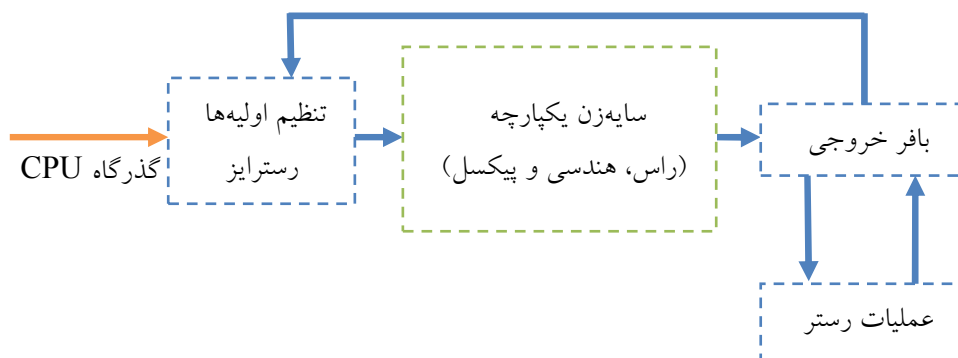


در زنوس بافر خروجی، سایه‌زن یکپارچه، تنظیم کننده اولیه‌ها و رسترایز در پل شمالی^۱ قرار داشتند و واحد عملیات رستر در چپ می‌مجا انجام می‌شد.

از سال ۲۰۰۵ تا ۲۰۰۷، رایانه‌های رومیزی در انتظار سایه‌زن یکپارچه بودند و در این سال‌ها تحولی در معماری GPUها شکل نمی‌گرفت. از جمله کارت‌هایی که در این سال‌ها عرضه شدند GeForce 6800 و Radeon X800 اشاره کرد. برخی از تحولاتی که در این سال‌ها شکل گرفتند، بهبود رسترایز، تکنیک‌های جدید ضدناصافی و افزایش دقت محاسبات بودند.

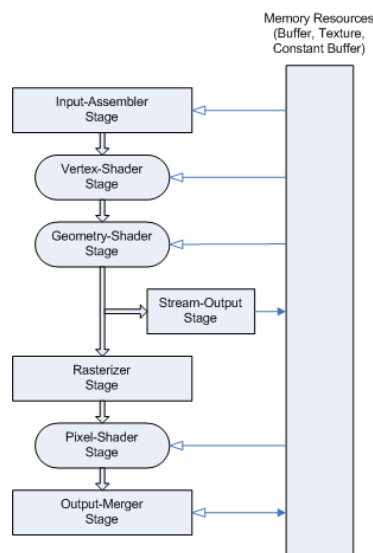
۱-۳-۶ نسل ششم (۲۰۰۶) - سایه‌زن راس، سایه‌زن هندسی و سایه‌زن پیکسل برنامه‌پذیر

کارت GeForce 8800 که سال ۲۰۰۶ عرضه شد، اولین کارت با معماری سایه‌زن یکپارچه برای رایانه‌های رومیزی بود. نام مستعار GPU این کارت، G80 بود و برخلاف زنون، واحد رستر را در کنار باقی واحدهای GPU روی یک چیپ داشت. سال ۲۰۰۷ GPU دیگری نیز ساخته شد که متعلق به همین نسل است. این GPU، R600 نام داشت و محصول ATI بود و از زنوس مشتق شده بود. R600 در ساخت کارت Radeon HD 2900 استفاده شد.



¹ North Bridge

کارت‌های این نسل از DirectX10 پشتیبانی می‌کنند که مرحله جدیدی به خطلوله گرافیکی اضافه کرد. این مرحله سایه‌زنی هندسی نام داشت و کاملاً قابل برنامه‌ریزی بود. همانطوری که در شکل زیر مشاهده می‌کنید، علاوه بر سایه‌زنی هندسی، مرحله خروج جریان^۱ نیز به خطلوله اضافه شده است که برنامه‌نویس را قادر می‌سازد چندین عملیات سایه‌زنی راس و یا سایه‌زنی هندسی متفاوت روی جریان ورودی انجام دهند. پیش از این ورودی هر مرحله خطلوله فقط خواندنی^۲ بود و خروجی آن فقط نوشتنی^۳. یعنی وقتی عملیات مرحله‌ای تمام می‌شد نمی‌توانستیم سایه‌زنی دوباره‌ای روی نتیجه انجام دهیم ولی خروج جریان این امکان را می‌دهد. علاوه بر این خروج جریان قابلیت همه‌منظوره کردن GPU را نیز افزایش می‌دهد. یعنی اینکه داده‌هایی غیر گرافیکی وارد خطلوله کنیم و محاسباتی نه الزاماً گرافیکی توسط سایه‌زن یکپارچه روی آنها انجام دهیم و پس از عبور بی‌تغییر از سایه‌زن هندسی، نتیجه را به حافظه بازگردانیم. این تکنیک که GPGPU^۴ نام دارد دنیای GPU ها را متحول کرد و باعث شد که GPU پردازنده‌ای مخصوص گرافیک نباشد و پردازنده‌ای همه‌منظوره شود.



تصویر ۱۶- خطلوله گرافیکی DirectX10. (عکس از msdn.microsoft.com)

^۱ Stream-Output Stage

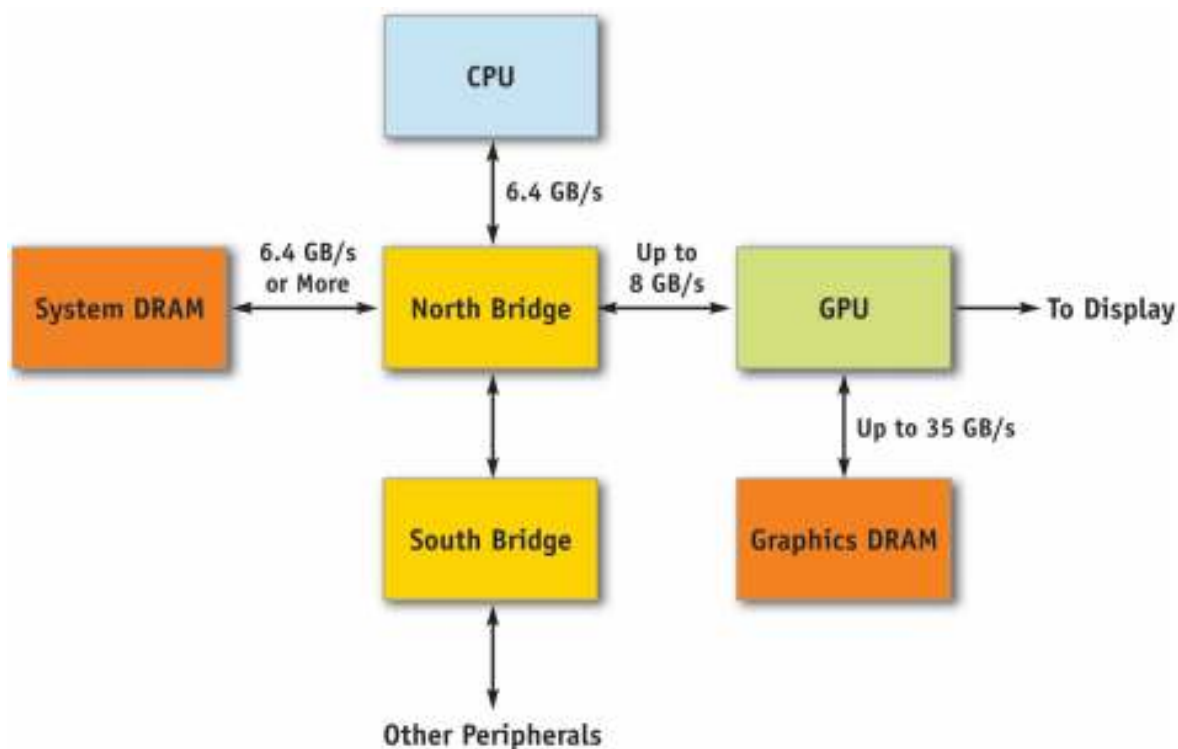
^۲ Read-Only

^۳ Write-Only

^۴ General-Purpose GPU

فصل دوم: معماری NV40

در این فصل معماری NV40 را بررسی می‌کنیم. این GPU در سال ۲۰۰۴ توسط شرکت انویدیا ساخته و در کارت گرافیک GeForce 6800 استفاده شد. NV40 متعلق به نسل چهارم GPU ها است و یکی از آخرین و پیشرفته‌ترین GPU های این نسل به حساب می‌آید. در تصویر ۱۷ معماری کامپیوتر و سیستمی که NV40 در آن بکار گرفته می‌شود، مشاهده می‌کنید:



تصویر ۱۷-پهنای باند بخش‌های مختلف معماری کامپیوتر NV40. پهنای باند ارتباط CPU و حافظه اصلی با فرض FSB 800MHz است. (تصویر از منبع [18])

همانند تمامی GPU های مدرن، NV40 حافظه مخصوص به خود را دارد که پهنای باند آن بسیار بالاست (GPU های آن‌بورد^۱ از حافظه سیستم استفاده می‌کنند و حافظه تخصصی ندارند). یکی از مهم‌تری نکات و مصالحه‌ها در طراحی GPU، پهنای باند حافظه و تناسب آن با سرعت پردازش داده‌ها است. ارتباط CPU با GPU از طریق AGP و یا PCI-Express برقرار می‌شود. سرعت این گذرگاه‌ها به اندازه کافی سریع است زیرا تنها رئوس و بافت‌ها (که حجم زیادی ندارند) هستند که باید از طریق

¹ Integrated GPU (IGP)

گذرگاه منتقل شوند. پهنای باند حافظه تخصصی GPU بسیار مهم است زیرا ماهیتی مثل پیکسل‌ها وجود دارند که تعداد دفعات دسترسی به آنها زیاد است. همانطور که در شکل تصویر ۱۷ مشاهده می‌کنید، این پهنای باند 35GB/s است.

عموما در GPUها، هزاران نخ همزمان در حال اجرا هستند و هرکدام نیاز به دسترسی به حافظه دارند. طراحی سلسله مراتب حافظه و حافظه نهان به افزایش کارایی GPU بسیار کمک می‌کند ولی افزایش حافظه نهان به اندازه‌ای که درصد برخورد^۱ را برای CPU بالا می‌برد، به GPU کمک نمی‌کند. علت این است که ماهیت داده‌هایی که GPU با آنها سروکار دارد به گونه‌ای است که دسترسی به یک داده چندین بار تکرار نمی‌شود. با تعریف نظریه محلیت^۲، می‌توان گفت که محلیت در زمان^۳ خیلی به کارایی GPU کمک نمی‌کند و محلیت در مکان^۴ است که درصد برخورد حافظه نهان GPU را ۹۰ درصد نگه می‌دارد (نقل از منبع [16]). (محلیت در زمان: اگر به داده‌ای رجوع شد، احتمالا دوباره بزودی به آن رجوع می‌شود. محلیت در مکان: اگر به داده‌ای رجوع شد، احتمالا به آدرسی‌هایی که در نزدیکی آن قرار دارند نیز رجوع می‌شود. (منبع [17]))

برای کاهش تاخیر حافظه، NV40 حافظه را به چهار بخش مستقل تقسیم کرده است و برای هر بخش یک کنترل کننده ۶۴ پین دارد که این کنترل کننده‌ها برای استفاده موثر از پهنای باند، فشردگی و گسترده‌سازی داده‌ها انجام می‌دهند که از دید کاربر پنهان است. این کنترل کننده‌ها عملیات صفحه‌بندی و ترجمه آدرس خطی را نیز انجام می‌دهند.

از لحاظ قابلیت برنامه‌ریزی، NV40 قادر به اجرای دو برنامه مجزا است: یکی برنامه راس که روی تک‌تک رئوس اجرا می‌شود و دیگری برنامه پیکسل که روی تک‌تک پیکسل‌ها اجرا می‌شود. سیستم وظیفه دارد که نخ^۵ برنامه راس را روی جریان ورودی رئوس اجرا کند. در ضمن وظیفه انتقال خروجی رسترایزر به ورودی برنامه پیکسل را هم دارد. مدیریت اجرای این نخ‌ها تماما دور از دید برنامه‌نویس انجام می‌شود. هر نخ، ورودی‌های منحصر به فردی دارد که از طریق ثبات‌های فقط خواندنی به آنها دسترسی می‌یابد. هر نخ، خروجی و نتیجه را از طریق ثبات‌های فقط نوشتنی به مرحله بعدی خط‌لوله

¹ Hit-Ratio

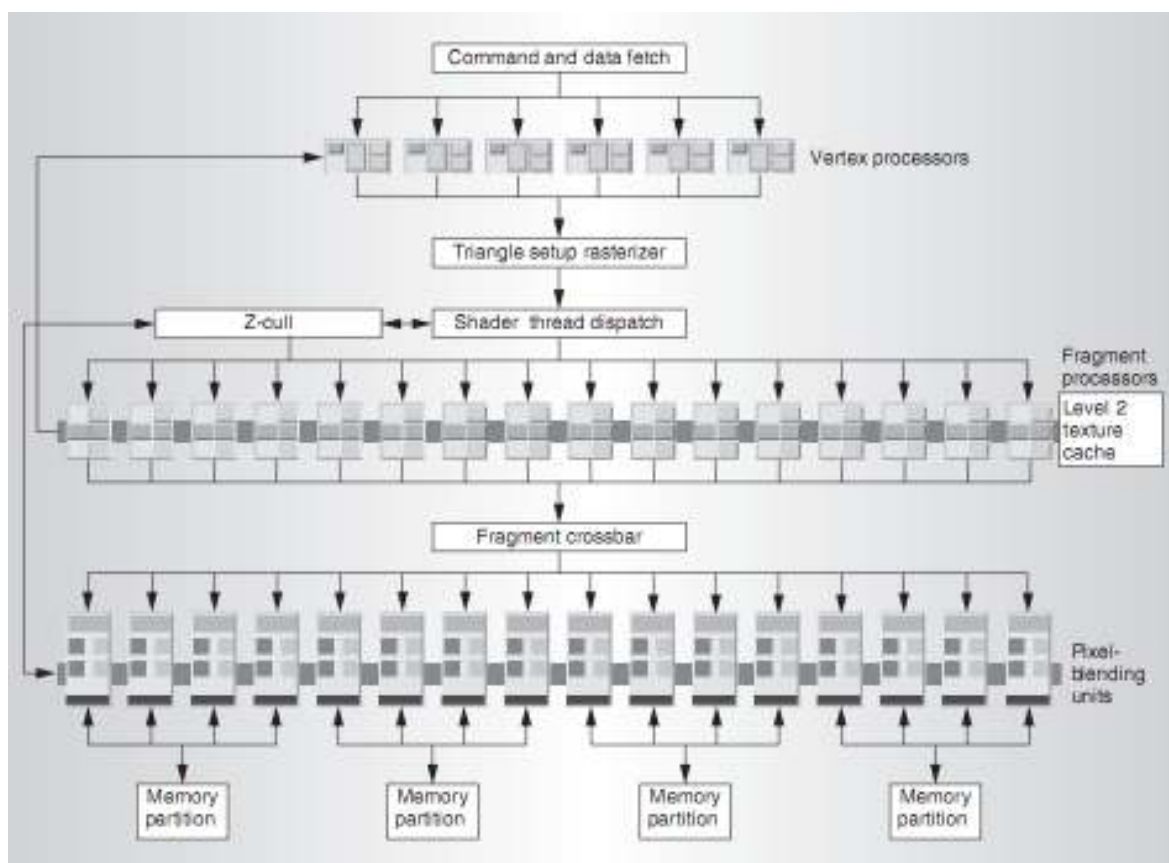
² Principle of Locality

³ Locality in Time

⁴ Locality in Space

⁵ Thread

می‌فرستد. علاوه بر این هر نخ به ثابت‌های خصوصی موقت^۱، پارامترهای ورودی برنامه (که فقط خواندنی هستند) و تصویر نقشه بافت^۲ نیز دسترسی دارد. درحالی دسترسی به بافت‌ها در هنگام سایه‌زنی راس امکان‌پذیر شده که پیش از این دسترسی به بافت‌ها تنها در مرحله سایه‌زنی پیکسل ممکن بود. توضیحات بالا مقدمه‌ای بود که وارد دنیای NV40 شویم. در تصویر ۱۸ نمایی از معماری این GPU را مشاهده می‌کنید:



تصویر ۱۸-دیاگرام بلوکی NV40. داده‌ها از بالا به پایین در خطلوله پردازش می‌شوند و پس از طی کردن مراحل آن،

نتیجه در حافظه ذخیره می‌شود. (تصویر از منبع [16])

در بالای شکل فوق، ۶ سایه‌زن راس مشابه و قابل برنامه‌ریزی قرار دارند. از آنجایی که رئوس مستقل از یکدیگرند، data fetcher آزاد است که هر راس را به هر پردازنده راس بیکار بسپارد. نتیجه مرحله راس به واحد Tringle setup rasterizer می‌رود. برای هر اولیه، رسترایزر پیکسل‌های سازنده را

¹ Private Temporary Register

² Texture map

مشخص می‌کند و به پردازنده فرگمنت می‌فرستد. این پردازنده‌های فرگمنت وظیفه سایه‌زنی پیکسل‌ها را دارند. ۱۶ سایه‌زن پیکسل قابل برنامه‌ریزی روی پیکسل‌ها به صورت موازی کار می‌کنند. هر نخ سایه‌زن پیکسل، مختصات (x,y) و رنگ درونیایی شده توسط رسترایزر را دریافت می‌کند. در پایان crossbar عمق و رنگ نتیجه‌ی پردازنده‌ی فرگمنت را برای ترکیب نهایی به ۱۶ واحد ترکیب پیکسل^۱ می‌فرستد. این واحدها وظیفه عملیات بافر فریمی مثل ترکیب رنگ و ضدناصافی را برعهده دارند.

۱-۲ پردازنده رئوس

NV40 ۶ پردازنده راس (سایه‌زن راس) دارد. معماری این پردازنده VLIW^۲ است و از این رو چندین دستور اجرا می‌شوند و کامپایلر عدم وابستگی این دستورات به هم را تضمین می‌کند (در پردازنده‌های VLIW، هر دستور از چند بخش تشکیل شده است که به ALU های مختلف پردازنده (ALU ممیزشناور، ALU صحیح) دستورات متفاوتی را در یک سیکل ارسال می‌کند). دستورات ۱۲۳ بیتی هستند و از Instruction RAM واکشی می‌شوند که ظرفیت ذخیره ۵۱۲ دستورالعمل را دارد. مسیر داده هر پردازنده واحد ضرب-جمع برداری^۳، واحد توابع خاص اسکالر^۴ و واحد بافت راس^۵ را دربردارد. هر پردازنده راس برای آنکه تاخیر ناشی از دستیابی به حافظه را جبران کند، ۳ نخ همزمان اجرا می‌کند.

در تصویر ۱۹ معماری داخلی یکی از این پردازنده‌ها را ملاحظه می‌کنید:

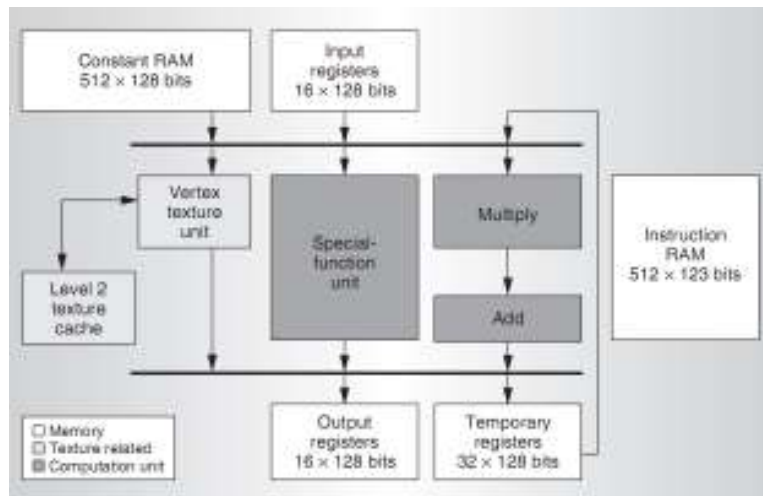
^۱ Pixel-Blending Unit

^۲ Very Large Instruction Word

^۳ Vector multiply-add unit

^۴ Scalar special-function unit

^۵ Vertex Texture Unit



تصویر ۱۹- مسیر داده پردازنده راس (تصویر از منبع [16])

واحد ضرب-جمع برداری دارای معماری $SIMD^1$ با پهنای ۴ است و تنها قادر به کار کردن روی داده‌های ممیز شناور IEEE دقت ساده ۳۲ بیتی است. (بدین ترتیب ورودی داده‌ای این واحد برداری 4×32 بیت یا ۱۲۸ بیت است) واحد برداری علاوه بر انجام ضرب، جمع، و ضرب-جمع، قادر به انجام ضرب داخلی، ماکزیمم و مینیمم یابی نیز هست.

واحد توابع خاص عددی مشابه واحد برداری روی ممیزشناور ۳۲ بیتی کار می‌کند و برای محاسبه توابع ریاضی سینوس، کسینوس، لگاریتم، نمایی، جذر و $1/x$ طراحی شده‌است. واحدهای محاسباتی عملوندهای خود را از 512*128bit Constant RAM، 32*128bit Temporary Registers یا 16*128bit Input Register واکشی می‌کنند و نتیجه را در ثبات‌های موقت یا 16*128bit Output Register ذخیره می‌کنند.

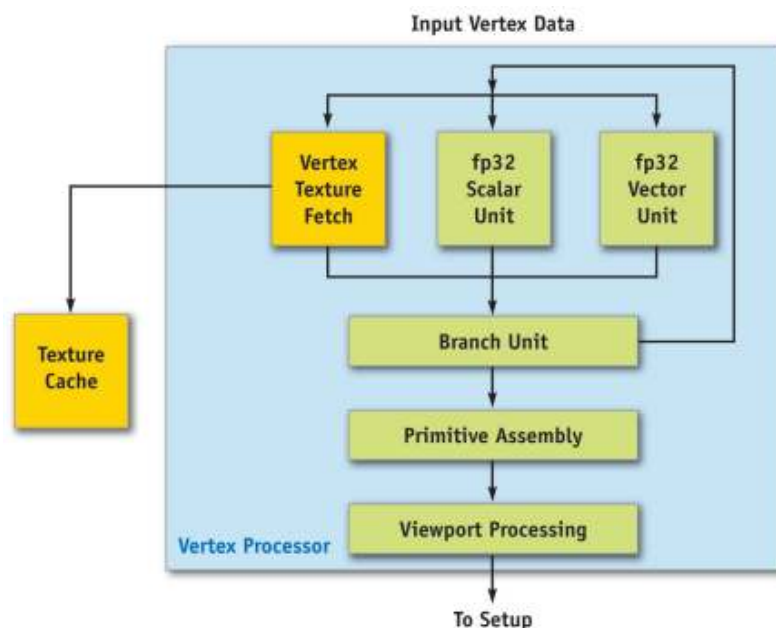
معماری پردازنده‌های راس $MIMD^2$ است زیرا هر کدام از آنها یک پردازنده SIMD هستند و وقتی کنار هم کار کنند هر کدام دستورالعملی متفاوت روی داده‌هایی متفاوت اجرا خواهند کرد. هر پردازنده راس در هر سیکل ساعت، می‌تواند همزمان دستوراتی به هر یک از دو واحد برداری و عددی ارسال کند.

همانطور که در **Error! Reference source not found.** مشاهده می‌شود، واحد بافت راس با حافظه نهان بافت در ارتباط است، این حافظه نهان بین پردازنده راس و پردازنده فرگمنت مشترک است.

¹ Single Instruction Multiple Data

² Multiple Instruction Multiple Data

واحد بافت راس، قادر به انجام محاسبات روی داده‌های صحیح و ممیزشناور دقت ساده است. این واحد، دسترسی پردازنده راس به بافت‌ها را ممکن می‌سازد. واحد بافت قادر به دسترسی به حداکثر ۴ بافت است و البته به هر بافت می‌توان چندین بار دستیابی پیدا کرد.



تصویر ۲۰- نگاهی کامل ولی با جزئیات کمتر به دیاگرام بلوکی پردازنده راس. (تصویر از منبع [18])

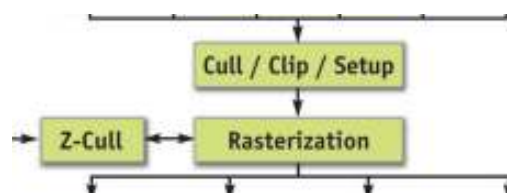
همانطوری که در تصویر ۲۰ مشاهده می‌کنید، یکی از قابلیت‌های این پردازنده امکان کنترل پرش است. این واحد وقتی بکار می‌آید که لازم است که یک شی، چندین بار در موقعیت‌های مختلف پردازش شود (مانند آدم‌های درون یک استادیوم، چمن یا تعدادی سرباز) و اصطلاحاً از روی آن نمونه‌سازی^۱ شود. در این تصویر Texture Cache بخوبی خارج از پردازنده راس نشان داده شده است.

۲-۲ تنظیم اولیه و رسترایزر

به کمک توابع واسط موجود در API^۲ های گرافیکی (DirectX یا OpenGL)، عملیات‌های مختلف بین پردازنده راس و پردازنده فرگمنت را می‌توان کنترل کرد. این عملیات قابل برنامه‌ریزی نیستند و تنها قابل پیکربندی اند و با واحدهای ثابت‌کاره انجام می‌شوند.

^۱ Instancing

^۲ Application Programming Interface



تصویر ۲۱- واحدهای ثابت کاره ماقبل از سایه‌زنی پیکسل (تصویر از منبع [18])

همانطور که در تصویر ۲۱ نشان داده شده، رئوس حاصل از سایه‌زنی به واحد ساخت اولیه می‌رسند که نقطه، خط یا مثلث‌ها را تشکیل می‌دهد.

پیش از اینکه اولیه‌ها وارد رسترایز شوند، انتخاب^۱ و برش^۲ روی آنها انجام می‌شوند. انتخاب اولیه‌هایی که از زاویه‌ی دید جاری قابل رویت نیستند را از جریان اولیه‌ها حذف می‌کند و برش اولیه‌هایی را که بخشی از آنها با زاویه‌ی دید اشتراک دارد برش می‌دهد و بخش‌های غیرقابل رویت آنها را حذف می‌کند. انتخاب و برش حجم پردازش را بسیار کاهش می‌دهند؛ اولیه‌هایی که دیده نمی‌شوند لازم نیست به پیکسل‌ها تجزیه شوند و باقی مراحل خطلوله را طی کنند.

سپس واحد Setup ضرابی را تولید که لبه‌های اولیه را توصیف می‌کند. در پایان رسترایز اولیه‌ها را به پیکسل‌ها تبدیل می‌کند تا آماده پردازش در پردازنده فرگمنت شوند. گفتنی است که رسترایز با ترتیب Page-friendly این کار را انجام می‌دهد.



تصویر ۲۲- ترتیب رستر برای یافتن پیکسل‌های درون اولیه (تصویر از منبع [16])

¹ Cull

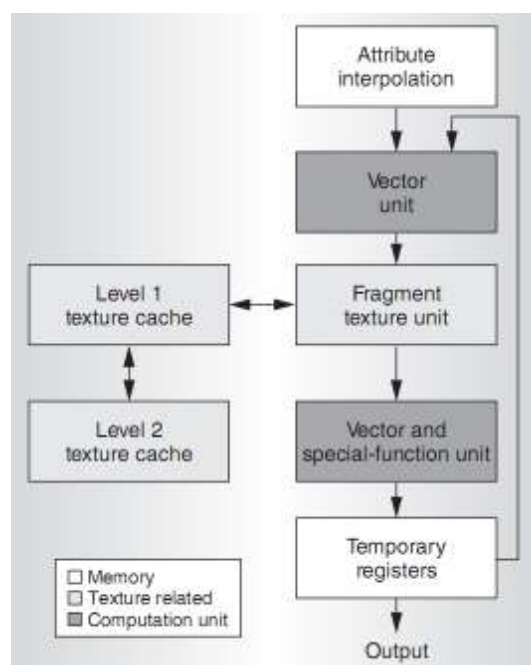
² Clip

خروجی رسترایزر در دسترس Z-Cull قرار می‌گیرد تا پیکسل‌های غیرقابل رویت را حذف کند. همانند انتخاب و برش، Z-Cull نیز نقش بسزایی در افزایش کارایی دارد.

۳-۲ پردازنده فرگمنت

پیش از اینکه پیکسل‌های خارج شده از رسترایزر، وارد پردازنده فرگمنت شوند، باید درونیابی خصوصیات آنها نیز انجام شود. این درونیابی به کمک خصوصیت‌های رئوس انجام می‌شود. توجه کنید که اگر عملیات بافت در مرحله سایه‌زنی راس انجام شود، اطلاعات بافت به خصوصیات رئوس اضافه می‌شود و پس از رسترایزر این اطلاعات بافت نیز برای پیکسل‌ها باید درونیابی شوند.

پردازنده فرگمنت NV40 قادر به انجام عملیات ممیز شناور ۱۶ و ۳۲ بیتی است. هر پیکسلی که وارد پردازنده فرگمنت می‌شود باید خصوصیت‌های مکان، رنگ، عمق و مه^۱ را داشته باشد (مه جزو خصوصیت‌هایی است که برای ایجاد مه و واقعی‌تر کردن تصویر استفاده می‌شود). پردازنده فرگمنت که برای سایه‌زنی پیکسل‌ها ساخته شده، ویژگی‌هایی مشابه به پردازنده راس دارد؛ از جمله انشعاب و پرش، ثبات‌های موقت، ثبات‌های ثابت و واحدهای برداری.



تصویر ۲۳-دیاگرام بلوکی پردازنده فرگمنت (تصویر از منبع [16])

¹ Fog

NV40 جمعاً ۱۶ پردازنده فرگمنت دارد، هر ۴ پردازنده فرگمنت یک واحد چهارپیکسلی^۱ را تشکیل می‌دهند که این چهار پیکسلی یک پردازنده SIMD است. هریک از ۱۶ پردازنده فرگمنت دارای دو واحد ریاضی برداری (با تفاوت اندک)، واحد توابع خاص و نرمالایزر و واحد بافت هستند. پهنای واحدهای برداری مشابه پردازنده راس، ۴ است (یک دستور را روی ۴ ورودی متمایز ۳۲ بیتی می‌تواند اجرا کند و نتیجه را در ۴ خروجی ذخیره نماید). همانطور که در تصویر ۲۳ مشاهده می‌شود، دو واحد برداری وجود دارند. تنها واحد پایینی قادر به انجام دستورالعمل ضرب-جمع است. واحد پایینی قادر به محاسبه همان توابع خاص پردازنده راس نیز هست (کسینوس، سینوس، لگاریتم و نمایی). از آنجایی که واحد بافت فرگمنت گاهی نیاز به دسترسی به DRAM برای واکنشی عملوندهایش دارد، ممکن است تاخیر زیاد دسترسی به حافظه کارایی پردازنده فرگمنت را پایین آورد. از این رو برای مخفی کردن تاخیر^۲، پردازنده فرگمنت از تکنیک چندنخی استفاده می‌کند؛ صدها نخ را همزمان نگاه می‌دارد و هرکدام که عملوندهایش رسیده باشد را اجرا می‌کند.

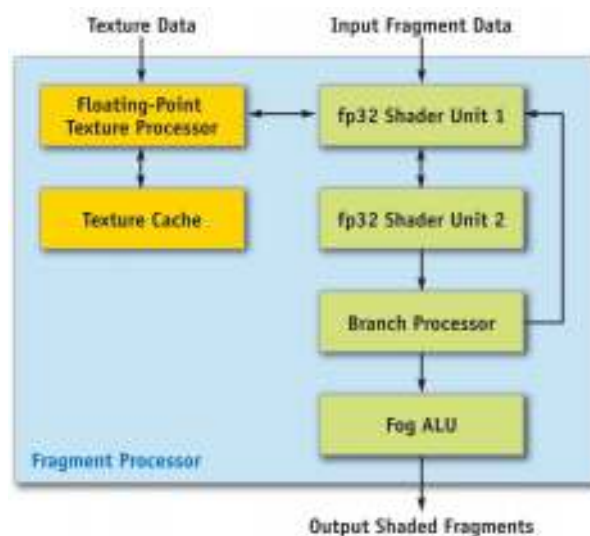
همانطور که در تصویر ۲۴ مشاهده می‌کنید، محاسبه مه در آخرین مرحله و پس از اتمام سایه‌زنی پیکسل انجام می‌شود. این کار با معادله ساده‌ی زیر انجام می‌شود:

$$(\text{درصد مه} - 1) * \text{رنگ مبدا} + \text{درصد مه} * \text{رنگ مه} = \text{رنگ خروجی}$$

در اکثر عملیات‌های سایه‌زنی، چهار خصوصیت قرمزی، سبزی، آبی و شفافیت رنگ پیکسل‌ها تحت عملی یکسان پردازش می‌کند و به این علت است که واحدهای برداری عملیات روی ۴ ورودی متفاوت انجام می‌دهد و پهنای SIMD آنها ۴ است. اما در برخی از پردازش‌های پیکسل، لازم است خصوصیت‌ها، مجزا پردازش شوند (مثلاً شفافیت مجزای RGB پردازش می‌شود).

^۱ Quad-Pixel Unit

^۲ Hide the latency

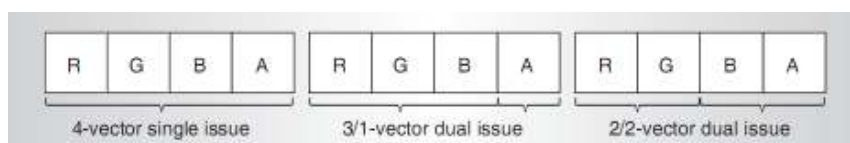


تصویر ۲۴-پردازنده فرگمنت

به این علت است که Shader Unit 1 و Shader Unit 2 به دسته‌بندی مجهز شده است. هرکدام از این دو واحد قادرند دو دستور همزمان اجرا کند:

یک دستور روی خصوصیت شفافیت و یک دستور روی سه خصوصیت دیگر
و یا اینکه یک دستور روی شفافیت و آبی و یک دستور روی قرمزی و سبزی.

در تصویر ۲۵ گونه‌های مختلف صادر کردن دستورات و استفاده از پهنای SIMD واحدهای Shader را مشاهده می‌کنید. با استفاده از این قابلیت (که Superscalar به آن می‌گویند) می‌توان دو دستور را همزمان روی زیر مجموعه‌ای از ۴ بردار صادر کرد.

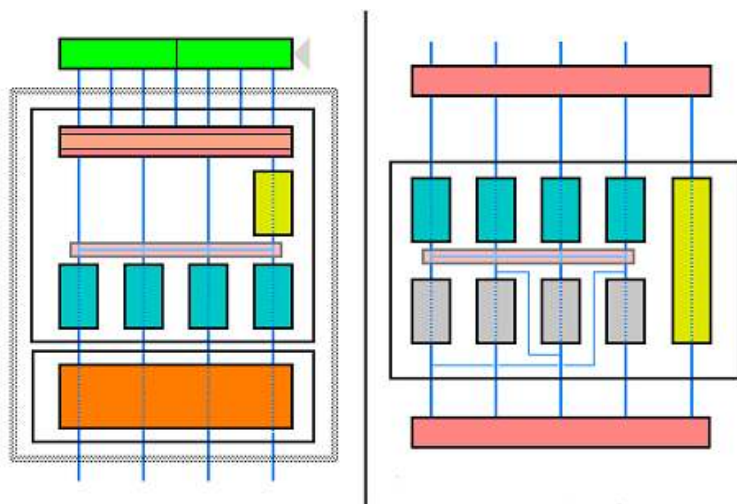


تصویر ۲۵-گزینه‌های پردازش برداری (تصویر از منبع [16])

توجه کنید که هرکدام از دو Shader، می‌توانند نهایتاً در هر سیکل ساعت دو دستور اجرا کنند که جمعا ۴ دستور اجرا می‌کنند.

زیرسیستم بافت در NV40 پیش از بازگرداندن نتیجه بافت‌نگاری به پردازنده فرگمنت باید نتیجه را فیلتر کند. در بخش (۱-۷-بافت‌نگاری) انواع فیلترکردن بافت را بررسی کردیم. روش‌های فیلترکردن نقطه، فیلترکردن دوخطی، فیلترکردن سه‌خطی و ناهمجهت تا نسبت 16:1 پشتیبانی شده‌اند. این فیلترکردن‌ها ثابت کاره هستند و توسط API‌های گرافیکی قابل تنظیم و پیکربندی‌اند. کار واحد بافت برگرداندن نمونه مناسبی از بافت است که مات شدگی تصویر نهایی را حداقل کند.

برگردیم به Shader Unit ها و جزئیات بیشتری درباره آنها مطرح کنیم. در تصویر ۲۶ بلوک دیاگرام این واحدها را مشاهده می کنید.



تصویر ۲۶- Shader Unit 1 سمت چپ و Shader Unit 2 سمت راست (تصویر از منبع [22])

مستطیل‌های فیروزه‌ای ضرب‌کننده، خاکستری جمع‌کننده و زرد واحد توابع خاص (سینوس، کسینوس و ...) هستند. مستطیل نارنجی رنگ عملیات بافت انجام می‌دهد و مخصوص محاسبات ¹ LOD است و خروجی آن ورودی واحد بافت است (محاسبات LOD حذف کردن برخی اطلاعات است که تاثیر ناچیزی در نمایش نهایی دارند ولی حذف آنها باعث افزایش سرعت چشمگیر عملیات رندر می‌شود).

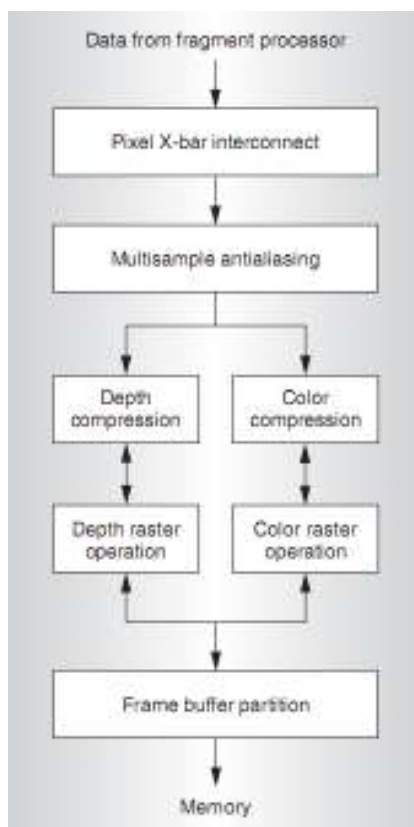
SU1 می‌تواند دو عملیات در هر سیکل انجام دهد؛ دو ضرب ۲ بعدی، یک ضرب ۳ بعدی و یک تابع خاص، یک تابع خاص و یک عملیات بافت. ضمناً SU1 می‌تواند داده را بی‌تغییر عبور دهد. SU2 نیز می‌تواند دو عملیات در هر سیکل انجام دهد یا اینکه داده را بی‌تغییر عبور دهد. عملیات‌های SU2 منحصر به ضرب نیست و چون جمع‌کننده دارد، می‌تواند جمع یا ضرب-جمع هم انجام دهد. خط‌های آبی رنگ اطراف جمع‌کننده‌ها معرف مسیر داده‌ای است که ضرب نقطه‌ای^۲ دو بردار در یک سیکل را ممکن می‌سازد. در SU2 پنج ورودی به واحدها رسیده‌اند ولی بسته به دستورالعمل یکی از این ۵ ورودی غیرفعال می‌شود.

¹ Level of Detail

² Dot product

۴-۲ واحد ترکیب پیکسل

۱۶ ترکیب کننده پیکسل در NV40 وجود دارد که وظیفه ترکیب نهایی در خطلوله گرافیکی را دارند و در پایان نتیجه را در بافر فریم می نویسند. این واحدها ثابت کاره هستند و ضدناصافی، سنجش عمق، سنجش استنسیل و ترکیب رنگ را با نرخ ۱۶ پیکسل در سیکل ساعت انجام می دهند. ترکیب رنگ با داده های ممیزشناور ۱۶ بیتی انجام می شود و برای کاهش تقاضای پهنای باند، عمق و رنگها فشرده سازی می شوند.



تصویر ۲۷-دیاگرام بلوکی واحد ترکیب پیکسل

برای افزایش سرعت دسترسی، هر واحد ترکیب کننده پیکسل به بخش خاصی از حافظه متصل شده است. برای انجام عملیات ضدناصافی، هر واحد قادر است که عمق و رنگ هر پیکسل را به چند نمونه بسط دهد. هر وقت ممکن باشد (از آنجایی که فشرده سازی بی کاستی^۱ است گاهی فشرده سازی ممکن

^۱ Lossless

نیست)، واحدها عمق و رنگ را فشرده می‌کنند (توسط بخش‌های فشرده‌سازی عمق و رنگ که در مشخص است) و سپس واحدهای عمق و رنگ به بخش حافظه محلی خود دسترسی پیدا می‌کنند و عملیات عمق و استنسیل و ترکیب رنگ را انجام می‌دهند.

هر دو روش ناصافی رایج یعنی چندنمونه‌برداری و ابرنمونه‌برداری توسط این واحد انجام می‌شود. ۲، ۴ یا ۸ نمونه برای هر پیکسل تولید می‌شود و میانگین این نمونه‌ها رنگ نهایی را تولید می‌کند. چندنمونه‌برداری بار کمتری بردوش پردازنده فرگمنت می‌گذارد ولی ابرنمونه‌برداری بار پردازنده فرگمنت را چندین برابر می‌کند.

۲-۵ سنجش کارایی

برای سنجش کارایی NV40 شمارش تعداد عملیات‌های ممیزشناور در ثانیه^۱ هر واحد قابل برنامه‌ریزی، معیار خوبی است. واحدهای قابل برنامه‌ریزی، پردازنده‌های راس و پردازنده‌های فرگمنت هستند.

تعداد عملیات ممیزشناوری که پردازنده راس در بهترین حالت انجام می‌شود، ۵ عمل است؛ ۴ عمل بوسیله واحد برداری و ۱ عمل بوسیله واحد عملیات خاص انجام می‌شود. از آنجایی که NV40، ۶ پردازنده راس دارد، $6 * 5 = 30$ عمل ممیزشناور بوسیله مجموعه پردازنده‌های راس انجام می‌شود.

$$\text{VertexProcessorsOps} = 6 * (\text{VectorUnitOps}(4) + \text{SpecialFuncUnitOps}(1)) = 30$$

تعداد عملیات ممیزشناوری که بوسیله پردازنده فرگمنت انجام می‌شود، ۸ عمل است؛ ۴ عمل بوسیله SU1 و ۴ عمل بوسیله SU2 انجام می‌شود. جمعا ۱۶ پردازنده فرگمنت وجود دارد پس $16 * 8 = 128$ عمل توسط پردازنده‌های راس انجام می‌شود.

$$\text{FragmentProcessorsOps} = 16 * (\text{ShaderUnit1Ops}(4) + \text{ShaderUnit2Ops}(4)) = 128$$

کارایی کلی NV40 از رابطه زیر محاسبه می‌شود:

$$\text{TotalFLOPs} = (\text{VertexProcessorsOps} + \text{FragmentProcessorsOps}) * \text{ClockFreq}$$

و چون کلاک پردازنده‌های گرافیکی 425MHz است، کارایی کلی 67.150 GFLOPs می‌شود.

¹ Floating Operation Per Second (FLOPs)

در محاسبه GFLOPs، عمل ضرب-جمع که در یک سیکل انجام می‌شود، دو عمل بحساب می‌آید. اگر این فرض را در نظر بگیریم، عملیات واحد برداری پردازنده راس و SU2 باید در ۲ ضرب شود و داریم:

$$\text{VertexProcessorsOps} = 6 * (2 * \text{VectorUnitOps}(4) + \text{SpecialFuncUnitOps}(1)) = 54$$

$$\text{FragmentProcessorsOps} = 16 * (\text{ShaderUnit1Ops}(4) + 2 * \text{ShaderUnit2Ops}(4)) = 192$$

$$\text{TotalFLOPs} = (\text{VertexProcessorsOps} + \text{FragmentProcessorsOps}) * \text{ClockFreq}$$

بدین ترتیب با همان کلاک قبلی، کارایی 104.55 GFLOPs محاسبه می‌شود.

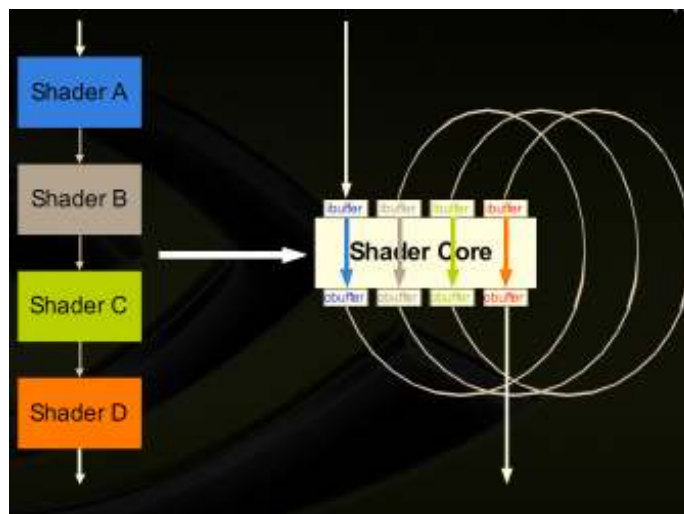
فصل سوم: معماری G80

اولین GPU با سایه‌زن‌های یکپارچه برای رایانه‌های شخصی در سال ۲۰۰۶ با کارت GeForce 8800 معرفی شد. این GPU که G80 نام دارد، سایه‌زن راس و پیکسل را یکی کرد و آنقدر درجه همه‌منظوره بودن در آن بالا است که قابل برنامه‌نویسی با زبان C از طریق مدل کودا^۱ است.

در فصل قبل معماری NV40 را بررسی کردیم. پردازنده راس برای محاسبات ریاضی (تاخیر کم و دقت زیاد) طراحی شد در حالی که پردازنده فرگمنت برای فیلترکردن بافت (تاخیر زیاد و دقت کمتر) طراحی شد. در گذشته دور، سایه‌زنی راس محاسبات پیچیده‌تری داشت و به‌همین علت قبل از سایه‌زنی پیکسل برنامه‌پذیر شد. اما از سال ۲۰۰۲ به بعد، برای پاسخ به نیاز عمومیت برنامه‌ریزی (پیاده‌سازی الگوریتم‌های متفاوت سایه‌زنی)، پردازنده راس و پردازنده فرگمنت بیش از پیش مشابه شدند. از طرف دیگر چون در عملیات رندر، تعداد پیکسل‌ها بیشتر از رئوس است، عموماً GPUها پردازنده‌های فرگمنت بیشتری از پردازنده‌های راس داشتند (حدود دو یا سه برابر). این درحالی است که توزیع بار متعادل نیست و منجر به کاهش کارایی می‌شود. مثلاً با مثلث‌های بزرگ پردازنده راس بیکار می‌ماند و پردازنده فرگمنت بسیار مشغول می‌شود. با مثلث‌ها کوچک، عکس همین صادق است. به‌همین دلیل پیدا کردن نسبت تعداد پردازنده فرگمنت به پردازنده راس بهینه دشوار است. مجموعه این دلایل منجر به آن شد که این دو پردازنده، متحد شوند.

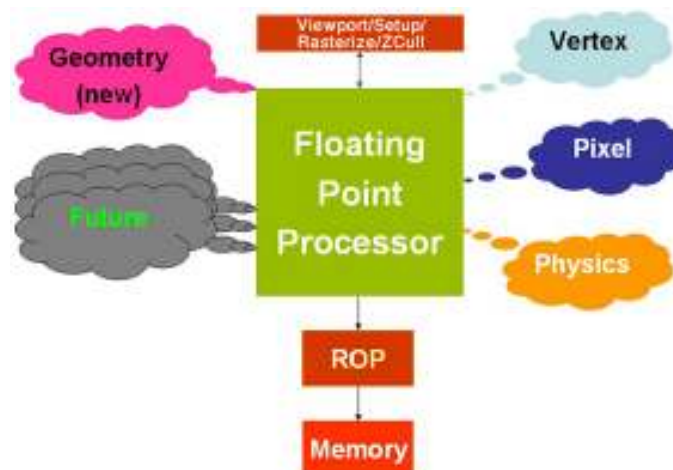
برای پیاده‌سازی خط‌لوله گرافیکی، هر مرحله یک بافر ورودی و یک بافر خروجی دارد. پردازنده متحد هر مرحله را پردازش می‌کند و خروجی هر مرحله خودکار به ورودی مرحله بعد انتقال پیدا می‌کند. تصویر ۲۸ این طراحی را به‌خوبی نمایش می‌دهد.

¹ Compute Unified Device Architecture (CUDA)



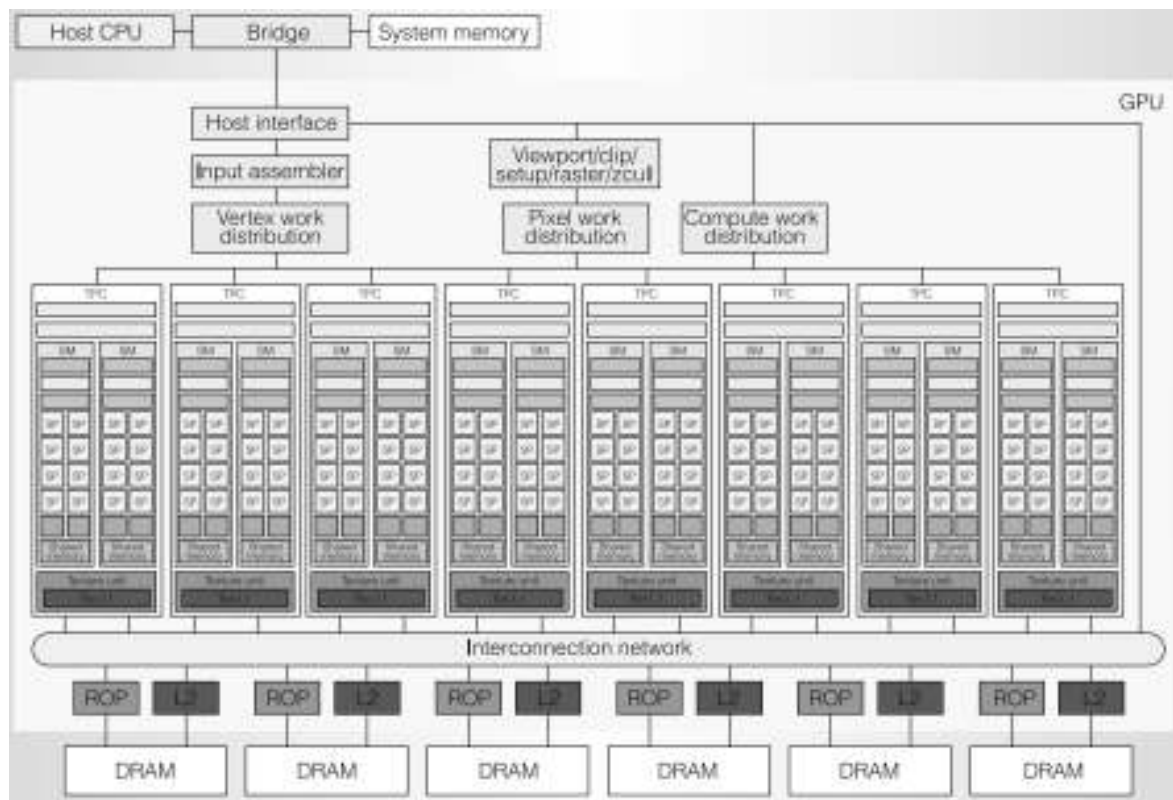
تصویر ۲۸- طراحی خطلوله با سایه‌زن یکپارچه (تصویر از منبع [24])

بدین ترتیب GPU دارای هسته‌ی برنامه‌پذیری خواهد بود که کلیه عملیات برنامه‌پذیر را انجام می‌دهد و برخی عملیات (مانند رسترایز کردن) که ثابت‌کاره‌اند در هنگام انتقال از یک مرحله به مرحله دیگر انجام می‌شوند. این هسته سایه‌زنی راس، هندسی و پیکسل را انجام می‌دهد و قابلیت پیاده‌سازی ایده‌های قابل برنامه‌ریزی آینده را هم دارد.



تصویر ۲۹- (مبتنی بر تصویر از منبع [25])

هدف اول در طراحی G80 یکی کردن سخت‌افزار اجرا کننده سایه‌زن راس و سایه‌زن پیکسل و هدف دوم تطابق کامل با DirectX10 بود. مشکلاتی چون تعدیل بار بهینه (نحوه اشتراک گذاری و دسترسی به حافظه و کنترل جریان و ...)، مقیاس‌پذیری، کارایی، توان و بهینگی استفاده از دای پیش رو بود. معماری G80 مبتنی بر آرایه‌ای مقیاس‌پذیر از پردازنده‌ها است.



تصویر ۳۰- بلوک دیاگرام G80 (تصویر از منبع [23])

G80، ۱۲۸ پردازنده جریان^۱ دارد که با ۱۶ چندپردازنده جریان^۲ سازماندهی می‌شوند. هر دو SM یک واحد مستقل به نام خوشه بافت/پردازنده^۳ را می‌سازند. مشابه بلوک دیاگرام NV40، جریان داده در تصویر ۳۰ هم از بالا به پایین جاری می‌شود. به علت معماری سایه‌زن یکپارچه، معماری G80 شباهتی به مراحل خطلوله گرافیکی ندارد. اگرچه برای معرفی معماری از خطلوله استفاده می‌کنیم. در بالاترین سطح، آرایه مقیاس‌پذیر پردازنده جریان^۴ تمام محاسبات قابل برنامه‌ریزی GPU را انجام می‌دهد. در پایین دیاگرام کنترل کننده‌های DRAM و پردازنده‌های عملیات رستر^۵ (که عملیات ترکیب رنگ و عمق بافر فریم را مستقیماً در ارتباط با حافظه انجام می‌دهند) را مشاهده می‌کنید و عضو سیستم حافظه هستند. شبکه‌ی اتصال^۶ عمق و رنگ‌های محاسبه شده هر پیکسل را از SPA به ROPها انتقال

^۱ Stream Processor (SP)

^۲ Stream Multiprocessor (SM)

^۳ Texture/Processor Cluster (TPC)

^۴ Stream Processor Array (SPA)

^۵ Raster Operation Processor (ROP)

^۶ Interconnect Network

می‌دهد. این شبکه وظیفه انتقال درخواست‌های خواندن بافت از SPA به DRAM و خواندن داده‌ها از DRAM (از مسیر حافظه نهان سطح دوم) به SPA را هم دارد. باقی بلوک‌ها در **Error! Reference source not found.** وظیفه رساندن کار به SPA را دارند.

اما خطلوله گرافیکی چگونه پردازش می‌شود؟ بخش اسمبلر ورودی^۱ کار رئوس را از دستور فراخوانی شده می‌گیرد. پخش کار رئوس^۲، بسته‌های کار رئوس را بین TPC‌های مختلف SPA پخش می‌کند. TPC‌ها برنامه سایه‌زنی رئوس یا سایه‌زنی هندسی را اجرا می‌کنند و خروجی‌ها در بافری ذخیره می‌شوند. سپس این بافرها نتیجه را به viewport/clip/setup/raster/zcull می‌فرستند تا این واحدهای ثابت‌کاره، پیکسل‌های پوشش‌دهنده‌ی اولیه‌ها را ایجاد کنند. این واحدها درست همان کاری را انجام می‌دهند که در NV40 دیدیم (البته با بهبود کیفی).^۳ واحد پخش کار پیکسل^۴، پیکسل‌ها را برای پردازش بین TPC‌های مناسب توزیع می‌کند. پیکسل‌های سایه‌زده شده از طریق شبکه اتصال، برای پردازش عمق و رنگ به واحدهای ROP فرستاده می‌شوند که مرحله ترکیب نهایی خطلوله گرافیکی را انجام می‌دهد. G80 قادر به پردازش غیرگرافیکی هم هست. بلوک پخش کار محاسبه^۵، آرایه‌ی نخ‌های محاسباتی (نخ‌های برنامه‌های غیرگرافیکی) را بین TPC‌ها پخش می‌کند. هر SPA کارهای رئوسی، پیکسلی و محاسباتی مختلفی را همزمان می‌پذیرد و انجام می‌دهد. بخش‌های مختلفی مثل واحدهای پخش، پردازنده‌ها، DRAM و باقی واحدها کلاک‌های متفاوتی دارند که امکان بهینه‌سازی توان و کارایی مستقل را به هر بخش می‌دهد.

واحد واسط میزبان^۶، (در واژه‌نامه G80، میزبان همان پردازنده مرکزی است) با پردازنده مرکزی ارتباط برقرار می‌کند، به فرمان‌های CPU پاسخ می‌دهد، داده‌ها را از حافظه اصلی واکشی می‌کند، وابستگی دستورات را بررسی می‌کند و تعویض محتوا^۷ انجام می‌دهد. اسمبلر ورودی علاوه بر واکشی خصوصیات رئوس، اولیه‌های هندسی (نقطه، خط و مثلث) را نیز جمع‌آوری می‌کند (باتوجه به اطلاعات

¹ Input Assembler

² Vertex Work Distribution

³ Pixel Work Distribution

⁴ Compute Work Distribution

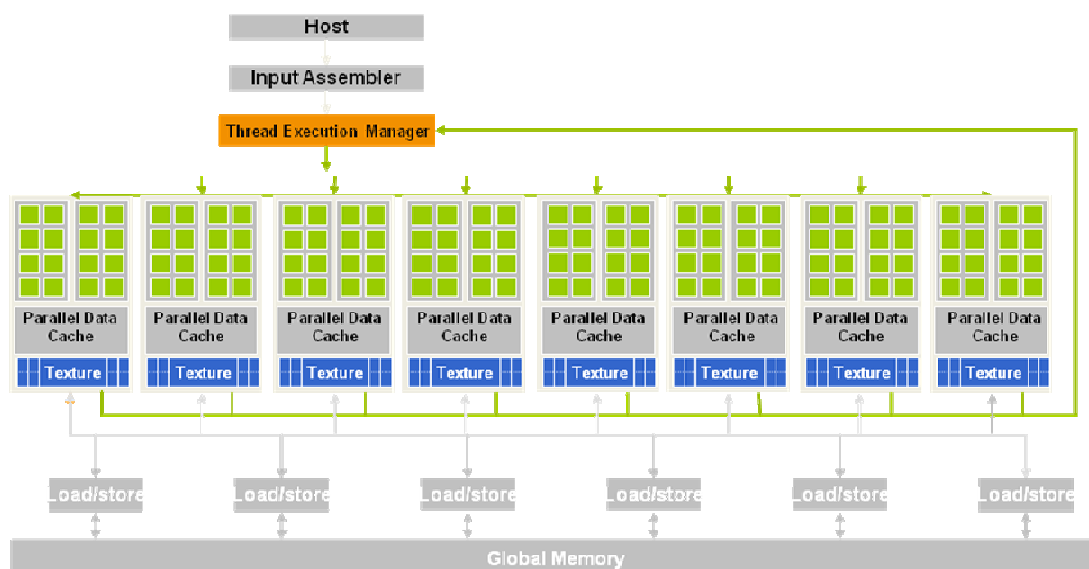
⁵ Host Interface

⁶ Context Switching

افزون، رئوس را در تشکیل خط یا مثلث شامل می‌کند). اسمبلر ورودی با کلاک هسته^۱ GPU کار می‌کند که حدود 600MHz است.

واحدهای پخش کار، جریان خروجی «اسمبلر ورودی» را به پردازنده‌ها می‌فرستند که برنامه راس، هندسی، پیکسل و محاسباتی را اجرا می‌کنند. واحدهای پخش رئوس و محاسبه، کار را با الگوریتم round-robin به پردازنده‌ها می‌رسانند که در ادامه بیشتر بررسی می‌کنیم. پخش کار پیکسل مبتنی بر موقعیت X,Y پیکسل است.

با ساده کردن بلوک دیاگرام G80 می‌توان آن را به صورت آرایه‌ای از پردازنده‌های چندهسته، یک نخ و تعدادی حافظه نهان دید.



تصویر ۳۱-بلوک دیاگرام ساده شده G80 (تصویر از منبع [26])

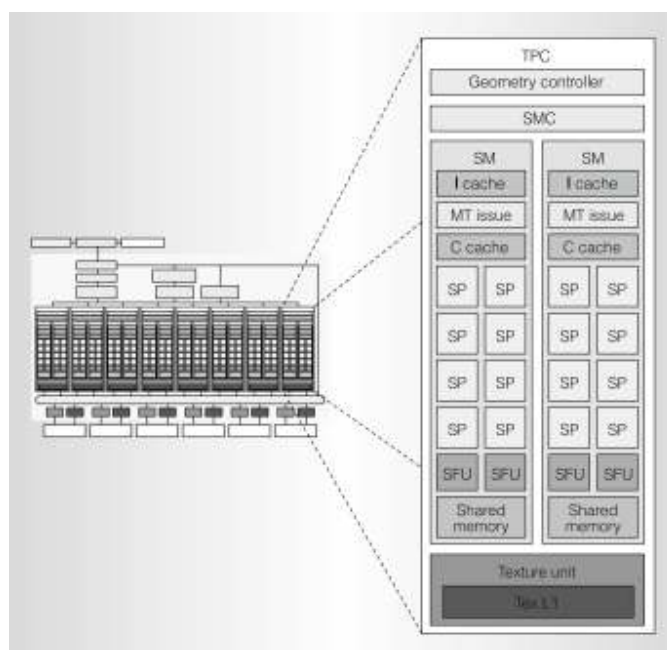
۳-۱ آرایه‌ی پردازنده جریان

SPA نخ سایه‌زنی گرافیکی و محاسباتی را اجرا می‌کند و مدیریت و کنترل نخ‌ها را هم بر عهده دارد. هر TPC در SPA مشابه یک واحد چهارپیکسلی در معماری NV40 هستند (هم واحد توابع خاص دارد و هم واحد بافت). تعداد TPCها کارایی پردازش‌های قابل برنامه‌ریزی GPU را تعیین می‌کنند. تعداد این TPCها در ضعیف‌ترین GPUها مشتق‌شده از G80 یکی است و در قوی‌ترین آنها تا بیش از ۸ عدد می‌شود.

¹ Core Clock

۳-۲ خوشه بافت/پردازنده

همانطور که در تصویر ۳۲ مشاهده می‌کنید، هر TPC یک کنترل‌کننده هندسی، یک کنترل‌کننده SM، دو چندپردازنده جریان (SM) و یک واحد بافت دارد.



تصویر ۳۲-واحدهای یک TPC

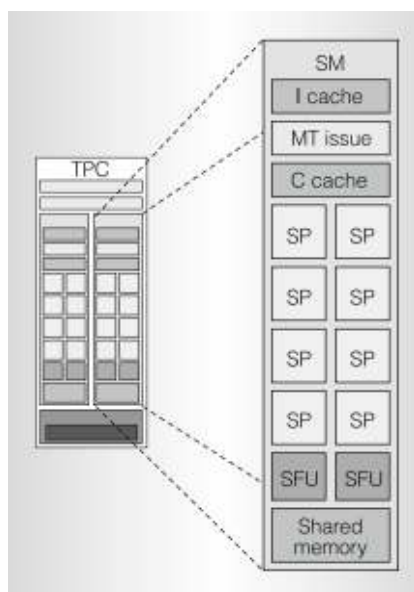
کنترل‌کننده هندسی وظیفه نگاشت سایه‌زنی راس به SM را دارد. وظیفه هدایت خصوصیت اولیه‌ها و رئوس در TPC را برعهده دارد. این کنترل‌کننده وظیفه مدیریت فضای ویژه‌ی ذخیره‌ی ورودی‌ها و خروجی‌های خصوصیات رئوس و در صورت نیاز ارسال محتوای آنها (جابه‌جایی بافر برای مراحل بعدی خطلوله) را دارد.

کنترل‌کننده SM (SMC) در خدمت تمام برنامه‌های راس، هندسی و پیکسل قرار دارد تا ورودی‌های این برنامه‌ها را به کلاف‌های تبدیل کند، پردازش‌های سایه‌زن را راه‌اندازی اولیه کند و در پایان نتیجه‌ها را جمع‌آوری کند. سه نوع برنامه مذکور مسیر ورودی و خروجی مجزایی دارند و کنترل‌کننده SM مسئول تعدیل بار (ایستا و پویا) بین آنها است که یکی از بحث برانگیزترین مقوله‌ها در طراحی G80 است. اگرچه وظیفه اصلی SMC کلاف‌بندی و سپردن به SM است ولی رابط واحد بافت و SM‌ها نیز هست.

واحد بافت برای دسترسی سریع به بافت‌ها به حافظه نهان سطح اول مجهز شده است. در ادامه درباره این واحد بیشتر بحث می‌کنیم.

۳-۳ چندپردازنده جریان

اجرای برنامه‌های گرافیکی مثل سایه‌زنی رئوس، هندسی، پیکسل و غیرگرافیکی محاسبه، توسط چندپردازنده جریان (SM) انجام می‌شود. همانطور که تصویر ۳۳ نشان می‌دهد هر SM هشت پردازنده جریان (SP)، دو واحد توابع خاص (SFU)، واحد واکنشی و صدور دستورهای چندمنحی (MT Issue)، حافظه نهان دستورالعمل (I cache)، حافظه نهان ثابت فقط خواندنی (C cache) و ۱۶ کیلوبایت حافظه مشترک^۱ خواندنی/نوشتنی دارد.



تصویر ۳۳-واحدهای یک SM

حافظه مشترک برای بافر کردن ورودی‌ها و خروجی‌های مراحل گرافیکی یا داده‌های مشترک بین نخ‌های برنامه غیرگرافیکی استفاده می‌شود (برای اینکه کارهایی که به SM محول شده، خط‌لوله‌ای انجام شوند، همانطوری که گفته شد نخ‌های رئوس، اولیه‌ها و پیکسل‌ها بافرهای ورودی و خروجی مستقلی

¹ Shared Memory

دارند که همین حافظه مشترک است). در SM، اجرای نخ همزمان با گرفتن کار جدید می‌تواند انجام شود.

هر SP واحد اسکالر ضرب-جمع (MAD) دارد که جمعا ۸ واحد MAD به SM می‌دهد. هر SM از واحد SFU برای محاسبه توابع سینوس، کسینوس، لگاریتم، نمایی و ... و درونیابی خصوصیات (مثلا درهنگام درونیابی خصوصیات پیکسل‌ها از رئوس اولیه نظیر) استفاده می‌کند. هر SFU چهار ضرب‌کننده ممیزشناور هم دارد [23].

SM از SMC و واحدهای ROP برای پیاده‌سازی دستورات ذخیره، بازیابی و اتومیک استفاده می‌کند. شبکه اتصال کم‌تاخیری بین SPها و حافظه مشترک وجود دارد که دسترسی به این حافظه را مهیا می‌کند.

۳-۳-۱ سیستم چندنخی^۱ در SM

قابلیت اجرای همزمان هزاران نخ ویژگی کلیدی G80 است. یک سایه‌زن راس یا پیکسل برنامه‌ای است تک‌نخ، که نحوه پردازش رئوس یا پیکسل‌ها را تعیین می‌کند. یک کرنل^۲ نوشته شده به زبان کودا نیز برنامه‌ای تک نخ است که مشخص می‌کند چگونه یک نخ، حاصل را بدست می‌آورد. هر برنامه‌ای برای پردازش آرایه‌ی ورودی چندین نخ تولید می‌کنند. برای تعدیل بار، نخ‌ها گروه‌بندی می‌شوند و چندین گروه همزمان نگهداری می‌شوند. زمان‌بندی نخ‌ها نیز گروهی انجام می‌شود و داده‌های هر گروهی که آماده باشد، می‌تواند اجرا شود. اگر داده‌های نخ‌ها در حافظه نهان نباشد موقتاً کنار گذاشته می‌شوند و گروه دیگری جای آنها را می‌گیرد (تعویض محتوای SM) تا SPها همواره درحال پردازش باشند. برای اینکه این سیستم چندنخی کارایی بالایی داشته باشد، مدیریت نخ‌ها کاملاً توسط سخت‌افزار انجام می‌شود و عملاً زمان‌بندی نخ‌ها بدون سربار انجام می‌شود. در ادامه سیستم مدیریت نخ‌ها را بررسی می‌کنیم.

¹ Multithreading System

² Kernel

۳-۳-۲ مدیریت و زمان‌بندی نخ‌ها

برای پشتیبانی از هردو برنامه‌نویسی گرافیکی و غیرگرافیکی، اطلاعات منحصربه‌فردی از هر نخ ذخیره می‌شود؛ مانند وضعیت اجرا. این اطلاعات نخ‌های یک گروه را قادر می‌سازد که انشعاب‌های متفاوتی از کد را دنبال کنند. از آنجا که انشعاب در گروه نخ‌ها باعث پس و پیش شدن آنها می‌شود، SM دارای دستوراتی برای همگام کردن آنها است.

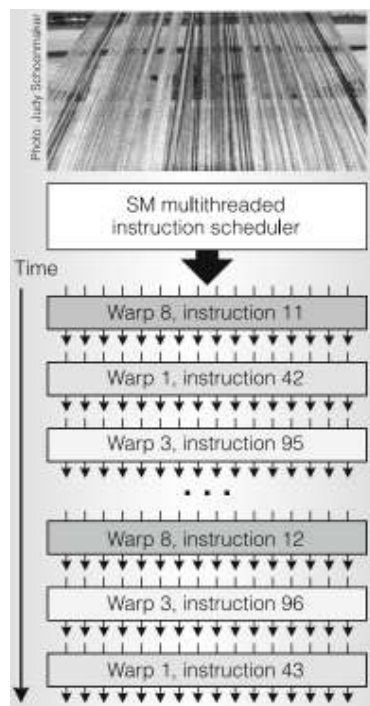
معماری SM نه MIMD است و نه کاملاً SIMD. نامی که شرکت سازنده برای این معماری گذاشته ^۱SIMT است. معماری SIMT شبیه طراحی SIMD است که یک دستور را روی چندین خط داده اجرا می‌کند. تفاوت اینجاست که SIMT یک دستور را روی چند نخ موازی اجرا می‌کند؛ نه فقط روی چندین خط داده. دستورات SIMD کنترل برداری داده‌ها را برعهده دارند که البته پهنای SIMD برای نرم‌افزار آشکار است، درحالی که دستورات SIMT کنترل اجرا و انشعاب یک نخ را برعهده دارند. SIMT به برنامه‌نویس امکان می‌دهد تا نخ‌هایی بسازد که موازی اجرا می‌شوند و البته این نخ‌ها با داشتن شناسه منحصر به فردی می‌توانند از موازی سازی داده هم استفاده کنند. برای نوشتن کد SIMT، می‌توان کاملاً فراموش کرد که چندنخ همزمان اجرا می‌کند (اگر بهینه‌سازی مورد توجه نباشد)، ولی کدهای SIMD همواره پهنای سخت‌افزار را به‌یاد دارند.

واحدی به نام دستور چندنخ‌شده^۲ کار ساخت، مدیریت و زمان‌بندی نخ‌ها را انجام می‌دهد. این واحد نخ‌ها را در گروه‌های ۳۲ تایی دسته بندی می‌کند که کلاف^۳ نامیده می‌شوند. در تصویر ۳۴ این زمان‌بندی و اجرا را مشاهده می‌کنید.

^۱ Single Instruction Multiple Thread

^۲ Multithreaded instruction unit

^۳ Warp



تصویر ۳۴-زمان بندی SIMT

هر SM انبوهی از ۲۴ کلاف را مدیریت می‌کند که نهایتاً ۷۶۸ نخ می‌شوند. تمام نخ‌های درون یک کلاف از یک نوع‌اند (راس، پیکسل یا محاسباتی)، یک برنامه دارند، از یک آدرس اجرای برنامه را شروع می‌کنند ولی آزادند که از هم گسسته شوند و انشعاب‌های متفاوتی از آن برنامه را اجرا کنند. در زمان صدور دستور، واحد دستور چندنخ‌شده کلافی را انتخاب می‌کند که آماده اجرا است و سپس دستور بعدی را به نخ‌های فعال کلاف صادر می‌کند. نخ‌هایی که انشعاب متفاوتی نسبت به باقی کلاف بگیرند، موقتاً غیرفعال می‌شوند.

نخ‌های کلاف به SPها سپرده می‌شود. پس از آن هر نخ در فضای آدرس و داده‌هایش مستقل عمل می‌کند. به عنوان نکته‌ای از کارایی، پردازنده SIMT وقتی با بیشترین کارایی کار می‌کند که تمام ۳۲ نخ کلاف یک مسیر اجرا داشته باشند و منشعب نشوند. اگر نخ‌های یک کلاف بوسیله دستورات انشعاب از هم جدا شوند، هر انشعاب را سریال اجرا می‌کند و نخ‌هایی را که متعلق به انشعاب جاری نیستند، غیرفعال می‌کند و این کار را ادامه می‌دهد تا انشعاب‌ها به پایان برسند و نخ‌ها دوباره هم مسیر شوند. این

کار با روش هم‌گرایی پشته‌ای^۱ انجام می‌شود که در پایان این قسمت آن را معرفی می‌کنیم. توجه کنید که انشعاب تنها درون کلاف اهمیت دارد و باید مدیریت شود و بیرون آن، کلاف‌ها مستقل هستند.

در معرفی معماری NV40 گفتیم که هر ۴ پردازنده فرگمنت یک واحد چهارپیکسلی می‌سازد که چهار پیکسل را پردازش می‌کنند و با هم همگام هستند. در G80، هر SM کار ۸ واحد چهارپیکسلی را انجام می‌دهد. در انجام محاسبات سایه‌زنی پیکسل، SMC هر ۴ پیکسل را گروه‌بندی و هر ۸ تایی از آنها (که ۳۲ پیکسل می‌شوند) در یک کلاف جای می‌گیرند. گروه‌بندی و تشکیل کلاف مشابهی برای رئوس و اولیه‌ها انجام می‌شود. طراحی SIMT، واحدهای واکنشی و صدور دستور را بین ۳۲ نخ به اشتراک می‌گذارد و همانطور که گفته شد وقتی این عملیات بهینه است که کلاف ۳۲ نخ فعال داشته باشد.

زمان‌بند کلاف‌ها کلاکی نصف کلاک پردازنده‌ها (SP) دارد و در هر سیکل یکی از ۲۴ کلاف موجود را برای اجرای دستورالعمل‌های انتخاب می‌کند. این زمان‌بند تمام کلاف‌های آماده را اولویت‌بندی می‌کند و سپس بالاترین اولویت را انتخاب می‌کند. این اولویت براساس نوع کلاف (راسی، هندسی، پیکسلی یا محاسباتی)، نوع دستورالعمل و سالخورده‌گی^۲ تعیین می‌شود.

دستورالعمل واکنشی شده هر کلاف، با دومجموعه ۱۶ نخ در ۴ سیکل پردازنده اجرا می‌شود. هسته‌های SP و واحدهای SFU مستقل از هم کار می‌کنند و SM با واکنشی دستورالعمل‌ها به صورت تناوبی بین آنها، هردو را مشغول می‌کند.

در ادامه این قسمت روش هم‌گرایی پشته‌ای که پیشتر به آن اشاره شد را معرفی می‌کنیم. این روش برای کنترل نخ‌هایی از یک کلاف است که انشعاب‌های متفاوتی دارند. انشعاب در زبان‌های سطح بالایی مثل کودا با if، switch، while یا for ایجاد می‌شود که در زیر نمونه‌ای کد که برای یک کلاف انشعاب می‌سازد آورده شده (توجه شود که i برای هر نخ منحصر به فرد است):

```

1:  if (a[i]==0) // نقطه واگرایی کلاف
2:  {
3:      انشعاب اول //
4:  }
5:  else
6:  {
```

^۱ Stack-Based Reconvergence

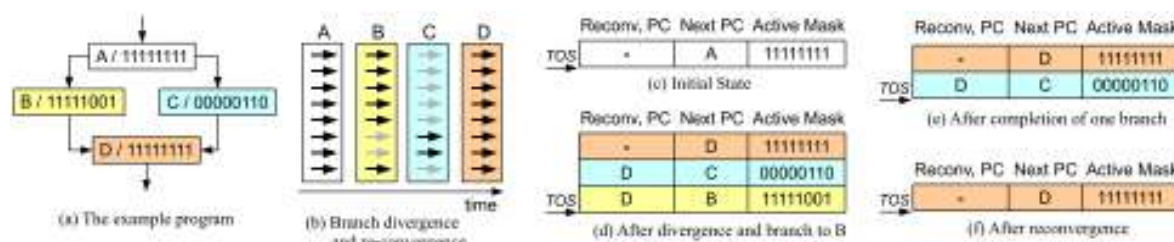
^۲ Aging

7: انشعاب دوم //

8: }

9: نقطه همگرایی //

درهنگام منشعب شدن نخ‌ها، کلاف به دو زیرکلاف تجزیه می‌شود که مشخصه‌ی «نخ‌های فعال» در هرکدام معرف نخ‌های عضو آن زیرکلاف است. برای مدیریت آنکه نخ‌های یک کلاف بتوانند انشعاب‌های متفاوتی را داشته باشند، باید اطلاعات اضافی از آنها ذخیره کرد. این اطلاعات که در ساختاری پشته‌ای ذخیره می‌شوند (تنها دسترسی پوش و پاپ ممکن است) منحصر به یک کلاف هستند و درهنگام انشعاب و جدایی نخ‌های کلاف، باید به‌روز شود. اطلاعات ذخیره شده برای هر زیرکلاف، ماسک فعال کلاف (نخ‌هایی از کلاف که فعال‌اند)، آدرس پسین (آدرسی که نخ‌های فعال کلاف باید دستور بعدی را از آن واکنشی کنند) و آدرس همگرایی (آدرسی که کلاف فعلی باید دوباره به مسیر اصلی بازگردد) هستند. این ساختار پشته‌ای را در تصویر ۳۵- (c تا f) مشاهده می‌کنید. بالای این پشته با ¹TOS مشخص شده است.



تصویر ۳۵- مدیریت انشعاب با همگرایی پشته‌ای (تصویر از منبع [27])

تصویر بالا همگرایی پشته‌ای برای سیستمی را نشان می‌دهد که اندازه کلاف‌ها ۸ نخ است. در تصویر ۳۵- a دنباله دستورات را مشاهده می‌کنید که در دستور A به انشعاب می‌رسند و تا این دستور تمامی نخ‌های کلاف فعال بوده‌اند که با رشته‌ای از 1ها نشان داده شده‌اند. پیش از اجرای دستور A، بالای پشته تنها یک کلاف وجود دارد که مشخصات آن به ترتیب از راست به چپ: ماسک نخ‌های فعال، آدرسی پسین و آدرس همگرایی هستند. چون کلاف مذکور زیرکلاف نیست آدرس همگرایی خالی گذاشته شده است و تمامی نخ‌ها فعال‌اند.

¹ Top Of Stack

پس از انشعاب دو زیرکلاف ایجاد می‌شوند و در هنگام رخ دادن چنین اتفاقی سه سطر به بالای پشته باید پوش شود: اول سطر مربوط به نقطه همگرایی (D) یا آدرسی که اجرای دو انشعاب (دو زیرکلاف) نهایتاً به هم می‌رسند، پوش می‌شود. دوم و سوم سطرها کلاف‌هایی‌اند که هرکدام آدرس متفاوتی را باید اجرا کنند. این که سطر مربوط به نقطه همگرایی قبل از باقی سطرها پوش می‌شود، تضمین می‌کند قبل از اینکه D اجرا شود، هریک از انشعاب‌ها اجرای خود را تمام کرده‌اند. پس از به‌روز کردن پشته، TOS ماسک کلاف و آدرس بعدی که باید اجرا شود را مشخص می‌کند. در این مثال آدرس بعدی B است و ماسک کلاف 1111001 است. پس از پاپ کردن B از پشته، این زیر کلاف اجرا می‌شود و وقتی به آدرس D رسید متوقف می‌شود و زیرکلاف دیگری از بالای پشته برداشته می‌شود (C پاپ می‌شود) و با ماسک کلاف 00000110 اجرا می‌شود. توجه کنید که از مشخصه سوم هر سطر در پشته (یعنی آدرس همگرایی) برای توقف اجرای زیرکلاف استفاده می‌شود و کلافی که این مشخصه را ندارد زیرکلاف نیست و مسیر اصلی دنباله دستورالعمل‌ها است و متوقف نمی‌شود.

پس از اینکه انشعاب شاخه C به دستور D (آدرس همگرایی) رسید، متوقف می‌شود و کلاف دیگر از بالای پشته پاپ می‌شود که در این مثال این کلاف آدرس همگرایی یعنی D را دارد و دنباله اصلی دستورالعمل‌ها است. بدین ترتیب مدیریت انشعاب به روش همگرایی پشته‌ای انجام می‌شود و در تصویر بالا (b) خط‌زمان این اجرا را هم ملاحظه می‌کنید. ساختار پشته اجازه انجام انشعاب‌های تودرتو^۱ را هم می‌دهد.

بهینه‌سازی‌های زیادی برای افزایش کارایی در هنگام رخ دادن چنین انشعاب‌هایی صورت می‌گیرد. یکی از این بهینه‌سازی‌ها زمانی است که انشعاب رخ می‌دهد اما اگر آدرس دو انشعاب به هم نزدیک باشد (به نحوی که هر دو شاخه در حافظه نهان دستورالعمل قرار بگیرند)، اجرای دوزیرکلاف سریال نمی‌شود و SM هر دو را همزمان اجرا می‌کند [28].

۳-۳-۳ تحمل تاخیر^۲

به‌طور کلی در دو حالت کلاف به دو زیرکلاف تقسیم می‌شود: زمانی که دنباله دستورات به انشعابی برسند یا زمانی که دسترسی به داده‌های یک یا چند نخ کلاف با فقدان^۱ روبرو شود. از آنجایی که

^۱ Nested Branch

^۲ Latency Tolerance

دسترسی به حافظه اصلی برای مدیریت فقدان صدها سیکل زمان می‌برد، SM باید بتواند این سیکل‌های بیکار ماندن را جبران کند. تکنیک‌های متفاوتی برای جبران هست که یکی از آنها چندکلافی کار کردن است. بدین صورت که SM همواره چندین کلاف دارد و از بین آنهایی که آماده‌اند انتخاب می‌کند که کدام را اجرا کند. در هنگام فقدان عملوندهای کلاف، با تعویض متن و اجرای کلاف آماده‌ی دیگری تاخیر واکشی عملوندها جبران می‌شود. اما آیا هرچه ظرفیت پذیرش کلاف بیشتر باشد، به صرفه‌تر است؟ جواب منفی است. از آنجایی که اطلاعات مدیریت کلاف‌ها در حافظه‌های سریع (و گران قیمت) ذخیره می‌شود، افزایش ظرفیت پذیرش کلاف، افزایش این حافظه سریع است و این به معنی افزایش شدید هزینه ساخت است. این هزینه ساخت در قیاس با کارایی بدست آمده مطلوب نیست.

۳-۳-۴ دستورالعمل‌های SM

برخلاف GPUهای پیشین که دستوراتی برداری داشتند، SM دستورالعمل‌های اسکالر دارد. همانطوری که در معماری NV40 دیدیم، انجام عملیات‌های همزمان روی رنگ و شفافیت (جمعا ۴ داده) ۱۶ پیکسل همزمان ممکن بود. ولی در G80، SPها پردازنده‌ای SIMD نیستند و اسکالرنند. اگر در NV40 برای انجام عملی یکسان روی ۴ مشخصه یک پیکسل، یک کلاک زمان لازم دارد، در G80 این کار باید در ۴ مرحله انجام شود و در هر مرحله یک خصوصیت پردازش گردد. با این تفاسیر چرا عملیات برداری توسط G80 پوشش داده نمی‌شود؟

محققان انویدیا از سال ۲۰۰۲ تا ۲۰۰۶ برای ساخت G80 کار کردند و برداری یا اسکالر بودن SPها را تحلیل کردند. آنها آمار و ارقامی از برنامه‌های سایه‌زنی متفاوت در سال‌های متوالی تهیه کردند. این آمار نشان می‌داد که برنامه‌های سایه‌زنی از برداری بودن به سمت اسکالر شدن در حالا انتقال‌اند. نتایج نشان می‌داد که عملیات‌های برداری روی تمام خصوصیات بشدت در حال نزول است و حتی عملیات‌های برداری با پهنای ۲ هم به‌ندرت رخ می‌دهد. از طرف دیگر در راستای تلاش برای همه‌منظوره کردن GPU، برداری بودن SPها اتلاف سخت‌افزاری بود و تنها معماری GPU را پیچیده‌تر می‌کرد. این مجموعه دلایل باعث شد که SPها معماری اسکالر داشته باشند. ولی معماری واحد بافت

همچنان برداری هستند؛ مختصات بردار مبدا را می‌گیرند و بردار رنگ فیلترشده را برمی‌گردانند. (در ادامه به معرفی واحد بافت خواهیم پرداخت)

برای سازگاری با دستورهای برداری زبان DirectX10 و نسخه‌های ما قبل، بهینه‌ساز واسط می‌شود و دستورات برداری را به چند دستور اسکالر SM تبدیل می‌کند. برای برنامه‌نویسی غیرگرافیکی G80، زبان اسمبلی به نام PTX^۱ معرفی شد. هر دستور از مجموعه دستورالعمل‌های اسکالر PTX تقریباً معادل یک دستورالعمل اسکالر SM است. تمامی کامپایلرهای زبان‌های برنامه‌نویسی G80، زبانی میانی تولید می‌کنند که توسط دایور گرافیک به باینری قابل اجرا تبدیل می‌شود. تمامی زبان‌های میانی^۲ (از جمله PTX) با ثبات‌های مجازی سروکار دارند و بهینه‌ساز ثبات‌های واقعی را تخصیص می‌دهد.

مجموعه دستورالعمل‌های ثباتی SM شامل عملیات‌های ممیزشناور، صحیح، بیتی، تبدیلی، پیشرفته^۳، کنترل جریان، ذخیره/بازیابی از حافظه و بافت هستند.

دستورات ممیزشناور و صحیح شامل جمع، ضرب، ضرب-جمع، مینیمم، ماکزیمم، مقایسه، تبدیل بین صحیح و ممیزشناور هستند.

دستورات بیتی شامل شیفت به چپ، شیفت به راست، عملگرهای منطقی و انتقال هستند. دستورات پیشرفته شامل کسینوس، سینوس، نمایی با پایه ۲، لگاریتم در مبنای ۲، معکوس و معکوس مجذور هستند.

دستورات کنترل جریان شامل انشعاب (پرش)، فراخوانی^۴ (فراخوانی تابع و ذخیره آدرس بازگشت)، بازگشت^۵ (بازگشت به دستور قبل از فراخوانی)، ترپ^۶ (ایجاد وقفه به پردازنده میزبان) و همگامی^۷ (هماهنگ کردن نخ‌های و کنترل اجرای آنها) هستند. دستورات ممیزشناور و صحیح برای هر نخ، پرچم‌های سرریز، نقلی، منفی و صفر را مقدار می‌دهند که می‌توان با دستورات انشعاب از آنها استفاده کرد.

¹ Parallel Thread Execution

² Intermediate Languages

³ Transcendental

⁴ Call

⁵ Return

⁶ Trap

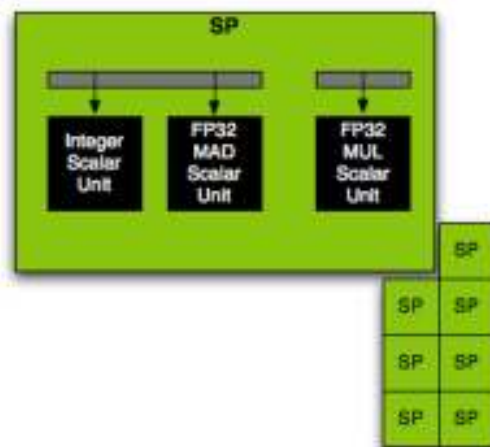
⁷ Synchronization

دستورات بافت، نمونه‌های بافت را از حافظه از طریق واحد بافت، واکشی و فیلتر می‌کنند. واحد ROP نیز پیکسل‌های خروجی را در حافظه می‌نویسند. ولی این خواندن و نوشتن حافظه برای GPU همه‌منظوره‌ای مثل G80 که با C/C++ قابل برنامه‌ریزی است، کافی نیستند. بدین جهت دستورات ذخیره/بازیابی حافظه به ISA اضافه شده‌اند. دستورات ذخیره/بازیابی به سه فضای حافظه نوشتنی/خواندنی دسترسی دارند: محلی، مشترک و سراسری. هر نخ حافظه‌ای خصوصی به نام حافظه محلی دارد که در DRAM خارجی تخصیص می‌شود. حافظه مشترک که از جنس نهان است، داخل SM قرار دارد و برای دسترسی سریع به داده‌های مشترک بین نخ‌ها طراحی شده است. حافظه سراسری در DRAM است و برای داده‌هایی است که بین تمام نخ‌ها مشترک‌اند. برای دسترسی به این فضاها ۶ دستور العمل مجزا وجود دارد: بازیابی-سراسری، ذخیره-سراسری، بازیابی-مشترک، ذخیره-مشترک، بازیابی-محلی و ذخیره-محلی. چون این سه فضا از لحاظ آدرس‌دهی از هم مجزا هستند، درهنگام کدنویسی و مشخصا کار کردن با اشاره‌گرها در C مشکلاتی در دسترسی به این فضاها پدید می‌آید البته با استفاده از این دستورالعمل‌ها مشکلی وجود ندارد.

۳-۳-۵ پردازنده جریان

هسته SP پردازنده نخ در SM است. این واحد علاوه بر عملیات‌های صحیح، مقایسه و تبدیلی، عملیات ممیزشناور جمع، ضرب و ضرب-جمع را هم انجام می‌دهد. این پردازنده کاملاً خط‌لوله‌ای کار می‌کند. عملیات ضرب و جمع SP مطابق با IEEE 754 هستند و به‌طور پیش‌فرض از روش گردکردن round-to-nearest-even استفاده می‌کنند. عملوندهای غیرنرمال با حفظ علامت به صفر تبدیل می‌شوند و همین اتفاق برای محاسباتی که فروریز^۱ دارند، رخ می‌دهد.

^۱ Underflow



تصویر ۳۶- معماری ALUهای یک SP (تصویر از منبع [29])

همانطوری که در تصویر ۳۶ مشاهده می‌کنید، SP قادر است یک ضرب موازی کلیه عملیات بالا انجام دهد. عموماً کلاک SPها 1350MHz یا 1.35GHz است، پس برای محاسبه GFLOPs هر SM داریم:

$$SPPeak = 2 * SPMAD(1) + SPMUL(1) = 3$$

$$PeakGFLOPs = 8 * SPPeak * ClockFreq$$

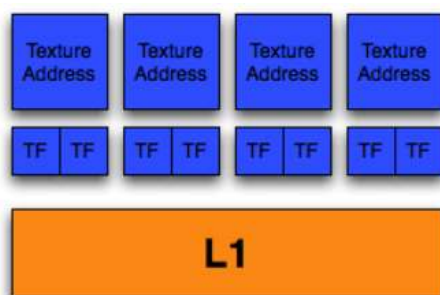
توجه کنید که SPPeak به این علت در ۸ ضرب شده که هر SM، ۸ SP دارد و SPMAD در ۲ ضرب شده زیرا عمل ضرب-جمع دو عمل ممیز شناور به حساب می‌آید. با محاسبه این پیک کارایی 32.4GFLOPs بدست می‌آید.

۳-۳-۶ واحد توابع خاص

علاوه بر توابع پیشرفته، این واحد دورنمایی خصوصیات را هم انجام می‌دهد. در NV40 دیدیم که هرکدام از پردازنده‌های راس و فرگمنت واحد توابع خاص داشتند. علاوه بر این پردازنده فرگمنت واحد درونیایی خصوصیات پیکسل‌ها از رئوس اولیه هم داشت. در G80 تمام این عملیات با این دو SFU انجام می‌شود. دستورات پیشرفته که شامل کسینوس، سینوس، نمایی با پایه ۲، لگاریتم در مبنای ۲، معکوس و معکوس مجذور هستند، توسط SFU انجام می‌شوند.

۳-۴ واحد بافت

واحد بافت در هرسیکل یک گروه چهار نخه را پردازش می‌کند. مبدا دستورالعمل‌های بافت، مختصات بافت هستند و خروجی نمونه‌های فیلترشده‌اند که معمولاً چهار بخش RGBA دارد. واحد بافت خارج از SM است و از طریق SMC با آن ارتباط برقرار می‌کند. واحد بافت شامل ۴ تولیدکننده آدرس بافت^۱، ۸ فیلترکننده و سطح اول نهان است.



تصویر ۳۷- واحد بافت (تصویر از منبع [29])

۳-۵ رسترایزر

اولیه‌هایی که در SM پردازش شده‌اند، برای طی کردن باقی مراحل خطلوله گرافیکی به viewport/clip/setup/raster/zcull فرستاده می‌شوند. واحد viewport و clip مشابه NV40، وظیفه تبدیل مختصات و برش اولیه‌ها از زاویه دید جاری را دارند. واحد setup نیز معادلات لبه را برای رسترایزر تولید می‌کند. معادلات خصوصیات سطح اولیه که برای درونیابی خصوصیات پیکسل‌های پوشش‌دهنده‌ی سطح اولیه لازم‌اند، تولید می‌شوند.

واحد raster تمام پیکسل‌هایی که در سطح اولیه هستند را در دو مرحله تولید می‌کند. در مرحله اول (coarse-rasterization) پیکسل‌هایی که حتی قسمتی از آنها در اولیه است، تولید می‌شوند و واحد zcull بررسی می‌کند که پیکسل تولید شده توسط مرحله اول رسترایزر آیا بوسیله پیکسل‌های دیگری غیرقابل رویت شده‌اند یا نه. با نرخ بیشینه ۲۵۶ پیکسل در هر کلاک، پیکسل‌های غیرقابل رویت را حذف می‌کند. پیکسل‌های حذف نشده وارد مرحله دوم رسترایزر (fine-rasterization) می‌شوند که اطلاعات کامل‌تر پوشش و عمق تولید می‌شود.

¹ Texture Address

صفحه نمایش به بخش‌های تقسیم می‌شود و هر TPC مسئول پردازش بخشی می‌شود. هر پیکسل بنابر مختصات که دارد به TPC نظیر فرستاده می‌شود. واحد SMC پیکسل‌های رسیده به TPC را در کلاف‌ها بسته‌بندی می‌کند تا SM سایه‌زنی پیکسل روی آنها انجام دهد. با پایان سایه‌زنی پیکسل، SMC پیکسل‌های سایه‌زده شده و داده‌های مرتبط به آنها (مانند بافت‌ها) را به ROP می‌فرستد. اگرچه zcull به صورت خودکار پیکسل‌های غیرقابل رویت را پیش از سایه‌زنی پیکسل حذف می‌کند، اما امکان این هست که از طریق API‌های گرافیکی فرمان سنجش عمق صادر شود و پیکسل‌های غیرقابل رویت حذف شوند. در API‌هایی مثل OpenGL و Direct3D سنجش عمق پس از سایه‌زنی پیکسل انجام می‌شود.

۳-۶ پردازنده عملیات رستر^۲

هر ROP با بخش مشخصی از حافظه جفت شده است. هر بخش حافظه ۶۴ بیت پهنا دارد و با پشتیبانی از پروتکل‌های DDR2 و GDDR3 و کلاک 1GHz کار می‌کند. با این مشخصات هر بخش حافظه پهنای باند $2 \times 8 \text{Byte} \times 1 \text{GHz} = 16 \text{Gbyte/s}$ خواهد داشت (پهنای بایتی هر بخش است و ضریب ۲ به علت پروتکل DDR است که در هر دو لبه (بالارونده و پایین‌رونده) داده انتقال داده می‌شود [31]). برای کاهش انتقال بین حافظه و ROP، کنترل‌کننده‌های حافظه فشرده‌سازی عمق و رنگ را انجام می‌دهند.

ROP سنجش‌های عمق و استنسیل را انجام می‌دهد. این واحد بسیار شبیه واحد ترکیب پیکسل در NV40 است. عملیات ضدناصافی با روش‌های چندنمونه‌برداری و ابرنمونه‌برداری تا مقیاس $16 \times$ پشتیبانی می‌شود. بالاتر از $4 \times$ هزینه ذخیره نمونه‌ها با نرخ زیاد می‌شود که کیفیت تصویر زیاد نمی‌شود. چون چندنمونه‌برداری برای هر نمونه، سایه‌زن پیکسل را تنها یکبار اجرا می‌کند، سرعت بیشتری دارد و محبوب‌تر از ابرنمونه‌برداری است. در G80 روش ضدناصافی جدیدی معرفی شد که ضدناصافی نمونه‌های پوششی^۳ نام دارد. این روش بهینه‌سازی برای چندنمونه‌برداری است که با افزودن چندنمونه که تنها خصوصیت پوشش دارند، کیفیت تصویر را بالا می‌برد [32].

¹ Tile

² Raster Operation Unit (ROP)

³ Coverage Sampled Antialiasing (CSAA)

فصل چهارم: مقایسه معماری CPU و GPU

تفاوت بنیادی CPU و GPU از داده‌ها و برنامه‌هایی ناشی می‌شود که با آنها سروکار داشته‌اند. از اولین روزهای پیدایش CPU، وظیفه دنبال کردن یک مسیر دستورالعمل را داشت و هدف طراحی آن سرعت بخشیدن به اجرای دنباله دستورات بود. اگرچه پردازنده‌های امروزی چند هسته هستند و اهداف دیگری را هم دنبال می‌کنند، ولی در قلب پردازنده‌ها، هر هسته برای همان هدف اصلی CPU تلاش می‌کند. در نقطه مقابل، GPU برای پردازش موازی داده‌هایی مشابه ساخته شد و برون‌دهی بالا اصلی‌ترین هدف طراحی آن بود. معماری هریک از این دو پردازنده در جهت تحقق این هدف پیشرفت کردن و رفته‌رفته از هم فاصله گرفتند. در ادامه برخی از تفاوت‌های کلیدی امروزی این دو پردازنده را بررسی می‌کنیم.

۴-۱ حجم نخ

اگر تعداد دستورالعمل‌ها و مدت‌زمانی که لازم است صرف اجرای یک نخ شود را حجم نخ بنامیم، GPU معمولاً با نخ‌های سبک کاردارند ولی نخ‌های CPU معمولاً سنگین هستند و زمانی زیادی از پردازنده می‌گیرند. به علت اینکه حافظه نهان در دسترس هر نخ GPU بسیار کوچک است، اگر داده‌های موردنیاز نخ زیاد باشد، بیشتر از اینکه نخ‌ها مشغول اجرا باشند، زمان صرف جابه‌جایی بین DRAM و نهان می‌شود.

۴-۲ وابستگی نهان^۱

در CPU چند هسته، بسیار اهمیت دارد که سازگاری بین حافظه‌های نهان خصوصی هر هسته بررسی شود. هر هسته حافظه نهان منحصر به خود (L1) دارد و سطوح بعدی بین دو یا چند هسته دیگر مشترک هستند. وقتی هسته‌ای داده‌ای را تغییر می‌دهد و در L1 ثبت می‌کند، L1 هسته‌هایی که کپی از این داده داشته‌اند، داده‌ای نامعتبر دارند. کنترل این ناسازگاری و جلوگیری از آن اغلب در CPUها انجام می‌شود.

¹ Cache Coherency

اما در GPU این مسئله اهمیت ندارد. زیرا GPU روی داده‌هایی کار می‌کند که ذاتاً از هم مستقل هستند. اگر لازم باشد داده‌ای بین تمام هسته‌ها به اشتراک گذاشته شود، از حافظه ثابت یا فقط خواندنی استفاده می‌شود. معماری GPU کنترل روی سازگاری نهان ندارد.

۳-۴ مقیاس‌پذیری

مقیاس‌پذیری GPUها بسیار بیشتر از CPUها است و یکی از دلایل این موضوع سازگاری نهان است. برای افزایش تعداد هسته‌های CPU یکی از دغدغه‌های اصلی، امکان بررسی سازگاری نهان است.

۴-۴ کلاکینگ و فرکانس

کلاک هسته‌های CPUها معمولاً حدود 3GHz است و این درحالی است که کلاک هسته‌های GPUها نصف این مقدار است. واضح است که برای یک نخ، هرچه کلاک بالاتر باشد، سرعت اجرا بیشتر است. تفاوت دیگری از کلاکینگ آن است که واحدهای مختلف GPUها برای افزایش کارایی با کلاک‌های متفاوتی کار می‌کنند ولی کلاک CPU یکتا است (صرف‌نظر از حافظه رم که البته خارج از پردازنده است).

۵-۴ سطح موازی‌سازی

پردازنده‌های چند هسته درست مثل GPU می‌توانند چندین کار را تحویل بگیرند و همزمان انجام دهند. با توزیع‌های SIMD که برای پردازنده‌های امروزی وجود دارد، آنها قادر هستند که موازی‌سازی داده هم انجام دهند. اگرچه CPU و GPU هر دو موازی‌سازی کار و موازی‌سازی داده انجام می‌دهند، ولی درجه اینکار در آنها متفاوت است. هر پردازنده‌ای به تعداد هسته‌هایش قادر است نخ موازی اجرا کند. در محصولات متفاوت، تعداد هسته‌ها متفاوت است ولی نوعاً تعداد هسته‌های GPUها صدها برابر CPUها است. موازی‌سازی داده در CPUها محدود به پهنای SIMD (عموماً ۱۶) است ولی GPUها موازی‌سازی داده با محدودیت «تعداد نخ‌های فراخوانی» (در G80 ۷۶۸ نخ) داشته باشند.

۴-۶ زمان‌بندی

پردازنده‌ی چند هسته می‌تواند چندین کار را همزمان انجام دهد ولی زمان‌بندی این کارها توسط سیستم عامل انجام می‌شود. زمان‌بندی در GPUها توسط سخت‌افزار انجام می‌شود و پس از سپردن کار به GPU کنترل زمان‌بندی در دسترس نیست. اگرچه دستوراتی برای کنترل و همگام کردن نخ‌های یک فراخوانی وجود دارد ولی این دستورات کنترلی برای زمان‌بندی بدست نمی‌دهند.

۴-۷ مصرف ترانزیستور

طبق قانون مور در مساحت معینی از چیپ، تعداد ترانزیستورهای ۱۸ ماه بعد دو برابر می‌شود. CPUها اغلب این ترانزیستورهای اضافی را صرف حافظه نهان و واحد کنترل می‌کنند. اما GPU اغلب ترانزیستورهای چیپ را برای پردازش کار می‌برند. دلیل این دو گرایش متفاوت همان هدف طراحی است. هرچه واحد کنترل CPUها از تخلیه خط‌لوله بیشتر جلوگیری کند، سرعت اجرای یک شاخه نخ بیشتر می‌شود و در GPU هرچه ALU بیشتری داشته باشیم، برون‌دهی بالاتر است.

۴-۸ سلسله مراتب حافظه

بزرگترین حافظه در دسترس CPU و GPU از جنس DRAM است. از آنجایی که این حافظه کند است، برای کاهش تاخیر تدابیری متفاوتی اندیشیده شده است. تدبیر CPU، افزایش درصد برخورد حافظه نهان و تدبیر GPU، افزایش پردازش چندنخی است. ساده‌ترین راه افزایش درصد برخورد، افزایش حافظه نهان است. اما در GPUها تکیه اصلی روی افزایش ظرفیت پذیرش نخ‌های همزمان است. کندی واکنشی عملوندها از حافظه DRAM با پردازش نخ دیگری که عملوندهایش حاضرند جبران می‌شود.

- [1] Fundamentals Of Computer Graphics Second Edition; Peter Shirley et al.
- [2] Computer Graphics; Amarendra N. Sinha, Arun D. Udai; Tata McGraw-Hill 2008
- [3] <http://www.devhardware.com/c/a/Video-Cards/The-Graphics-Pipeline/1/>
- [4] Introduction to GPU; Marc Olano, David Luebke et al.
- [5] The Graphics Pipeline; Kurt Akeley, Pat Hanrahan; Fall 2001.
- [6] [http://msdn.microsoft.com/en-us/library/ff476882\(v=VS.85\).aspx](http://msdn.microsoft.com/en-us/library/ff476882(v=VS.85).aspx)
- [7] How GPUs Work; David Luebke, Greg Humphreys;
- [8] The OpenGL Graphics System: A Specification (Version 4.0 (Core Profile) - March 11, 2010); Mark Segal, Kurt Akeley;
- [9] GPU Computing; John D. Owens, Mike Houston, David Luebke, Simon Green, John E. Stone, and James C. Phillips;
- [10] GPU Architecture Overview; John Owens, UC Davis;
- [11] GPU Architecture: Implications & Trends; David Luebke; SIGGRAPH2008;
- [12] Course Documentation; GPU Programming and Architecture, CIS 665, University of Pennsylvania, Spring 2010;
- [13] Graphic pipeline and implementation; Zhenyu Ye;
- [14] State of the Art Report on GPU; Rita Borgo, Ken Brodlie; University of Leeds; <http://www.gpucomputing.org/>;
- [15] Xenos: XBOX360 GPU; Michael Doggett; September 1, 2005
- [16] The GeForce 6800; John Montrym, Henry Moreton; IEEE Computer Society, IEEE Micro p.41
- [17] Computer Organization & Design: the Hardware / Software Interface Second Edition; David A. Patterson, John L. Hennessy.
- [18] GPU Gems 2, Chapter 30: The GeForce 6 Series GPU Architecture;
- [19] NVIDIA GeForce 6800 Ultra (NV40). Part One; <http://ixbtlabs.com/articles2/gffx/nv40-part1-a.html>
- [20] P.S. Heckbert, "Survey of Texture Mapping," IEEE Computer Graphics and Applications, vol. 6, no. 6, Nov. 1986, pp. 56-67
- [21] Mathematics for 3D Game Programming and Computer Graphic Second Edition; Eric Lengyel
- [22] GPUs - Graphics Processing Units; Minh Tri Do Dinh; Vertiefungsseminar Architektur von Prozessoren, SS 2008; Institute of Computer Science, University of Innsbruck; July 7, 2008
- [23] NVIDIA Tesla: a Unified Graphics and Computing Architecture; Erik Lindholm, John Nickolls, Stuart Oberman, John Montrym;
- [24] The NVIDIA GeForce 8800 GPU; Erik Lindholm, Stuart Oberman; August 2007;
- [25] Technical Brief NVIDIA GeForce 8800; November 2006;
- [26] Threading Hardware in G80; David Kirk, Wen-mei W. Hwu; ECE 498AL, University of Illinois, Urbana-Champaign, 2009;
- [27] Dynamic Warp Subdivision for Integrated Branch and Memory Divergence Tolerance; Jiayuan Meng, David Tarjan, Kevin Skadron; proceedings of ACM ISCA 2010.
- [28] Deterministic Sample Sort For GPUs; Frank Dehne, Hamidreza Zaboli; February 24, 2010;
- [29] AnandTech.com - NVIDIA's GeForce 8800 (G80): GPUs Re-architected for Di=

rectX 10; <http://www.anandtech.com/print/2116>

[30] 3DCenter - Multisampling Anti-Aliasing: A Closeup View;
http://alt.3dcenter.org/artikel/multisampling_anti-aliasing/index2_e.php

[31] Computer architecture : a quantitative approach; John L. Hennessy, David A. Patterson ; with contributions by Andrea C. Arpaci-Dusseau; Amsterdam: Morgan Kaufmann. ISBN 0123704901;

[32] Coverage Sampled Antialiasing; NVIDIA Technical Report

3	Fragment, 7	O
3dfx, 21	G	OpenGL, 32, 59
A	G80, 41, 44, 54	P
Aging, 51	GDDR3, 59	Page-friendly, 33
AGP, 22, 27	GeForce 6800, 27	Parallel Thread Execution, 55
Aliasing, 8	GeForce FX 5800, 23	PCI-Express, 27
Anti-Aliasing, 8	Geometry Shader, 6	Pixel Shader, 8
API, 32	GFLOPs, 57	Primitive, 6
B	GPGPU, 25	Primitive Assembly, 6
Blending, 9	Graphical Pipeline, 2	Principle of Locality, 28
Bottleneck, 17	H	PTX, 55
C	Hide the latency, 35	Q
C cache, 47	I	Quad-Pixel Unit, 35
Cache Coherency, 61	IBM PC, 21	R
clip, 58	IEEE 754, 56	Radeon 9800, 23
Clip, 33	Interconnect Network, 43	Radeon7500, 22
coarse-rasterization, 58	ISA, 18, 56	RAMDAC, 21
Configurable, 20	K	raster, 58
Core Clock, 45	Kernel, 48	Raster Operation Processor
Cull, 33	L	(ROP), 43
Culling, 5	Latency Tolerance, 53	Raster Operation Unit, 10
D	Level of Detail, 37	Rasterizer, 7
data fetcher, 30	Load Balancing, 17	Read-Only, 25
Data Parallelsim, 16	Locality in Space, 28	RGBA, 5
DDR2, 59	Locality in Time, 28	ROP, 44, 48
depth buffering, 10	Lossless, 38	round-robin, 45
Direct3D, 59	M	round-to-nearest-even, 56
DirectX, 32	MAD, 48	S
DirectX10, 25, 42, 55	MIMD, 31, 49	Setup, 33
DirectX9, 23	Miss, 54	SFU, 47
Dot product, 37	MT Issue, 47	Shared Memory, 47
DRAM, 35, 43, 44, 56, 63	Multisampling, 15	SIMD, 31, 35, 49, 54, 62
E	Multithreaded instruction unit, 49	SIMT, 49, 50, 51
EGA, 21	Multithreading System, 48	Single Instruction Multiple
F	N	Thread, 49
fine-rasterization, 58	Nested Branch, 53	SM, 46, 55
Fixed-Function, 7	North Bridge, 24	Smoother, 1
Floating Operation Per Second	NV40, 27, 54	SPA, 44
(FLOPs), 39		Stack-Based Reconvergence, 50

Stream Processor Array (SPA),
43
Supersampling, 14
Superscalar, 36
Synchronization, 55

T

Task, 16
Task Parallelism, 16
Texture Cache, 32
Texture Mapping, 9
Texture/Processor Cluster (TPC),
43
TPC, 44, 46, 59
Transcendental, 55
Trap, 55

U

Underflow, 56
Unified Shader, 18

V

Vertex, 1
Vertex Shader, 3
viewport, 58
Viewport Transformation, 4
VLIW, 30
Voodoo, 21

W

Warp, 49

Write-Only, 25

X

Xbox360, 24
Xenos, 24

Z

z-buffer, 10
zcull, 58
Z-Cull, 34