بسمه تعالی

افراز فضای دودویی
Binary Space Partitions
(BSP)

دانشگاه یزد
علوم کامپیوتر
فرزانه مرادی
۱۳۹۰

مقدمه:

اولین تجربه پرواز خلبانان بصورت شبیه سازی بروی زمین انجام می گیرد که این موضوع برای شرکتهای هوایی ، خود خلبان و محیط بسیار به صرفه است .که در اینجانکته مهم تجسم محیطی است که خلبان بالای آن در حال پرواز است. که این شامل مدل سازی منظره و اجرای مدل است.

- برای اجرای این طرح باید برای هر پیکسل در صفحه نمایش یک شی ء در آن قابل رؤیت گردد که به این پوشش تمام سطح [۱] می گویند.
- همچنین باید به محاسبه جزئیات وشدت ،تأثیر و بازتاب نور و همینطور بلندی واقعی اجسام بپردازیم .
- پرواز شبیه سازی باید در زمان واقعی عمل کند و زمان کافی برای محاسبه جزئیات نیست بنا براین پوشش تمام سطح و یک فناوری سریع ساده سازی جزئیات [۲] مهمترین عوامل در زمان اجرا هستند.

---

[۱] hidden surface removal

[۲] simple shading technique

الگوریتم z-buffer:

ابتدا طرح طوری منتقل می شود که جهت مثبت محور z هاجهت دید است واشیاء در یک ترتیب اختیاری رؤیت و تبدیل می شوند، که تعداد پیکسل های مورد نیاز برای پوشش یک تصویر تعیین می شود.

الگوریتم فرایند اجسام را در دو بافر نگهداری می کند:

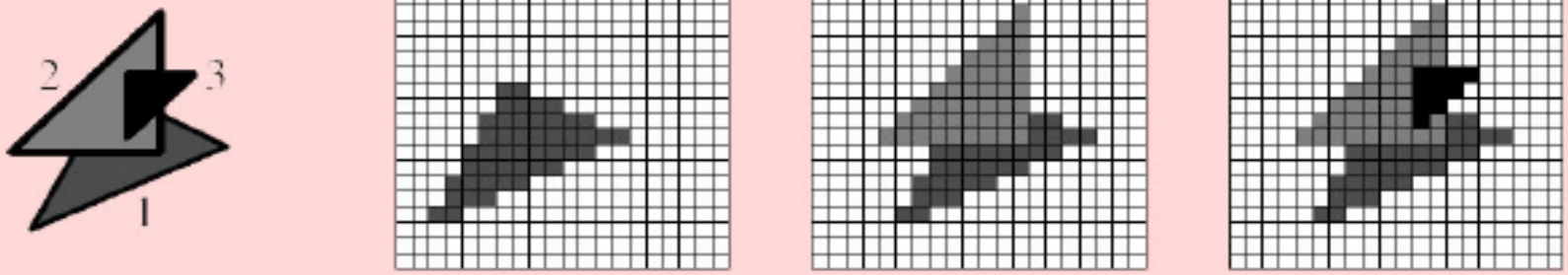
- *frame buffer* :در هر پیکسل درجه جسم در حال رؤیت را ذخیره می کند.
- $z-buffer$:مختصات z جسم در حال رؤیت را ذخیره می کند.

فرض کنیم موقع رؤیت و تبدیل یک شیء ،یک پیکسل را انتخاب می کنیم. اگر مختصات z شی ء در پیکسل کوچکتر از مختصات z ذخیره شده در $z-buffer$ باشد آنگاه جسم جدید جلوتر از جسم در حال رؤیت قرار می گیرد.بنابراین درجه جسم جدید را در *frame buffer* ومختصات آن رادر $z-buffer$ می نویسیم. اگر مختصات z شی ء در پیکسل بزرگتر از مختصات z ذخیره شده در $z-buffer$ باشد آنگاه جسم جدید قابل رؤیت نیست و *frame buffer* و $z-buffer$ بدون تغییر می مانند.

- عیب الگوریتم $z-buffer$ این است که باعث هزینه های اضافی می گردد .
- الگوریتم نقاش که در زیر معرفی می گردد هزینه اضافی ندارد.

الگوریتم نقاش:

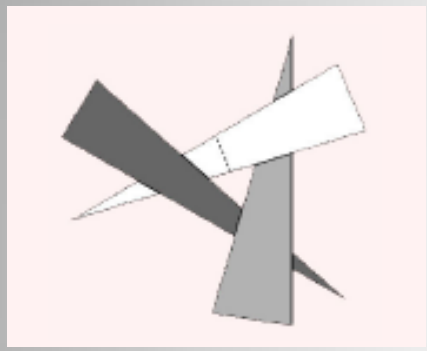
دراین الگوریتم از دورترین شیء از نقطه دید شروع ودر ترتیب عمقی و با توجه فاصله تا نقطه دید اشیاء را مرتب می کنیم.وقتی شیء رؤیت و تبدیل می شود مختصات z را بررسی نمی کنیم اما درجه را در *frame buffer* می نویسیم . و در صورتی که قبلا پر شده روی داده های قبلی می نویسیم.



که تصویر سمت چپ اشیاء با ترتیبی که رؤیت و تبدیل شده اند شماره گذاری شده اند.
این فرایند شبیه کار نقاشانی است که نقاشی را به صورت لایه ای روی همدیگر قرار می دهند.

الگوریتم نقاش:

- گاهی ممکن است یک ترتیب عمقی موجود نباشد،مثلا اشیا یک دور ایجاد نمایند.در این حالت بایستی با شکافتن یک یا بیشتر اجسام دور را بشکنیم.



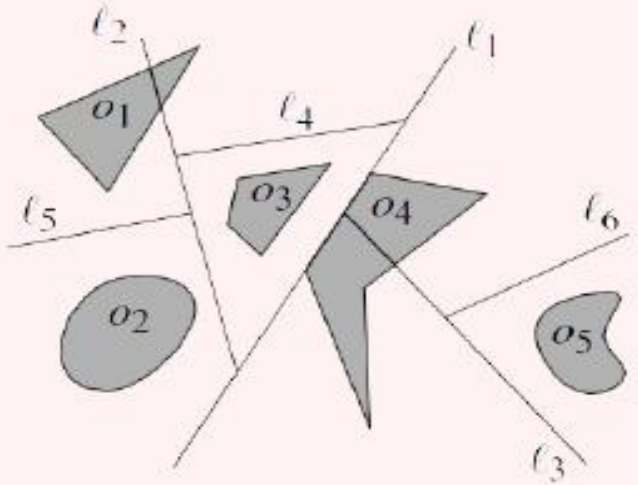
مثال:وقتی سه مثلث موجودند که یک دور ایجاد می کنند با شکافتن یکی از آنها یک مثلث ویک چهار ضلعی تولید می کنیم که یک ترتیب از نمایش درست این چهار جسم وجود دارد.

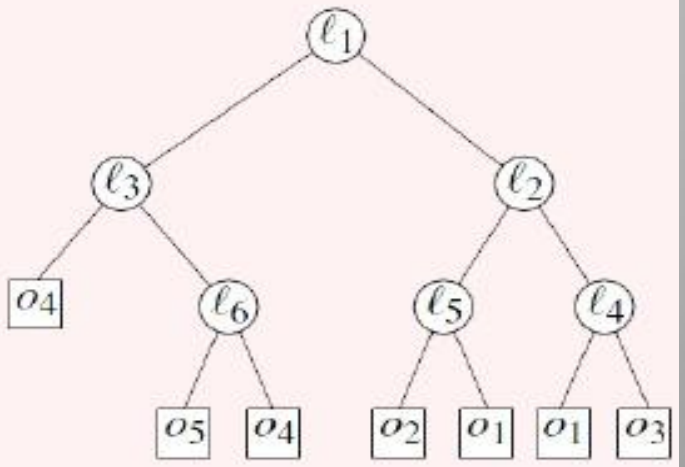
- **نکته:**

محاسبه اینکه کدام شی باید شکافته شود و مرتب سازی آن فرآیندی پرهزینه است.زیرا ترتیب به نقطه دید بستگی دارد و نقطه دید متحرک است.
اگر بخواهیم از الگوریتم نقاش در یک زمان واقعی ومحیط شبیه سازی شده استفاده کنیم باید به عنوان یک پیش فرآیند طرح را به صورت ترتیب نمایش درست و سریع از هر نقطه دید داشته باشیم.که زیباترین ساختمان داده برای این کار Binary space partitions یا BSP tree است.

تعریف درخت های BSP:

- اگر مجموعه ای از اشیاءدر یک نقشه داشته باشیم ابتدا ، نقشه را با خط l_1 سپس نیم صفحه بالای خط l_1 را با خط l_2 و نیم صفحه پایین خط l_1 را با خط l_3 می شکافیم و...
- خطوط شکاف نه تنها نقشه را بخش بندی می کندبلکه گاهی اشیاء را نیزقطعه بندی نماید. و این فرایند تا زمانی که در هر ناحیه فقط یک قطعه از یک شیء قرار گیرد ادامه دارد.
- این فرایند یک درخت را مدل سازی می کند که هر برگ درخت متناظر با یک زیر بخش نهایی است وقطعات اشیاء در برگها ذخیره می شوند.





درخت BSP در فضای d بعدی

ابر صفحه:

ابر صفحه $h : a_1x_1 + a_2x_2 + \ldots + a_dx_d + a_{d+1} = 0$ فضا را به دو نیم فضای $h^+ = \{(x_1, x_2, \ldots, x_d) : a_1x_1 + a_2x_2 + \ldots + a_dx_d + a_{d+1} > 0$ و $h^- = \{(x_1, x_2, \ldots, x_d) : a_1x_1 + a_2x_2 + \ldots + a_dx_d + a_{d+1} < 0$ تقسیم می کند.

درخت BSP در فضای d بعدی:

یک درخت BSP مانند T برای مجموعه S از اشیاءدر فضای d بعدی :

- اگر $card(S) \leqslant 1$ آنگاه T یک برگ است .که اگر این برگ v باشد مجموعه ذخیره شده در برگ(که ممکن است خالی باشد)را با $S(v)$ نشان می دهیم.
- واگر $card(S) > 1$ انگاه درریشه v از درخت T یک ابر صفحه h_v نگهداری می شود که بچه چپ v ریشهٔ BSPدرخت T^- برای مجموعه $S^- = \{h_v^- \cap s : s \in S\}$ وبچه راست v ریشهٔ BSPریشهٔ T^+ برای مجموعه $S^+ = \{h_v^+ \cap s : s \in S\}$ است.

اندازه درخت BSP:

- اندازه درخت BSP جمع کل اندازه مجموعه های S(V) روی کلیه ندهایv از درخت است یا تعداد کلی قطعاتی ازاشیا که تولید می شود.

- اگر BSP بجز خطوط شکاف شامل چیزی نیست،آنگاه تعداد گره های درخت حداکثر خطی و اندازه درخت BSP است.

- اندازه درخت BSP اطلاعی درمورد مقدار حافظه مورد نیاز برای یک قطعه نمی دهد.

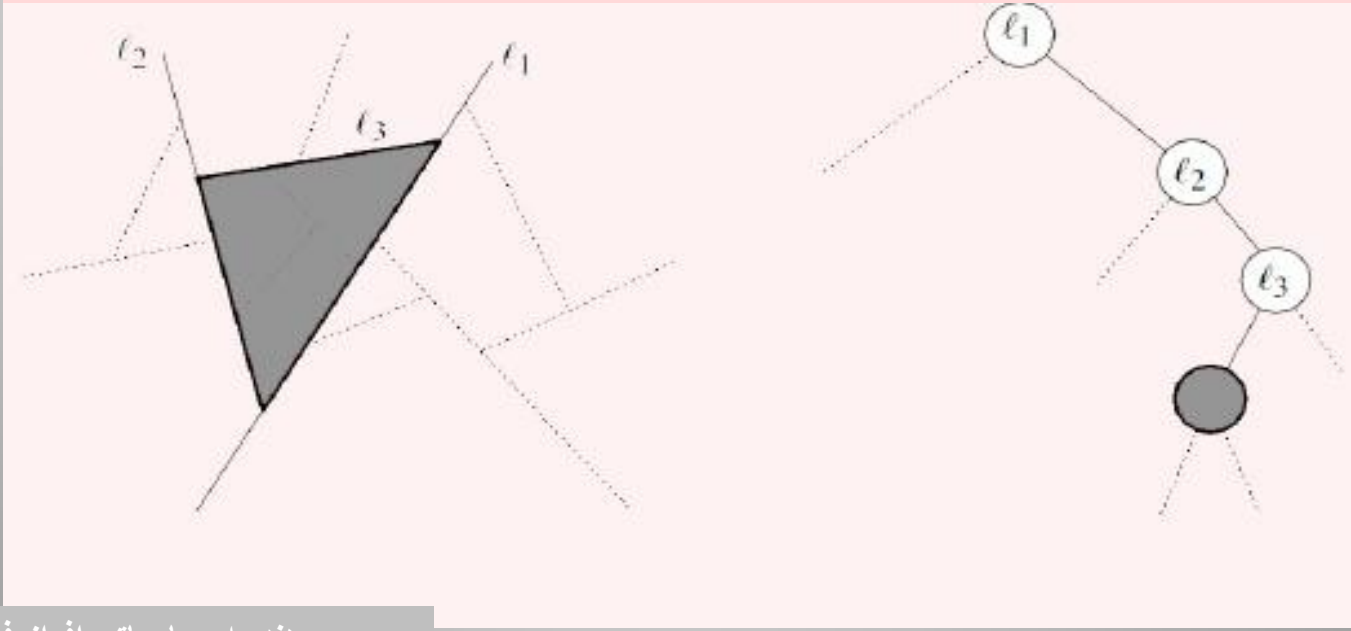
- اندازه درخت BSP معیار خوبی برای مقایسه کیفیت درختهای BSP متفاوت برای یک مجموعه داده شده از اشیاء است.

اندازه درخت BSP:

- برگها در یک درخت BSP یک $face$ در یک زیر بخش است که BSP ایجاد می کند.
- به هر گره v در درخت BSP مانند T یک ناحیه محدب متناظر می شود. که این ناحیه تقاطع نیم فضاهای $h_\mu^{\square}$ است که μ جد v واگر v در زیر در خت چپ μ باشد $\square = -$ واگر v در زیر در خت راست μ باشد $\square = +$ است.

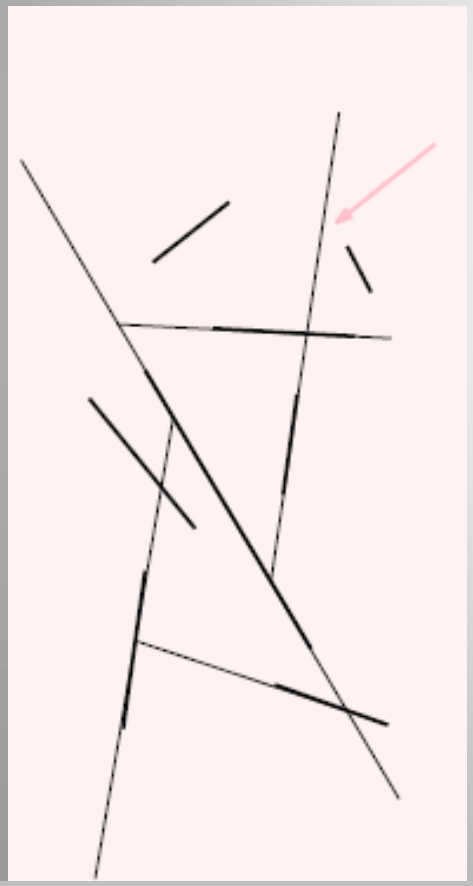
- **مثال:**

در شکل زیرگره خاکستری متناظر با ناحیه خاکستری $l_1^+ \cap l_2^+ \cap l_3^-$ است.



BSP خودافراز

شکافتن ابر صفحات که در BST استفاده می شود اختیاری است.
مطلوب است که مجموعه های مجاز شکافت ابر صفحات را محدود کنیم.



مثال:

می خواهیم BSP مجموعه ای از پاره خطها در یک نقشه رابسازیم .توسیع پاره خطهای ورودی را می توان به عنوان خطوط شکاف انتخاب کردکه به آن BSP خود افراز می گویند.

مثال:

در مجموعهای از چند ضلعی های مسطح در فضای سه بعدی اگر از صفحه سطح چند ضلعی به عنوان صفحات شکاف استفاده کنیم یک BSP خود افراز تولید کرده ایم.

نکته:

اگرچه خود افرازها همیشه درخت BSP مینیمم تولید نمی کنند اما یکی از کوچکترین درختهای BSP هستند.

الگوریتم نقاش:

فرض کنیم درخت BSP بر روی مجموعه S در فضای ۳-بعدی ساخته شده است. یک ترتیب عمقی از اشیا با استفاده از این درخت بدست آمده حال الگوریتم نقاش داریم:

(فرض کنیم نقطه دید از صفحه شکاف که در ریشه درخت ذخیره شده بالاتر است.ترتیب قطعات اشیا در دو زیر درخت چپ و راست بصورت بازگشتی بدست می آید.)

```
Algorithm PAINTERSALGORITHM(T, p_view)
1.  Let v be the root of T.
2.  if v is a leaf
3.    then Scan-convert the object fragments in S(v).
4.    else if p_view ∈ h_v^+
5.           then PAINTERSALGORITHM(T^-, p_view)
6.                Scan-convert the object fragments in S(v).
7.                PAINTERSALGORITHM(T^+, p_view)
8.           else if p_view ∈ h_v^-
9.                  then PAINTERSALGORITHM(T^+, p_view)
10.                      Scan-convert the object fragments in S(v
11.                      PAINTERSALGORITHM(T^-, p_view)
12.                 else (* p_view ∈ h_v *)
13.                      PAINTERSALGORITHM(T^+, p_view)
14.                      PAINTERSALGORITHM(T^-, p_view)
```

الگوریتم نقاش:

نکته: فرض کنیم P_{view} (نقطه دید) روی صفحه شکاف h_v قرار گیرد چند ضلعی نمی تواندر $S(v)$ باشد زیرا چندضلعی دو بعدی مسطح است وبنابراین واضح است که از نقاطی که روی صفحات شامل آنها ست قابل رؤیت نیستند.

نکاتی از الگوریتم:

- الگوریتم نقاش به اندازه درخت BSP وابسته است.
- آن شکافی از نقشه را در نظر می گیریم که کمترین قطعات از اشیا را داشته باشد.
- از سطوح خمیده استفاده نکنیم در عوض چند وجهی مثلثی داریم.

الگوریتم 2DBSP

Algorithm 2DBSP(S)

Input. A set $S = \{s_1, s_2, \ldots, s_n\}$ of segments.

Output. A BSP tree for S.

1. **if** card(S) $\leqslant 1$
2. **then** Create a tree $\mathcal{T}$ consisting of a single leaf node, where the set S is stored explicitly.
3. **return** $\mathcal{T}$
4. **else** (* Use $\ell(s_1)$ as the splitting line. *)
5. $S^+ \leftarrow \{s \cap \ell(s_1)^+ : s \in S\}$; $\mathcal{T}^+ \leftarrow$ 2DBSP(S^+)
6. $S^- \leftarrow \{s \cap \ell(s_1)^- : s \in S\}$; $\mathcal{T}^- \leftarrow$ 2DBSP(S^-)
7. Create a BSP tree $\mathcal{T}$ with root node v, left subtree $\mathcal{T}^-$, right subtree $\mathcal{T}^+$, and with $S(v) = \{s \in S : s \subset \ell(s_1)\}$.
8. **return** $\mathcal{T}$

توجه:

- فرض کنیم S مجموعه n پاره خط غیر متقاطع در یک صفحه باشند ما توجه خود را به خود افراز ها محدود می کنیم.خطی را که از یکی از پاره خط های درون S میگذرد را به عنوان خط شکاف در نظر می گیریم.

بجای انتخاب کور کور انه s_1 می خواهیم پاره خط صحیحی را برای انجام شکافت انتخاب نماییم:

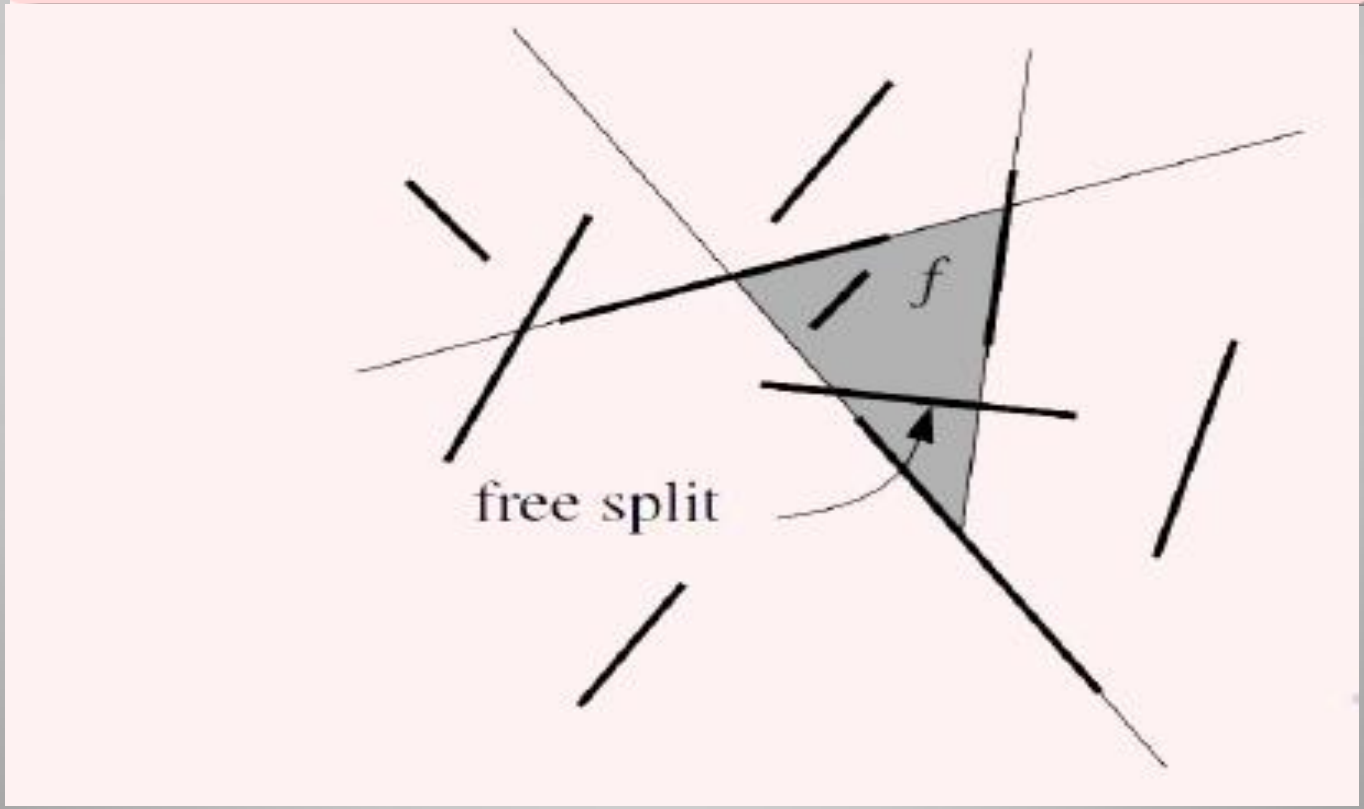
- یک ایده این است که پاره خط s را طوری انتخاب کنیم که l(s) تعدادی پاره خط را ببرد. یک پیکربندی از پاره خطها نشان میدهد که این روش خوب کار نمی کند.بعلاوه پیداکردن این پاره خط کمی وقت گیر است.
- یک ایده ی دیگر انتخاب تصادفی l(s) است:

Algorithm 2DRANDOMBSP(S)
1. Generate a random permutation $S' = s_1, \ldots, s_n$ of the set S.
2. $\mathcal{T} \leftarrow$ 2DBSP(S')
3. **return** $\mathcal{T}$

گاهی اوقات یک بهینه سازی ساده امکان پذیر است:

تعریف پاره خط شکاف آزاد:

یک ساخت بهینه از در خت BSP وجود دارد از زیر بخشهای نقشه یک face مانند f در نظر بگیرید پاره خط شکافت آزاد پاره خطی است که کاملا از f عبور کند و همینطور شکافت f موجب قطعه بندی سایر پاره خطهای درون f نگردد.و گاهی خودش توسط سایر پاره خطها بریده می شود.



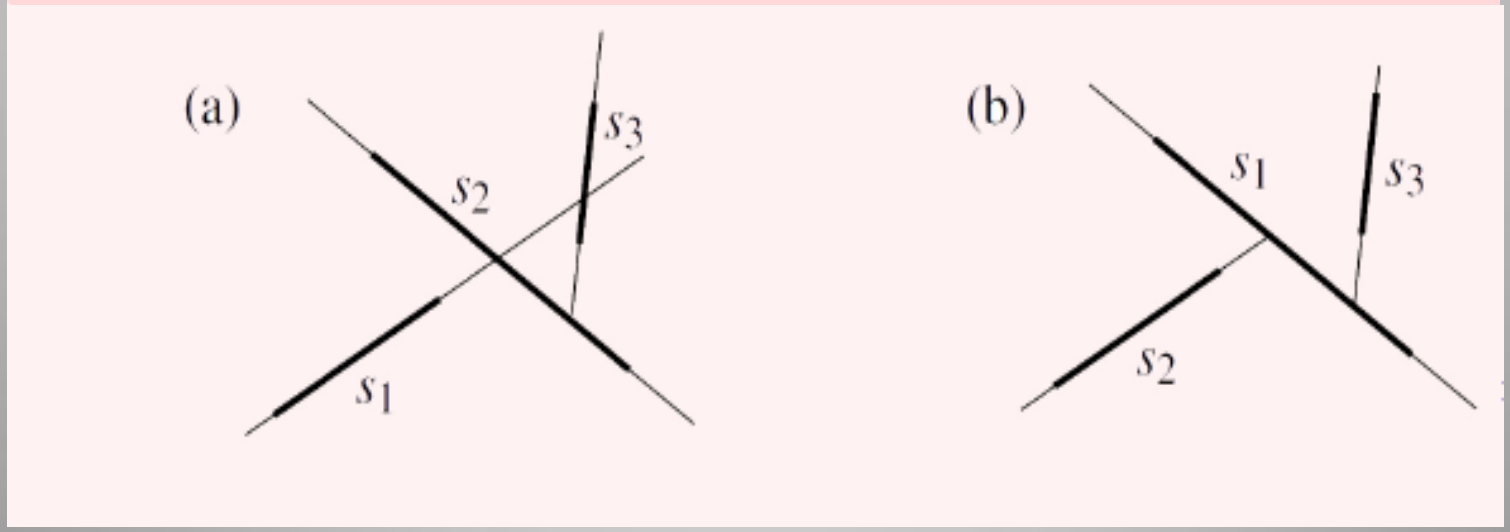
تحلیل الگوریتم 2DRANDOMBS

جایگشت یک درخت BSP

برای شروع اندازه درخت BSP (تعداد قطعات تولیدی)را تحلیل می کنیم که به $n!$جایگشت خطوط شکاف بستگی دارد.برخی از جایگشتها یک درخت BSP کوچک و برخی دیگردرختی بزرگ می دهند.

مثال:

سه پاره خط ترسیم شده در شکل زیر را در نظر بگیرید اگر پاره خطها مانند شکل (a) عمل کنند پنج قطعه ایجاد می گردد و در صورتی که مانند شکل (b) عمل کنند سه قطعه ایجاد می گردد.یعنی با جایگشتهای مختلف قطعات متفاوت ایجاد می گردد.



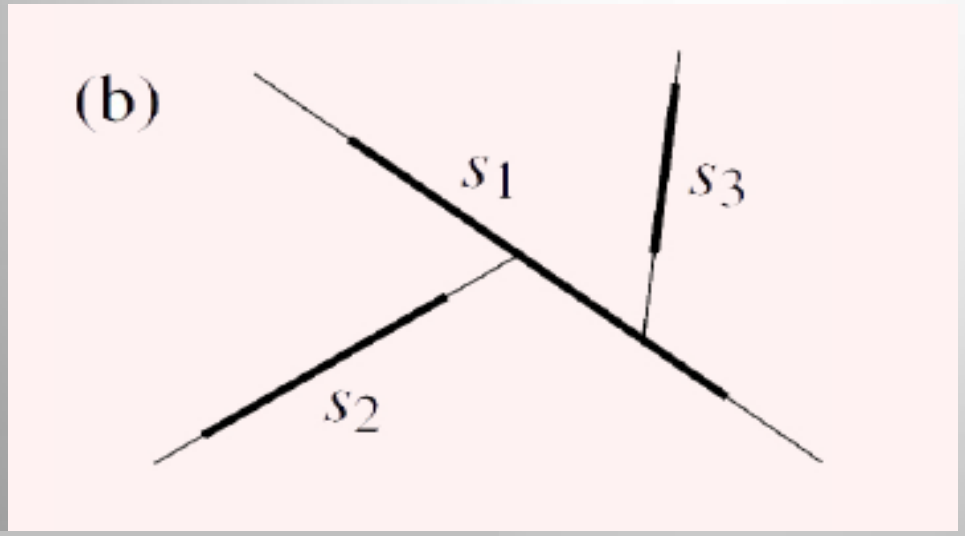
تحلیل الگوریتم 2DRANDOMBS

سپر[مانع] یک پاره خط

وقتی $l(s_i)$ اضافه می شود فرض کنیم s_j را می برد. پاره خطهایی موجودندکه برای s_jسپری هستند از خط s_i اگر $l(s_i)$ قبل از این پاره خطها اضافه شود در این صورت s_j را می برد.

مثال:

اگر پاره خطهای زیر به ترتیب شماره وارد شوند پاره خط s_1 سپری است برای s_3 از خط s_2.

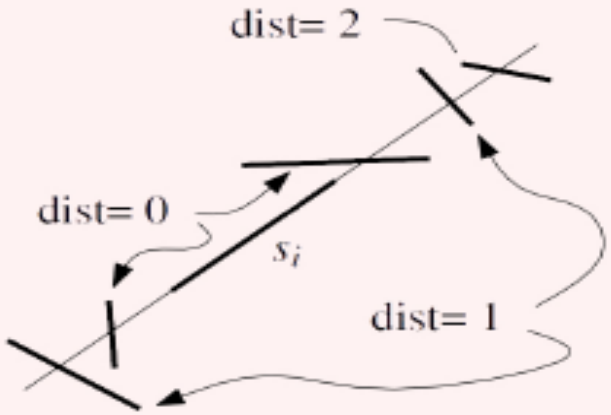


تحلیل الگوریتم 2DRANDOMBS

فاصله دو پاره خط

پاره خط ثابت s_i را در نظر بگیرید اگر $l(s_i)$ پاره خط s_j را قطع کند فاصله بین s_j و s_i تعداد پاره خطهایی است که $l(s_i)$ را قطع می کنند و بین s_j و s_i هستند.

$$\text{dist}_{s_i}(s_j) = \begin{cases} \text{the number of segments intersecting } l(s_i) \text{ in between } s_i \text{ and } s_j & \text{if } l(s_i) \text{ intersects } s_j \\ +\infty & \text{otherwise} \end{cases}$$



تحلیل الگوریتم 2DRANDOMBS

اندازه BSP

لم 12. 1: امید تعداد قطعات تولید شده توسط الگوریتم 2DRANDOMBSP ، $O(n \log n)$ است.

برهان:

فرض کنید s_i یک پاره خط ثابت در S باشد و $l(s_i)$ بعنوان خط شکاف بعدی اضافه گردد.وهمین طور $dist_{s_i}(s_j) = k$ و s_{j_1} و s_{j_2} و...و s_{j_k} پاره خطهای بین s_i و s_j باشند.با چه احتمالی $l(s_i)$ پاره خط s_j را می برد؟

برای این اتفاق بایستی در ترتیب تصادفی s_i قبل از s_j بوده وهمینطور قبل از هر پاره خط دیگری باشد که بین s_i و s_j و سپری برای s_j از s_i است.

اگر $P = \{i, j, j_1, \ldots, j_k\}$ باشد و $Card(P) = t = k + 2$ داریم:

$$pr[\ s_i \text{ before any segments }] = \frac{(t-1)!}{t!} = \frac{1}{t} = \frac{1}{k+2} = \frac{1}{dist_{s_i}(s_j)+2}$$

از آنجا که ممکن است s_r که $r \notin P$ نیز سپری برای s_j از $l(s_i)$ باشد لذا این احتمال خیلی کمتر می گردد و داریم:

$$pr[\ l(s_i) \text{ cuts } s_j\ ] \leqslant \frac{1}{dist_{s_i}(s_j)+2}$$

تحلیل الگوریتم 2DRANDOMBS

ادامه برهان:

$$E[\text{ number of cuts generated by } s_i\ ]$$

$$\leqslant \sum_{j \neq i} \frac{۱}{dist_{s_i}(s_j) + ۲}$$

$$\leqslant ۲ \sum_{k=۰}^{n-۲} \frac{۱}{k+۲}$$

$$\leqslant ۲ \ln n$$

امید تعداد کلی برشهای تولید شده توسط همه پاره خطها حداکثر $۲n \ln n$ می باشد.
چون با n پاره خط شروع می کنیم امید تعداد کلی قطعات به $n + ۲n \ln n$ محدود می شود.

تحلیل الگوریتم 2DRANDOMBS

زمان اجرا

- نشان دادیم امید اندازه BSP که بوسیله2DRANDOMBSP تولید می گردد $n + ۲n \ln n$ است.
- یک BSP در اندازه $n + ۲n \ln n$ برای هر مجموعه از n پاره خط وجود دارد و حداقل نیمی از جایگشتها به ما یک BSP در اندازه $n + ۴n \ln n$ می دهند.
- پس از اجرای الگوریتم ما اندازه را بررسی می کنیم اگر از کران بیشتر باشد الگوریتم را از یک جایگشت تازه بدست می آوریم.

تحلیل زمان اجرا

زمان اجرا نیز به جایگشت تصادفی وابسته است که استفاده می شود بنابر این به زمان اجرای مورد انتظار توجه می کنیم.محاسبه جایگشت تصادفی دارای زمان خطی است .و لذا زمان صرف شده بوسیله الگوریتم 2DBSP خطی وتعداد قطعات در S است . که این تعداد از n بیشتر نیست و با هر بار بازگشت کوچکتر می گردد و در نهایت تعداد بازگشتها به تعداد قطعات تولید شده محدود می شود که $O(n \log n)$ است. لذا زمان ساخت $O(n^۲ \log n)$ است.

تحلیل الگوریتم 2DRANDOMBS

زمان اجرا

Algorithm 2D BSP(S)
Input. A set $S = \{s_1, s_2, \ldots, s_n\}$ of segments.
Output. A BSP tree for S.

1. **if** card(S) ≤ 1
2. **then** Create a tree $\mathcal{T}$ consisting of a single leaf node, where the set S is stored explicitly.
3. **return** $\mathcal{T}$
4. **else** (* Use $\ell(s_1)$ as the splitting line. *)
5. $S^+ \leftarrow \{s \cap \ell(s_1)^+ : s \in S\}$; $\mathcal{T}^+ \leftarrow$ 2DBSP(S^+)
6. $S^- \leftarrow \{s \cap \ell(s_1)^- : s \in S\}$; $\mathcal{T}^- \leftarrow$ 2DBSP(S^-)
7. Create a BSP tree $\mathcal{T}$ with root node v, left subtree $\mathcal{T}^-$, right subtree $\mathcal{T}^+$, and with $S(v) = \{s \in S : s \subset \ell(s_1)\}$.
8. **return** $\mathcal{T}$

the time taken by algorithm 2DBSP is linear in the number of fragments in S : O(n)

قضیه ۱۲. ۲: یک BSP به اندازه $\mathcal{O}(n \log n)$ به امید زمان محاسبه $\mathcal{O}(n^2 \log n)$ است.

تعمیم الگوریتم 2DBSP

تعمیم الگوریتم به فضای سه بعدی

- فرض کنیم s مجموعه nمثلث غیرمتقاطع در فضای سه بعدی باشد.
- فقط خودافراز با سطح افرازهای که شامل مثلثی درs باشد داریم.
- برای یک مثلث t سطح شامل آن h(t) است.

Algorithm 3DBSP(S)
Input. A set $S = \{t_1, t_2, \ldots, t_n\}$ of triangles in $\mathbb{R}^3$.
Output. A BSP tree for S.

1. **if** card$(S) \leqslant 1$
2. **then** Create a tree $\mathcal{T}$ consisting of a single leaf node, where the set S is stored explicitly.
3. **return** $\mathcal{T}$
4. **else** (* Use $h(t_1)$ as the splitting plane. *)
5. $S^+ \leftarrow \{t \cap h(t_1)^+ : t \in S\}$; $\mathcal{T}^+ \leftarrow$ 3DBSP(S^+)
6. $S^- \leftarrow \{t \cap h(t_1)^- : t \in S\}$; $\mathcal{T}^- \leftarrow$ 3DBSP(S^-)
7. Create a BSP tree $\mathcal{T}$ with root node v, left subtree $\mathcal{T}^-$, right subtree $\mathcal{T}^+$, and with $S(v) = \{t \in S : t \subset h(t_1)\}$.
8. **return** $\mathcal{T}$