موازی‌سازی مسئله‌ی قابلیت‌دید روی سطوح نامنظم مثلث‌بندی‌شده

حمید ضرّابی‌زاده **محمّد قدسی**

آزمایشگاه پردازش موازی

دانشکده‌ی مهندسی کامپیوتر، دانشگاه صنعتی شریف

zarrabi@ada.ce.sharif.ac.ir

ghodsi@sharif.ac.ir

چکیده

مسئله‌ی قابلیت دید، یکی از مسایل کاربردی مهم در هندسه‌ی محاسباتی است و هدف آن، پیدا کردن نواحی قابل‌دید از یک نقطه، برروی یک سطح مثلث‌بندی‌شده می‌باشد. در این مقاله، ضمن بررسی الگوریتم‌های مختلف برای حل مسئله‌ی قابلیت دید، یک الگوریتم عملی برای حل این مسئله ارائه خواهیم نمود. سپس به موازی‌سازی این الگوریتم می‌پردازیم و نتایج حاصل از شیوه‌های مختلف پیاده‌سازی موازی این الگوریتم را مورد ارزیابی قرار خواهیم داد.

کلیدواژه‌ها *: هندسه‌ی محاسباتی، پردازش موازی، قابلیت دید، TIN*

۱ مقدمه

مسئله‌ی محاسبه‌ی اطلاعات مربوط به نقشه‌ی قابلیت دید روی یک سطح مثلث‌بندی‌شده، کاربردهای عملی مهم و فراوانی دارد. یکی از این کاربردها، نمایش سطوح سه‌بعدی است که هدف از آن، ارائه‌ی یک تصویر دوبعدی از اجسام سه‌بعدی می‌باشد. این مسئله در کاربردهای گوناگون ازجمله نرم‌افزارهای مهندسی، معماری، گرافیک، *GIS* و غیره به چشم می‌خورد. جدای از مسئله‌ی نمایش، مسایل مختلف دیگری هم‌چون مسئله‌ی پیداکردن نقاط مشاهده‌ی بهینه، مکان فرستنده‌ها و گیرنده‌های رادیویی، پیدا کردن کوتاه‌ترین مسیرهای روی سطح با قابلیت دید و بسیاری دیگر از مسایل بهینه‌سازی، نیازمند داشتن اطلاعات مربوط به نقشه‌ی قابلیت دید روی یک سطح می‌باشد.

در این مقاله ابتدا به تعریف مسئله و بررسی کارهای انجام شده در این زمینه می‌پردازیم. سپس نحوه‌ی پیاده‌سازی الگوریتم ترتیبی برش افق را بررسی می‌کنیم و سپس به موازی سازی این الگوریتم خواهیم پرداخت. در نهایت با ارائه‌ی نتایج به‌دست‌آمده از اجرای الگوریتم‌های ترتیبی و موازی، به مقایسه‌ی آن‌ها می‌پردازیم و میزان تسریع الگوریتم را از جنبه‌ی عملی مورد ارزیابی قرار می‌دهیم.

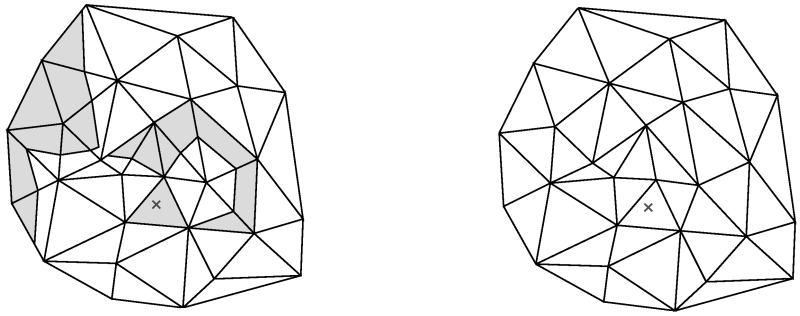
۱.۱ تعریف مسئله

یک سطح دونیم‌بعدی[۱] عبارت است از تابعی مانند $z = f(x, y)$ که بر روی یک دامنه‌ی پیوسته مانند D تعریف شده است. در کاربردهای عملی، دامنه‌ی این سطح به‌وسیله‌ی تعدادی نقاط گسسته با وجه‌های مثلثی شکل تقریب زده می‌شود که حاصلِ آن، یک سطح نامنظم مثلث‌بندی‌شده یا به‌اختصار *TIN*[۲] نامیده می‌شود. می‌گوییم دو نقطه از یک‌دیگر قابل رؤیت‌اند، اگر بتوان آن‌دو را با یک خط مستقیم به هم وصل کرد طوری‌که این خط تماماً بالای سطح قرار گیرد و سطح را فقط در این دو نقطه قطع کند. دراین‌صورت مسئله‌ی پیدا کردن نقشه‌ی قابلیت دید

---

[۱] *terrain*

[۲] *Triangulated Irregular Network*



(ب) (الف)

شکل ۱. (الف) یک *TIN* و یک نقطه‌ی دید روی آن (ب) نقشه‌ی قابلیت دید (با رنگ تیره‌تر)

عبارت است از پیدا کردن مجموعه نقاطی از سطح *TIN* که از یک نقطه‌ی دید روی سطح *TIN* قابل رؤیت باشند. در شکل ۱، یک *TIN* به‌همراه نقشه‌ی قابلیت دید آن نمایش داده شده است.

با توجه به یک نقطه‌ی دید داده‌شده‌ی V، رابطه‌ی جلو/عقب[۳] را به این صورت تعریف می‌کنیم: می‌گوییم وجه (یا یال) A جلوی وجه (یا یال) دیگر B قرار دارد، اگر یک شعاع خارج‌شونده از V، A را پیش از B قطع کند. اگر A جلوی B باشد، آن‌گاه B عقب A قرار خواهد داشت. یک سطح، ترتیب‌پذیر[۴] نامیده می‌شوند، اگر رابطه‌ی جلو/عقب بین وجه‌ها یا یال‌های آن، یک رابطه‌ی ترتیب جزیی باشد و به‌عبارتی دیگر، بتوان یک ترتیب توپولوژیکی بین وجه‌ها و یال‌های آن پیدا نمود. هر سطحی نسبت به نقاط دید خارج از دامنه‌ی خود ترتیب‌پذیر است، درصورتی‌که این موضوع برای نقاط داخل دامنه، همیشه صادق نیست. با این‌حال می‌دانیم که سطوح مثلث‌بندی‌شده دلون نسبت به هر نقطه‌ای ترتیب‌پذیرند و علاوه‌براین هر سطح ترتیب‌ناپذیر را می‌توان با تقسیم چند وجه آن ترتیب‌پذیر نمود.

۲ بررسی کارهای انجام شده

باتوجه به استفاده‌ی گسترده و روبه‌افزایش الگوریتم‌های قابلیت دید در بسیاری از کاربردهای *GIS* و همچنین نیاز به کارایی بالا در مواردی چون محاسبه‌ی بلادرنگ نقشه‌ی قابلیت دید، الگوریتم‌هایی جهت محاسبه‌ی موازی نقشه‌ی قابلیت دید ارائه گردیده است تا بتواند بازده لازم را به‌دست آورد. ازجمله بهترین پیش‌نهادهایی که تاکنون برای حل موازی این مسئله ارائه شده است، می‌توان به الگوریتم‌های *De Floriani et al.* [1]، *Reif and Sen* [5] و *Teng et al.* [6] اشاره نمود. پیچیدگی زمانی این الگوریتم‌ها و تعداد پردازنده‌های موردنیاز برای پیاده‌سازی آن‌ها در جدول ۱ آمده است. (در این جدول، منظور از $\alpha(n)$، عکس تابع آکرمان و منظور از k، اندازه‌ی خروجی الگوریتم است.)

۱.۲ الگوریتم *Reif and Sen*

الگوریتم *Reif and Sen* نخستین الگوریتم موازی برای مسئله‌ی قابلیت دید است که کار انجام‌شده در آن متناسب با اندازه‌ی خروجی است. با این‌حال استفاده از داده‌ساختارهای پیچیده و مدل محاسباتی خاص، باعث شده است که این الگوریتم، بیش‌تر جنبه‌ی نظری به‌خود بگیرد و در کاربردهای عملی کم‌تر مورد استفاده قرار گیرد.

اساس کار این الگوریتم بر روی سطوح *TIN* ترتیب‌پذیر بدین صورت است که ابتدا یال‌های سطح به دو زیرمجموعه‌ی تقریباً مساوی تقسیم می‌شود، به‌طوری‌که یک زیرمجموعه کاملاً در جلوی زیرمجموعه‌ی دیگر قرار گیرد و سپس به‌صورت بازگشتی، قسمت قابل دید در هر زیرمجموعه به‌طور موازی محاسبه می‌گردد و در نهایت قسمت قابل دید کل سطح، از روی قسمت‌های قابل دید در دو زیرمجموعه محاسبه می‌گردد.

Reif و *Sen* نشان دادند که الگوریتم آن‌ها با به‌کارگیری داده‌ساختارهای خاص، به پیچیدگی زمانی $O(\log^4 n)$ با استفاده از تعداد $O(\frac{n+k}{\log(n+k)})$ پردازنده منجر می‌شود. با این حال مدل محاسباتی به‌کار رفته توسط این الگوریتم، اندکی متفاوت با مدل معمولی *PRAM* است، چون در این مدل پیاده‌سازی فرض شده است که یک استخر از پردازنده‌های آزاد وجود دارد که می‌توان آن‌ها را در زمان اجرا به‌کار گرفت.

۲.۲ الگوریتم *Teng et al.*

این الگوریتم موازی، برپایه‌ی یک الگوریتم تقسیم‌وحل ایجاد شده است که اساس کار آن به صورت زیر است.

مرحله‌ی «تقسیم»، براساس عمق وجوه از نقطه‌ی دید انجام

---

[۳] *front/behind*

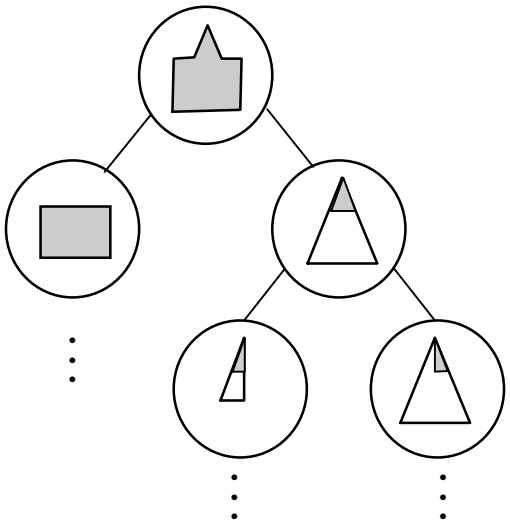
[۴] *sortable*

		Complexity	No. of processors
1988	Reif & Sen	$O(\log^4 n)$	$O(\frac{n+k}{\log(n+k)})$
1995	Teng et al.	$O(\log^2 n)$	$O(\frac{k+n\alpha(n)}{\log n})$
1994	De Floriani et al.	$O(n^2\alpha(n)/p)$	p

جدول ۱. الگوریتم‌های موازی برای پیدا کردن نقشه‌ی قابلیت دید

می‌پذیرد، بدین‌صورت‌که یک ترتیب توپولوژیکی بین وجه‌ها بر اساس رابطه‌ی جلو/عقب نسبت به نقطه‌ی دید پیدا می‌شود و سپس وجوه براساس این ترتیب، به برگ‌های یک درخت دودویی متوازن T اختصاص داده می‌شوند، به‌طوری‌که پیمایش چپ به راست برگ‌های درخت، متناظر با ترتیب جلو به عقب وجه‌ها باشد.

در مرحله‌ی «حل»، درخت دوبار پیمایش می‌شود: در پیمایش اول (پایین به بالا)، برای هر گره‌ی v، اجتماع تصاویر تمامی برگ‌هایی که در زیردرخت‌های پایین v قرار دارند (U_V)، محاسبه می‌شود. در پیمایش دوم (بالا به پایین)، برای هر گره‌ی v، قسمت قابل دید V_V از U_V محاسبه می‌شود. بدین ترتیب در



$$U_d = U_{lson(d)} \cup U_{rson(d)}$$

$$V_{root} = U_{root}$$
$$V_{lson(d)} = V_d \cap U_{lson(d)}$$
$$V_{rson(d)} = V_d - U_{lson(d)}$$

شکل ۲. مراحل اجرای الگوریتم *Teng et al.*

پایان الگوریتم، قسمت‌های قابل‌دید از هر وجه، برابر خواهد بود با مجموعه‌ی V_V که به برگ متناظر با آن وجه اختصاص داده شده است. شکل ۲ مثالی از نحوه‌ی محاسبه‌ی قسمت قابل دید در درخت را نشان می‌دهد. قسمت‌های سایه‌دار، نشان‌دهنده‌ی قمست‌های قابل‌دید (V_V) می‌باشند.

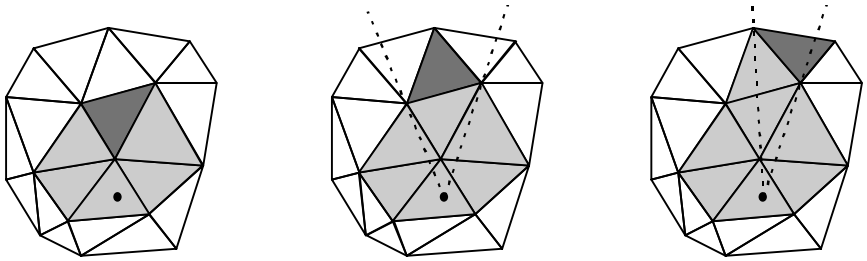
با در نظر گرفتن یک سطح از درخت و انجام موازی محاسبات مربوط به گره‌ها در آن سطح، می‌توان الگوریتم را به‌صورت موازی به اجرا درآورد. *Teng et al.* با این شیوه توانستند الگوریتمی ایجاد کنند که با کمک $O(\frac{k+n\alpha(n)}{\log n})$ پردازنده در مدل *CREW/PRAM*، زمان اجرای $O(\log^2 n)$ را تضمین نماید ولی کارایی این الگوریتم، به‌خاطر هزینه‌ی بالای ارتباط بین پردازنده‌ها و داده‌ساختارهای پیچیده، نسبتاً پایین است.

۳.۲ الگوریتم *De Floriani et al.*

یکی از الگوریتم‌هایی که طی چند سال اخیر توانسته است در مقام پیاده‌سازی عملی، کارایی قابل‌ملاحظه‌ای از خود نشان دهد، الگوریتم *De Floriani et al.* است[1]. یکی از ویژگی‌های اصلی این الگوریتم، کاهش فوق‌العاده‌ی ارتباط بین پردازنده‌ها و در نتیجه قابلیت‌حمل بالای آن می‌باشد و این ویژگی باعث شده است که الگوریتم فوق بتواند بر روی محیط‌های پردازش موازی معمول که در آن، پردازنده‌ها بیش‌تر حالت توزیع‌شده دارند، به‌سادگی پیاده‌سازی گردد و درعین حال کارایی بالای خود را حفظ نماید.

در ادامه، الگوریتم *De Floriani et al.* را به‌طور دقیق‌تر مورد بررسی قرار خواهیم داد. در بخش ۳، الگوریتم ترتیبی «برش افق»[5] را که پایه‌ی الگوریتم موازی *De Floriani et al.* است، توضیح خواهیم داد. در بخش ۴، با اعمال تغییرات لازم، الگوریتم ترتیبی را به‌صورت موازی درخواهیم آورد. در بخش ۵ نتایج حاصل از پیاده‌سازی ترتیبی و موازی الگوریتم را بررسی

[5] *horizon-cut*



شکل ۳. حالت‌های ممکن در افزایش یک مثلث به چندضلعی ستاره‌ای-شکل

می‌کنیم و نهایتاً در بخش ۶، با ارائه‌ی راه‌کارهایی درجهت افزایش میزان تسریع الگوریتم تلاش خواهیم نمود.

۳ الگوریتم ترتیبی برش افق

در این بخش فرض می‌کنیم که مدل سطح، به‌وسیله‌ی یک *TIN* بی‌دور برپایه‌ی مثلث‌بندی $\Sigma = \{t_1, t_2, \ldots, t_n\}$ تعریف شده است و نقطه‌ی دید V، درون دامنه‌ی مثلث‌بندی و بر روی سطح قرار دارد. الگوریتمی که در این بخش ارائه می‌شود، قسمت قابل‌دید از هر سطح را محاسبه می‌کند، یعنی به‌ازای هر مثلث t از Σ، زیرمجموعه‌ای متشکل از قسمت‌های چندضلعی‌شکل، که هرکدام از چندضلعی‌ها به‌طورکامل از نقطه‌ی دید V قابل‌دید باشند، معین می‌گردد. الگوریتم از دو مرحله‌ی اصلی تشکیل شده است:

- مرتب‌سازی مثلث‌ها براساس فاصله از تصویر نقطه‌ی دید در صفحه‌ی x-y (مرتب‌سازی شعاعی)
- محاسبه‌ی قسمت‌های قابل‌دید از هر مثلث (مرحله‌ی محاسبه‌ی قابلیت دید)

۱.۳ مرحله‌ی مرتب‌سازی شعاعی

مرحله‌ی مرتب‌سازی شعاعی، با ساختن یک چندضلعی ستاره‌ای-شکل Π به‌دور تصویر نقطه‌ی دید صورت می‌پذیرد، بدین‌ترتیب که در هر مرحله، به‌طور افزایشی یک مثلث به چندضلعی آغازین که از مثلث‌های مجاور نقطه‌ی دید تشکیل شده است، اضافه می‌شود. در هر مرحله، به‌خاطر خاصیت بی‌دور بودن Σ، همیشه امکان افزون یک مثلث با حفظ ویژگی ستاره‌ای‌شکل بودن چندضلعی وجود دارد. درنتیجه مثلث‌بندی Σ، در یک ترتیب تقریباً مارپیچی به‌دور نقطه‌ی دید و به‌سمت مثلث‌های مرزی مرتب می‌شوند.

در آغاز، چندضلعی Π متشکل از اجتماع مثلث‌هایی است، که نقطه‌ی دید V درون و یا روی مرزهای آن‌ها قرار دارد. در هر مرحله‌ی بعد، یک یال l روی مرز Π انتخاب می‌شود و مثلث T که از بیرون مجاور l است، مورد آزمون قرار می‌گیرد. دو حالت ممکن وجود دارد:

- مثلث T از طریق دو یال با Π مجاور است. این‌چنین مثلثی را یک مثلث فرورفته می‌نامیم و مثالی از آن در شکل ۳ .الف نشان داده شده است. مثلث‌های فرورفته می‌توانند به‌راحتی به‌عنوان مثلث بعدی، به چندضلعی Π اضافه شوند.
- مثلث T از طریق یک یال یعنی همان یال l با Π مجاور است. این‌چنین مثلثی را یک مثلث برآمده می‌نامیم. مثلث‌های برآمده فقط و فقط در صورتی می‌توانند به Π اضافه شوند که رأس مقابل آن‌ها، درون قطاع شعاعی تعریف‌شده توسط V و l قرار گیرد (شکل ۳ .ب) درغیراین‌صورت T به Π اضافه نمی‌شود، زیرا خاصیت ستاره‌ای‌شکل بودن را حفظ نخواهد کرد (شکل ۳ .ج) و این مثلث مجدداً در مراحل بعدی مورد آزمون قرار خواهد گرفت.

مثلث‌های برآمده‌ای که رد می‌شوند، مجدداً هنگامی مورد آزمون قرار می‌گیرند که یال دیگر آن مثلث‌ها طی مرحله‌ای به چندضلعی ستاره‌ای‌شکل افزوده شده باشد و بنابراین طی آزمون مجدد، مثلث برآمده به مثلث فرورفته مبدل می‌گردد و بدون هیچ شرطی به Π اضافه می‌شود. بنابراین واضح است که طی مراحل مرتب‌سازی شعاعی، هر مثلث حداکثر دوبار مورد آزمون قرار می‌گیرد و بنابراین پیچیدگی زمانی این مرحله، به‌طور خطی متناسب با تعداد مثلث‌ها خواهد بود. به‌عبارت دیگر، اگر تعداد مثلث‌ها در Σ برابر n باشد، آن‌گاه مرتبه‌ی زمانی مرحله‌ی مرتب‌سازی شعاعی $O(n)$ خواهد بود.

۲.۳ مرحله‌ی محاسبه‌ی قابلیت دید

در این قسمت مثلث‌ها براساس ترتیب شعاعی مورد آزمون قرار می‌گیرند و در هر مرحله با استفاده از افق جاری، قسمت قابل دید مثلث جدید محاسبه شده و افق جاری با افزودن مثلث

جدید به‌روز می‌گردد.

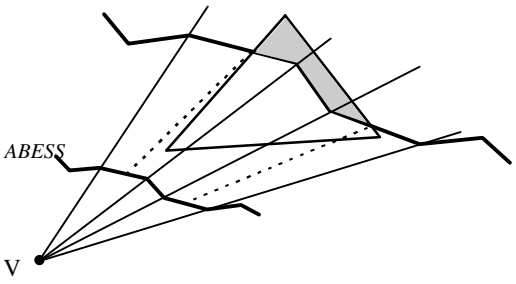
برای توضیح دقیق‌تر این مرحله لازم است که چند تعریف جدید را معرفی نماییم. می‌گوییم یک وجه نسبت به نقطه‌ی دید V «روبه‌بالا»[۶] است، اگر قسمت خارجی آن وجه، به‌طرف نقطه‌ی دید باشد درغیراین‌صورت وجه را «روبه‌پایین»[۷] می‌نامیم. همچنین با فرض این‌که e، یک یال و وجوه f_1 و f_2 دو وجه مجاور e باشند، به‌طوری‌که f_1 جلوتر از f_2 قرار داشته باشد، دراین‌صورت e را یک یال بلوکی می‌نامیم، اگر f_1 روبه‌بالا و f_2 روبه‌پایین باشد.

این الگوریتم، یک لیست دوّار پویا از یال‌های بلوکی نگه‌داری می‌کند که آن‌را *ABESS*[۸] می‌نامیم. *ABESS* شامل قسمت‌هایی از یال‌های بلوکی مثلث‌هایی است که تا کنون ملاقات شده‌اند و می‌توانند روی مثلث‌هایی که ملاقات نشده‌اند، سایه بیندازند. در آغاز، *ABESS* با یک لیست تهی مقداردهی اولیه می‌شود و در ادامه، مثلث‌های S یکی پس از دیگری، براساس ترتیب شعاعی پردازش می‌شوند تا قسمت‌های قابل دید آن‌ها محاسبه گردد. پردازش مثلث جاری شامل دو قسمت است:

- تعیین قسمت‌های قابل دید: اگر T نسبت به V روبه‌پایین باشد، تماماً غیر قابل رؤیت خواهد بود. درغیراین‌صورت تمامی پاره‌خط‌هایی از *ABESS* را که روی T تصویر می‌شوند، در نظر می‌گیریم و با استفاده از آن، قسمت‌های قابل‌دید مثلث جاری را که بالای تصویر *ABESS* قرار دارد، به‌دست می‌آوریم. (شکل ۴)
- به‌روزآوری *ABESS*: تمام یا قسمت‌هایی از پاره‌خط‌های *ABESS* که درون T تصویر می‌شوند، با یال‌های بالایی متناظر T یا قسمت‌هایی از آن یال‌ها جایگزین می‌شوند.

یک مسئله‌ی جدی در الگوریتم فوق، تکه شدن‌های متوالی *ABESS* است، زیرا پیچیدگی زمانی این قسمت از الگوریتم، به‌طور خطی متناسب با اندازه‌ی لیست *ABESS* می‌باشد و این اندازه در بدترین حالت می‌تواند رشد سریعی داشته باشد، ولی ازآن‌جاکه در هر مرحله از پردازش و در هر به‌روزآوری *ABESS*، تعدادی از پاره‌خط‌های آن که زیر تصویر *ABESS* روی مثلث جاری قرار دارند، حذف می‌شوند، بنابراین این رشد در حالت متوسط بسیار کم‌تر خواهد بود.

De Floriani et al. نشان داده‌اند که طول لیست *ABESS* در بدترین حالت می‌تواند به $O(n\alpha(n))$ برسد و ازآن‌جاکه $O(n)$ مثلث باید پردازش شوند، در نتیجه زمان موردنیاز برای محاسبه‌ی نواحی قابل‌دید در بدترین حالت $O(n^2\alpha(n))$ است.



شکل ۴. *ABESS* و مثلث در حال پردازش

یکی از نکاتی که می‌تواند در پیاده‌سازی بهتر این الگوریتم کمک شایانی داشته باشد، توجه به ساختار خاص مسئله است. یکی از نکات مهم این است که تصویر *ABESS* فقط می‌تواند با یال‌های فعال T، یعنی یال‌هایی که جزیی از لیست Π جاری نیستند، تقاطع داشته باشد، و این یک ساده‌سازی باارزش است زیرا درمورد مثلث‌های فرورفته، لازم است که فقط یک یال جهت تقاطع با هر قسمت *ABESS* مورد آزمون قرار گیرد و درمورد مثلث‌های برآمده نیزاین تعداد تنها به دو یال خواهد رسید.

لازم به ذکر است که در الگوریتم فوق، مراحل مرتب‌سازی شعاعی و محاسبه‌ی قابلیت دید، به‌عنوان دو مرحله‌ی جدا از هم در نظر گرفته شده‌اند، درصورتی‌که در پیاده‌سازی عملی الگوریتم، این دو مرحله هم‌زمان اجرا می‌گردند، یعنی به‌محض این‌که یک مثلث توسط مرتب‌ساز شعاعی انتخاب گردید، بلافاصله از جهت قابلیت دید مورد آزمون قرار می‌گیرد.

۴ موازی‌سازی الگوریتم برش افق

راه‌کارهای تبدیل یک الگوریتم ترتیبی به یک الگوریتم موازی به‌طورکلی به دو دسته‌ی تقسیم داده و تقسیم عمل‌کرد افراز می‌شوند. در تقسیم داده (یا موازات داده‌ای) داده‌های ورودی یا فضای خروجی بین پردازنده‌ها تقسیم می‌شود و هر پردازنده به‌طور مستقل پردازش‌های موردنیاز را بر روی داده‌های خود انجام می‌دهد و درنهایت جواب کل مسئله از اجتماع جواب‌های به‌دست‌آمده توسط هر پردازنده به‌دست می‌آید.

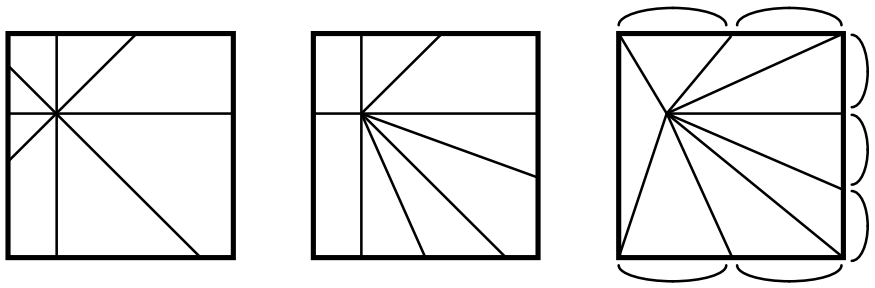
در این بخش یک الگوریتم موازی برای حل مسئله‌ی نقشه‌ی قابلیت دید با استفاده از موازات داده‌ای ارائه خواهیم کرد. الگوریتم برش افق را می‌توان به‌سادگی، با تقسیم دامنه‌ی ورودی آن به‌صورت موازی درآورد، ولی این تقسیم باید به‌گونه‌ای باشد که امکان پردازش مستقل برای کلیه‌ی پردازنده‌ها فراهم شود. این‌کار با تقسیم دامنه‌ی ورودی به قطاع‌هایی که همگی در نقطه‌ی دید V اشتراک دارند، انجام می‌گردد. بدین ترتیب

---

[۶] *face up*

[۷] *face down*

[۸] *Active Blocking Edge Segment Sequence*



شکل ۵. شیوه‌های مختلف افراز داده‌های ورودی

محاسبه‌ی قابلیت دید می‌تواند به‌طور مستقل در هر قطاع انجام شود، زیرا تمام مثلث‌هایی که ممکن است در قابلیت دید یک مثلث دخیل باشند، همگی در قطاعی مربوط به همان مثلث قرار می‌گیرند.

۱.۴ شیوه‌های افراز داده‌های ورودی

یکی از نکات مهم برای افزایش کارایی الگوریتم‌های موازی، ایجاد توازن بار بر روی پردازنده‌ها است. به‌همین‌خاطر تقسیم متوازن داده‌های ورودی، به‌نحوی که تعداد مثلث‌ها در هر قطاع تقریباً مساوی باشد، از اهمیت ویژه‌ای برخوردار است. البته باید توجه داشت که تقسیم داده‌ها به‌طوردقیق عملاً امکان‌پذیر نیست، زیرا این کار نیازمند داشتن اطلاعات دقیق از نقشه‌ی سطح مثلث‌بندی‌شده است و هزینه‌ی این‌چنین افرازی نیز بسیار زیاد خواهد بود. به‌همین خاطر ما چند شیوه‌ی تقریبی برای افراز داده‌ها در ادامه‌ی این بخش ارائه خواهیم کرد.

یک شیوه‌ی بدیهی برای افراز داده‌ها، تقسیم دامنه به m قطاع با زاویه‌های مساوی است (شکل ۵ .الف). وجوهی که روی مرز تقسیم قرار می‌گیرند، در هر دو افراز تکرار می‌شوند. این شیوه بسیار ساده است و در مواقعی که نقطه‌ی دید در نزدیکی مرکز دامنه قرار نداشته باشد، نتیجه‌ی مطلوبی نخواهد داشت.

در شکل ۵ .ب شیوه‌ی دیگری برای افراز نشان داده شده است. در این شیوه ابتدا دامنه به چهار تکه‌ی مستطیل‌شکل از مرکز نقطه‌ی دید تقسیم می‌شود. سپس نسبت r_i بین مساحت‌های چهار ناحیه محاسبه می‌شود و براساس این نسبت، هریک از چهار ناحیه مجدداً به $m \times r_i$ ناحیه با زاویه‌های مساوی تقسیم می‌شود. این شیوه بسیار بهتر از شیوه‌ی قبلی عمل می‌کند ولی در مواردی که نقطه‌ی دید در نزدیکی مرکز دامنه قرار نداشته باشد، بازهم نتیجه‌ی چندان مطلوبی نخواهد داشت.

شیوه‌ی سوّمی که در شکل ۵ .ج نشان داده شده است، به این صورت است که ابتدا دامنه، توسط چهار خط که نقطه‌ی دید را به رأس‌های دامنه متصل می‌کند، به چهار قطاع تقسیم می‌شود. سپس هریک از این چهار قطاع مجدداً به $m \times r_i$ قطاع کوچک‌تر تقسیم می‌شود، ولی این‌بار تقسیم به‌نحوی انجام می‌پذیرد که تکه‌های به‌وجود آمده روی هر ضلع دامنه، برابر باشند. این شیوه در عین سادگی و محاسبات غیرپیچیده، توازن بار خوبی روی پردازنده‌ها ایجاد می‌کند.

باید توجه داشت که پردازش کارا، تنها به مشخصات افراز بستگی ندارد، بلکه نحوه‌ی مثلث‌بندی سطح نیز در این مسئله بسیار مؤثر است. به‌عبارت دیگر اگر حتی شیوه‌ای برای افراز داده‌ها پیدا کنیم که قطاع‌هایی با مساحت‌های کاملاً مساوی برای ما ایجاد کند، باز هم ممکن است که به یک عدم توازن بار روی پردازنده‌ها دچار شویم. روی همین اصل و همچنین با توجه به این‌که شیوه‌های افراز دقیق‌تر، معمولاً هزینه‌ی محسباتی بالاتری دارند، به‌نظر می‌رسد که شیوه‌های دقیق‌تر، در عمل کاربرد کم‌تری داشته باشند.

۲.۴ الگوریتم موازی برای محاسبه‌ی قابلیت دید

برای تبدیل الگوریتم برش افق به یک الگوریتم موازی، کافی است چند تغییر کوچک در این الگوریتم ایجاد نماییم. داده‌های ورودی در زمان مقداردهی اولیه روی پردازنده‌ها بار می‌شود. تصمیم ما در پیاده‌سازی موازی به‌این صورت است که کل دامنه‌ی ورودی درآغاز کار روی تمام پردازنده‌ها بار شود تا بتوان الگوریتم را به‌طور متوالی و برای نقاط دید مختلف، بدون نیاز به بار کردن مجدد اطلاعات اجرا نمود.

الگوریتم برش افق موازی، ابتدا مثلث‌های درون قطاع محلی خود را تعیین می‌کند. سپس مثلث‌هایی را که روی مرز تقسیم‌بندی قرار دارند، با یک «نشان مرز» مشخص می‌نماید. در پیاده‌سازی موازی الگوریتم، $ABESS$ و Π به‌صورت لیست‌های دوطرفه‌ی غیردوّار پیاده‌سازی می‌شوند و بدیهی است که این لیست‌ها فقط با مثلث‌هایی که به‌طور کامل یا جزیی در قطاع

محلی قرار دارند، سروکار دارند.

پس از پایان مرحله‌ی مقداردهی اولیه، پردازش مربوط به محاسبه‌ی قابلیت دید آغاز می‌شود. تغییر عمده‌ای که در این قسمت باید صورت پذیرد، نحوه‌ی کنترل انتهای لیست‌های *ABESS* و Π می‌باشد، به‌طوری‌که وقتی به انتهای این دو لیست می‌رسیم، جهت حرکت در لیست عوض شود. تغییر دیگر در نحوه‌ی مدیریت برش مثلث‌ها توسط *ABESS* در زمانی است که مثلث جاری، یک مثلث مرزی باشد، چون در این مورد، قسمت قابل‌دید در مثلث به دو قطاع مجاور بستگی دارد. این مسئله را هم می‌توان به‌سادگی مرتفع نمود، بدین‌ترتیب که هر قطاع، قسمت قابل دید در ناحیه‌ای از مثلث را که درون آن قطاع گرفته است، محاسبه می‌کند و در انتها، ناحیه‌ی قابل دید مثلث، از اجتماع این دو قسمت به‌دست خواهد آمد.

۵ نتایج پیاده‌سازی

نتایج حاصل از پیاده‌سازی الگوریتم برش افق، به‌صورت ترتیبی و موازی در جداول زیر آمده است. برای اجرای موازی این الگوریتم، از محیط *LAM/MPI* با چهار پردازنده‌ی پنتیوم ۲۰۰ استفاده شده است.

تمام زمان‌های نشان شده در جدول‌های زیر، درحقیقت میانگین زمان محاسبه‌ی ۱۰۰ اجرای متوالی این الگوریتم، در نقاط دید مختلف روی یک *TIN* یکسان می‌باشند. این نقاط به‌صورت تصادفی درون دامنه‌ی *TIN* انتخاب می‌شوند.

جدول ۲ زمان اجرای الگوریتم ترتیبی برش افق را به‌ازای *TIN* های دارای تعداد نقاط متفاوت نشان می‌دهد.

No. of nodes	100	200	300	400	500	1000
Time (sec.)	0.83	2.30	3.36	4.43	6.08	12.71

جدول ۲. نتایج حاصل از اجرای الگوریتم ترتیبی برش افق

همان‌طور که در جدول ۲ مشاهده می‌شود، زمان اجرای الگوریتم برش افق در عمل، به صورت خطی افزایش می‌یابد که نسبت به پیچیدگی $O(n^2a(n))$ که برای بدترین حالت این الگوریتم محاسبه کرده بودیم، بسیار بهتر است.

در جدول ۳ نتایج حاصل از اجرای الگوریتم موازی، به ازای دو شیوه‌ی افراز داده با زاویه‌های مساوی و ناحیه‌های مساوی آورده شده است. لازم به ذکر است که در آزمون ما، به‌دلیل محدودیت در تعداد پردازنده‌های سیستم مورد استفاده، شیوه‌های چهارتکه و زاویه‌های مساوی عملاً به یک شیوه منتهی می‌شوند و درنتیجه فقط یکی از آن‌ها ذکر گردیده است.

در نمودار شکل ۶ می‌توان به‌خوبی نحوه‌ی رشد زمان اجرای الگوریتم و همچنین میزان تسریع دو شیوه‌ی مختلف موازی‌سازی را مشاهده نمود.

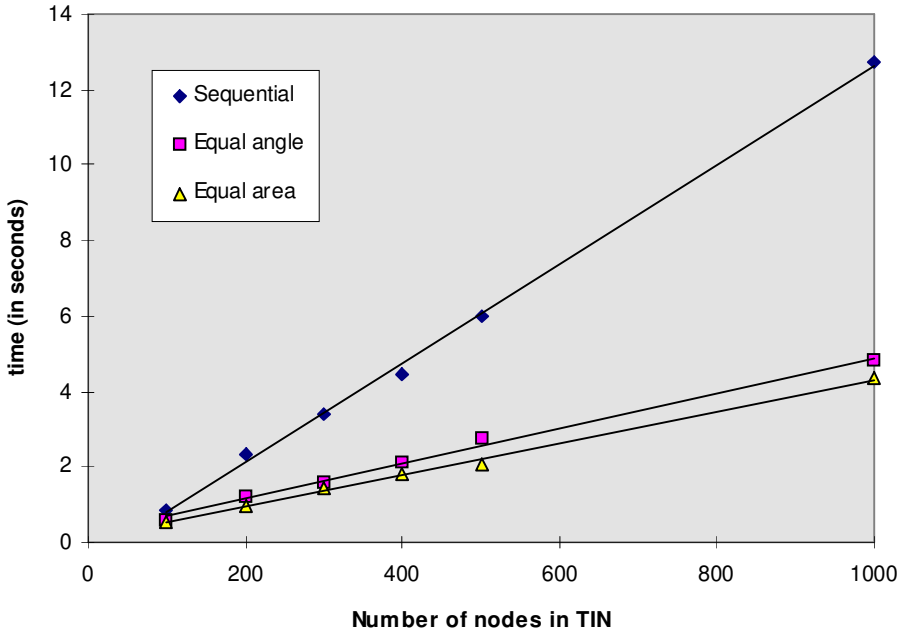
۶ تلاش جهت افزایش میزان تسریع

همان‌طور که در جدول پیش ملاحظه شد، شیوه‌های ساده‌ی افراز داده‌های ورودی توانستند میزان تسریع خوبی برای الگوریتم موازی ایجاد کنند. چنین به‌نظر می‌رسد که شیوه‌های ساده‌ی دیگری که بتواند درعین عدم پیچیدگی، تقسیم متوازن‌تری ازلحاظ مساحت قسمت‌های افرازشده ایجاد نماید، بتوانند میزان تسریع را افزایش دهند. با این ایده، روش دیگری برای افراز داده‌های ورودی مورد آزمون قرار گرفت که نتایج آن در ادامه آمده است.

اساس این روش که آن‌را روش مساحت‌های مساوی نامیده‌ایم، به این صورت است که ابتدا هر ضلع دامنه که به‌صورت مربع فرض شده است، به مضربی از تعداد پردازنده‌ها تقسیم می‌گردد (شکل ۷) و سپس طی یک پیمایش شعاعی از نقطه‌ی دید، با جهش روی قطعه‌ها مساحت افراز جدید محاسبه می‌گردد و چنان‌چه این مساحت مساوی یا بیش‌تر از مساحت کل دامنه، تقسیم بر تعداد پردازنده‌ها شد، افراز صورت می‌پذیرد و کار

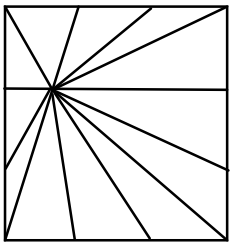
	Equal angle		Equal area	
No. of nodes	time (sec.)	speed up	time (sec.)	speed up
100	0.57	1.47	0.53	1.56
200	1.20	1.94	0.93	2.49
300	1.57	2.15	1.43	2.35
400	2.10	2.12	1.80	2.48
500	2.73	2.20	2.07	2.90
1000	4.80	2.65	4.33	2.94

جدول ۳. نتایج حاصل از اجرای موازی الگوریتم برش افق



شکل ۶. نمودار نتایج اجرای ترتیبی و موازی الگوریتم برش افق

برای افراز بعدی ادامه می‌یابد.



شکل ۷. شیوه‌ی افراز مساحت‌های مساوی

یک ویژگی در این شیوه، نحوه‌ی پیاده‌سازی کم‌هزینه‌ی آن است. پیاده‌سازی اولیه که با استفاده از توابع کتاب‌خانه‌ای *LEDA* صورت گرفته بود، به‌خاطر هزینه‌ی بالای آن به نتیجه‌ای بدتر از حالت ترتیبی انجامید. با استفاده از این اصل که مساحت مثلث‌هایی که درحین پیمایش یک ضلع آزموده می‌شوند همگی مساوی هستند (به‌خاطر ارتفاع و قاعده‌ی یکسان) می‌توان عمل پیمایش را بسیار ساده‌تر و با استفاده از چند عمل جمع وضرب ساده انجام داد و این‌کار بهبود قابل ملاحظه‌ای در رفتار الگوریتم ایجاد می‌نماید. همچنین فاکتور مهم دیگری که در این الگوریتم وجود دارد، ضریب تقسیم ضلع‌هاست که یک پارامتر بسیار مؤثر در بازده الگوریتم به حساب می‌رود. نقش این ضریب در آزمایش‌های انجام شده در جدول ۴ و نمودار شکل ۸ نشان داده شده است.

همان‌طور که مشاهده می‌شود، افزایش ضریب، از یک‌طرف باعث افزایش هزینه‌ی محاسبات خواهد شد، ولی در عوض میزان رشد زمان اجرا (شیب خط برازش) را کاهش می‌دهد و

No. of nodes	speed up			
	co = 4	co = 8	co = 16	co = 32
1000	1.93	1.96	1.94	1.91
2000	2.70	2.67	2.76	2.76
3000	2.84	2.85	2.73	2.85
4000	2.68	2.87	2.85	2.83
5000	3.04	3.03	3.06	3.10

جدول ۴. نتایج حاصل از شیوه‌ی افراز مساحت‌های مساوی با ضرایب مختلف

این باعث خواهد شد که در *TIN* های دارای نقاط زیاد، شیوه‌ی افراز مذکور بتواند میزان تسریع خوبی ایجاد نماید. آزمایش‌های انجام‌شده روی *TIN* هایی با تعداد نقاط ۱۰۰۰ تا ۱۰۰۰۰ نقطه نشان داد که افزایش ضریب تا حدود ۳۲ در کاهش شیب مؤثر است و افزایش بیش‌تر آن، تغییرات محسوسی را ایجاد نخواهد کرد.

در مجموع و در مقایسه با دو شیوه‌ی افراز پیشین، می‌توان گفت که شیوه‌ی افراز با مساحت‌های مساوی بسیار بهتر از شیوه‌ی زاویه‌های مساوی عمل می‌کند ولی نمی‌توان این شیوه را بهتر از شیوه‌ی ناحیه‌های مساوی دانست. به‌هرحال هنگامی‌که تعداد نقاط *TIN* زیاد می‌شود، این دو شیوه نتایج تقریباً یکسانی دارند.

۷ خلاصه

در این مقاله، الگوریتم‌های موازی مختلف برای حل مسئله‌ی پیدا کردن نواحی قابل‌دید از یک نقطه بر روی یک سطح مثلث‌بندی‌شده را مورد بررسی قرار دادیم. سپس یک الگوریتم عملی برای حل این مسئله ارائه کردیم و شیوه‌های مختلف را برای موازی‌سازی این الگوریتم ترتیبی بررسی نمودیم. سپس نتایج حاصل از هریک از شیوه‌های پیاده‌سازی را از لحاظ میزان تسریع مورد ارزیابی قرار دادیم و با ارائه‌ی روش‌هایی سعی در افزایش میزان تسریع الگوریتم موازی نمودیم.

مراجع

[1] L. De Floriani, C. Montanni and R. Scopigno, *Parallelizing visibility computation on trangulated terrains*, Int. J. Geometrical Information Systems, 1994, vol. 8, no. 6, pp. 515-531.

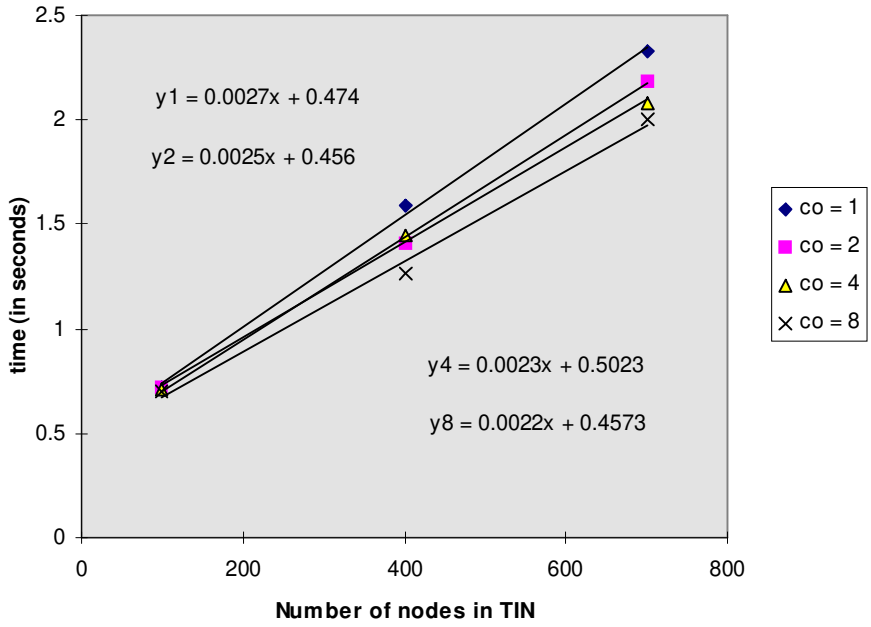
[2] L. De Floriani, P. Magillo, *Visibility algorithms on triangulated digital terrain models*, Int. J. Geometrical Information Systems, 1994, vol. 8, no. 1, pp. 13-41.

[3] L. De Floriani, P. Magillo, *Representing the Visibility Structures of a Polyhedral Terrain through a Horizon Map*, Int. J. Geometrical Information Systems, 1996, vol. 10, no. 15, pp. 541-562.

[4] P. Magillo, E. Puppo, *Algorithms for parallel terrain modeling and visualiztion*, Parallel Processing Alg. for GIS, 1996, pp. 1-24.

[5] J. H. Reif and S. Sen, *An efficient output-sensetive algorithm and its parallelization*, In 4th ACM Symposium in Computational Geometry (A.C.M. Press), 1998, pp. 193-200.

[6] Y. A. Teng, D. Mount, E. Puppo, L. S. Davis, *Parallelizing an algorithm for visibility on polyhedral terrain*, Int. J. Computational Geometry & Applications, 1994, pp. 1-9.



شکل ۸. نمودار نتایج اجرای موازی الگوریتم با شیوه‌ی افراز مساحت‌های مساوی

(اندیس y – ضریب تقسیم ضلع)