計算幾何学
Computational Geometry

第二章 線分の交差
Line segment intersection

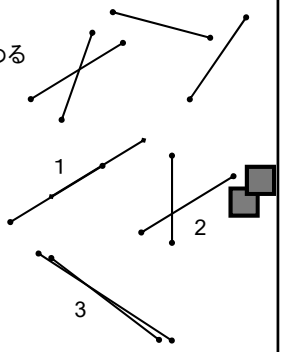
簡単な例

- 連立方程式から交点を求める

$$y = \alpha x + \beta$$

1. 同じ傾き→交点数＝0、∞
2. 垂直の時→勾配＝∞
3. 近い傾き→除算→誤差

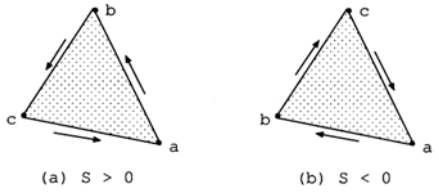
$$\frac{\beta_2 - \beta_1}{\alpha_1 - \alpha_2}$$



三角形の面積

$$S = \frac{1}{2}\left[(a.x - c.x)(b.y - c.y) - (b.x - c.x)(a.y - c.y)\right]$$

右手法則: { (a) 点c→線分abの左、S>0
(b) 線分abの右→点c、S<0



三角形面積の正負と線分交差

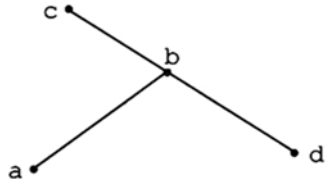
- 3点が同一直線上にある

$S(\Delta abc) > 0$

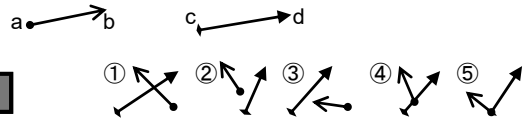
$S(\Delta abd) < 0$

$S(\Delta dca) > 0$

$S(\Delta dcb) = 0$



三角形面積と様々な線分交差

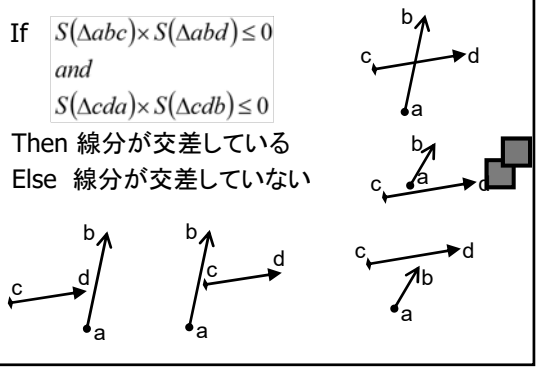


	①	②	③	④	⑤
$S(\Delta abc)$	+	+	+	+	0
$S(\Delta abd)$	−	−	−	−	−
$S(\Delta cda)$	−	+	−	0	0
$S(\Delta cdb)$	+	+	−	+	+
交差?	Y	N	N	Y	Y

三角形面積の正負による交差判定

- If $S(\Delta abc) \times S(\Delta abd) \le 0$ *and* $S(\Delta cda) \times S(\Delta cdb) \le 0$

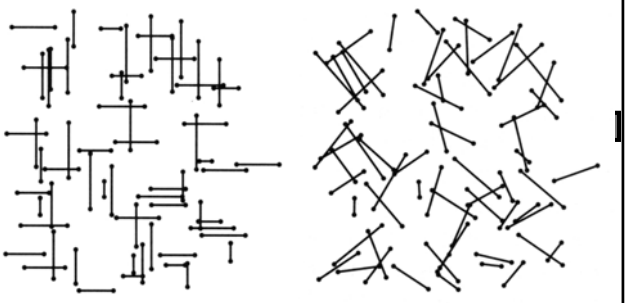
Then 線分が交差している
Else 線分が交差していない



計算量

1. 線分n本
2. 2本ずつ構成された4つの三角形面積を計算し、交差判定を行う
3. 総組み合わせ数＝$n(n\text{-}1)/2 \rightarrow n^2$
4. 総計算量$O(n^2)$
5. もっとスマートな方法がないのか？
6. 直交線分、任意線分

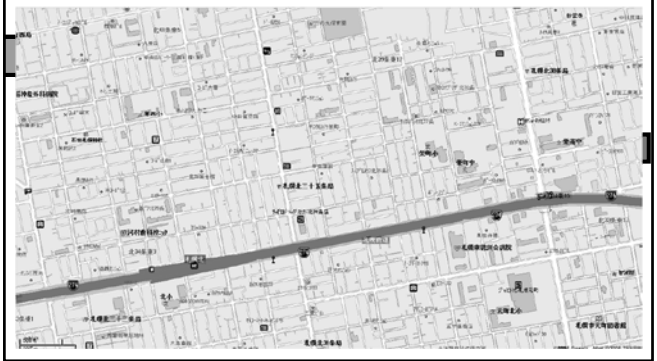
直交線分と任意線分



Manhattan航空写真

Manhattan地図

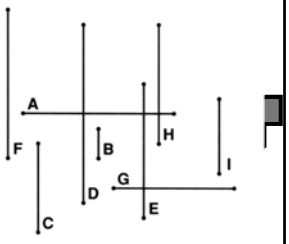
札幌地図



道路→直交線分

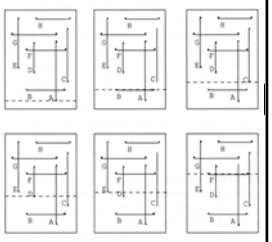
直交線分の交差

- 4個△面積の正負
- 2本ずつの計算
- 組み合せ数＝$n(n\text{-}1)/2$
- 計算量＝$O(n^2)$
- もっと賢い方法？
- マンハッタン幾何学
  (Manhattan geometry)



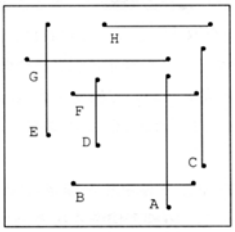
平面走査法(plane sweep)

- 基本考え方
  - 一本の水平走査線を平面上に下から上へ（逆でも同様）移動しながら線分の交差を見出す
- 考察と可能性
  1. 垂直線分と出会う時、垂直線分が走査線上に一点を現す
  2. 垂直線分と離れる時、上記の点が走査線から消える
  3. 水平線分と出会う時、走査線と一瞬だけ重ねる
  4. 水平線分と出会う時、この水平線分の区間に垂直線分の点がある？→yes!（交差！）



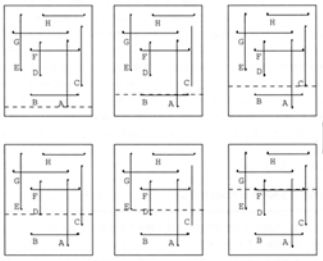
イベント計画(event schedule)

- y座標順で線分の端点を並び替える
  →ABCDEFDAGCEH
  - イベントポイント(event point)
    走査線の停止位置
  - リストで実現する



走査線計画(sweep-line schedule)

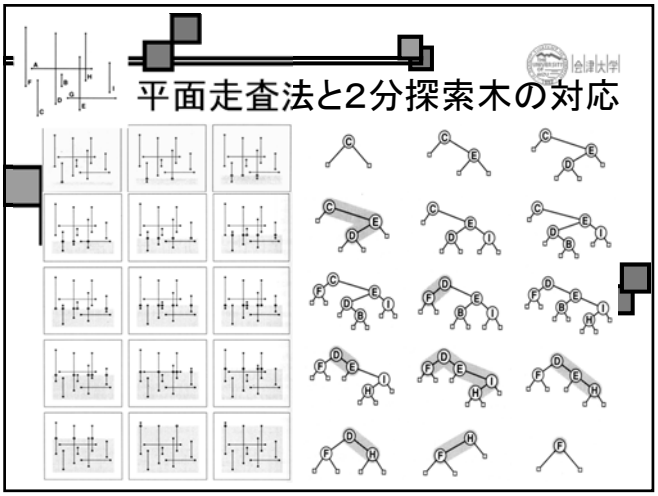
- 垂直線分の下端点
  →x座標を挿入
- 垂直線分の上端点
  →x座標を削除
- 水平線分
  →2端点のx座標
  →区間探索
  - 水平線分と交わる垂直線分を検出
  - 2分探索木で実現



アルゴリズム

1. 線分の端点をy座標にてソート、リストLに入れる
2. ダミー2分探索木Tを生成する
3. リストLの各点に下記の操作を行う
   (a)垂直線分の下端点→その線分をTに挿入
   上端点→その線分をTから削除
   (b)水平線分→左右2端点のx座標を用い、Tに対して区間探索を行う。即ち、垂直線分のx座標は水平線分2端点のx座標区間にあるかどうかを判断する

平面走査法と2分探索木の対応



計算量

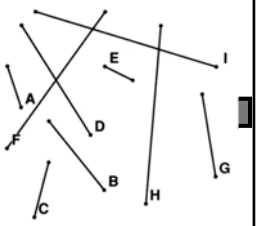
1. 直交線分n本
2. 一つの木操作は$O(\log n)$
3. 垂直線分→挿入・削除→2回→2n回以下の木操作が必要
4. 水平線分→区間探索→交差検出→交差点数k
5. 総計算量$O(n\log n+k)$
6. 格子状配置の場合→$k=n^2/4$

東京西ヶ原周辺航空写真

東京西ヶ原周辺地図

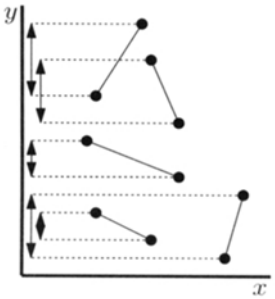
任意線分の交差

- 任意の傾きを有する線分
- 2本ずつ調べるのに$O(n^2)$
- ほんの少数の線分しか交差しない場合？
- 交差点数に依存するアルゴリズム(intersection-sensitive algorithm)?



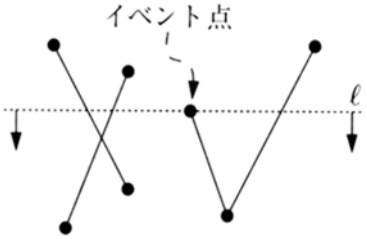
巧妙な方法？

- 考え方
  - すべての線分をy軸へ投影
    - x軸への投影も同様な原理
  - 重ねるもの→可能
    - 交差判定対象にする
  - 重ねないもの→不可能
    - 交差判定対象にせず
  - 平面走査法？
    - 水平走査線とイベント点

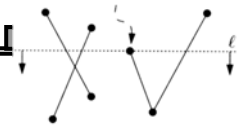


平面走査法

- 水平走査線
  - 上から下へ走査
  - 走査線状態を管理
  - 2分探索木
- イベント点
  - 線分の端点
    - 上端点
    - 下端点
  - 交差点？
  - リスト

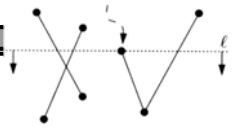


基本的な考え方



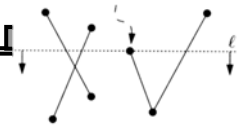
- イベント点→線分上端点→走査線状態に追加
- この新しい加えた線分と既に走査線と交差している複数の線分との間で交差判定を行う
- イベント点→線分下端点→この線分の終点、その後走査線とは交差しなくなる→走査線状態から削除
- 以上により、走査線と交差するような線分対だけに交差判定を行う→***垂直方向の対策***
- 全部の線分対なら、まだ多すぎる→***水平方向の対策？***→水平方向の要素を考えると・・・

基本的な考え方続き



- 走査線と交差する線分を*x*座標に基づいて並び替える
- 水平方向に隣接している時にだけ交差判定を行う（走査線が線分の上端点に会ったら、この新しい線分の左右両側すぐ隣の線分だけに対して交差判定を行う）
- 交差点があれば→線分の順序列変化→走査線状態にも反映させる
- イベントポイント←上端点、下端点、走査途中で求めた交差点（新たな種類のイベント点）
- 上記のイベント→近傍関係変化→交差判定

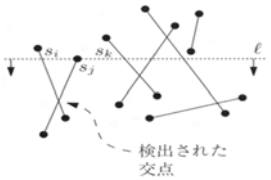
まとめ



1. イベント点＝前もって分った線分の上、下端点、走査途中で求めた交差点
2. 走査線と交差する複数の線分を順序に管理する
3. 走査線はイベント点ごとに止まって、線分の順序を更新し、交差を検出し、イベント点を追加・削除する

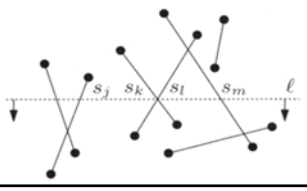
イベント点＝線分の上端点

1. 走査線と交差する新たな線分が存在
2. この新たな線分の左右近傍の2線分を求め、新線分と左右近傍2線分の交差判定を行う（走査線より下にある交差点だけで良い）
3. 走査線の下に交差点があれば、新たなイベント点とする



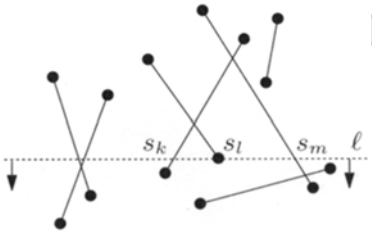
イベント点＝線分の交差点

1. 2本の線分は交差すると→順序関係が変化
2. 2本の線分はそれぞれ新たな近傍を得る
3. 新近傍の線分と交差判定をそれぞれ行う（走査線より下の交差点だけ）
4. 新たな交差点があれば→新たなイベント点



イベント点＝線分の下端点

1. この端点の左右近傍の2線分の交差判定を行う
2. 交差点は走査線より下にある→新たなイベント点とする

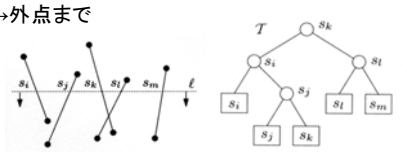


イベント点のデータ構造

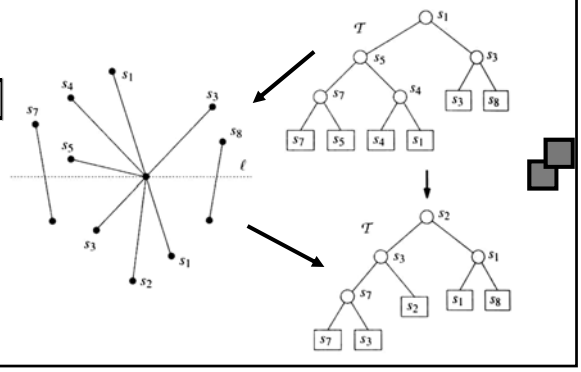
1. イベント点を蓄える（イベントキューQ）
2. 取出・追加操作
3. 同一y座標の2つイベント点は左側の点が優先する
4. 水平線分→上端点（左端点）、下端点（右端点）
5. 2分探索木（$p_y>q_y$又は$p_y=q_y$且つ$p_x<q_x$順）

走査線状態のデータ構造

1. 走査線と交わる線分の順序列を管理する
2. 線分と走査線を交差し始めたら→追加操作
3. 線分と走査線を交差しなくなったら→削除操作
4. 2分探索木(T)の外点に走査線と交差する線分を左から右への順に蓄える
5. 走査線上にある点のすぐ左・右側に位置する線分を探索する場合→内点と比較→比較結果に応じて左右の部分木に進む→外点まで

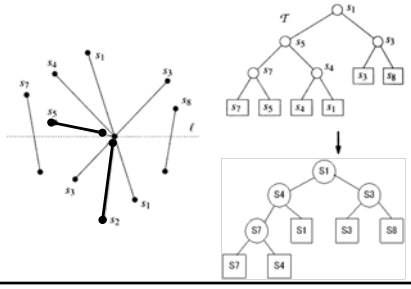


イベント点と走査線状態の変化



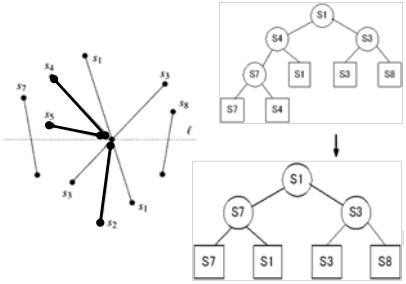
S_5の処理

- 下端点
  →走査線状態木TからS_5を削除→近傍関係変化



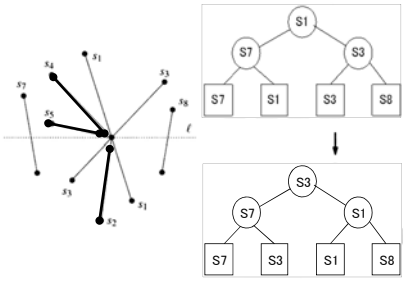
S_4の処理

- 下端点
  →走査線状態木TからS_4を削除→近傍関係変化



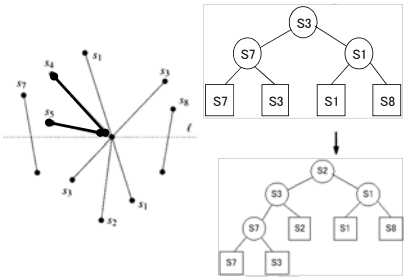
S_1とS_3の処理

- 交差点
  →走査線状態木Tの近傍関係変化→(S_1S_3→S_3S_1)



S_2の処理

- 上端点
  →走査線状態木TにS_2を挿入→近傍関係変化



アルゴリズム

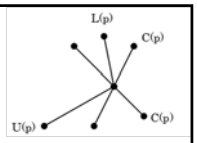
- FindIntersections(S)
  - HandleEventPoint(p)
    - FindNewEvent(s_l, s_r, p)

FindIntersections(S)

- 入力＝線分集合S
- 出力＝Sにあるすべての線分間の交点の集合、及び、各交点に対応する線分の集合

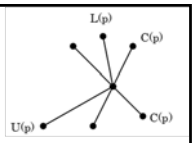
1. イベントキューQと走査線状態Tを空に初期化
2. 線分の上下端点をy座標の順でQを生成する
3. while Qが空でない
4. do Qの次のイベント点pを求め、Qから取出
5. HandleEventPoint(p)

HandleEventPoint(p)



1. $U(p)$、$L(p)$→pをそれぞれ上、下端点とする線分の集合
2. $C(p)$→pを含むすべての線分の集合（上下端点以外）
3. if $L(p)+U(p)+C(p)>=2$
4. then pを交差点として出力、$L(p)$, $U(p)$, $C(p)$も交差がある線分として出力
5. $L(p)+C(p)$の線分をTから削除
6. $U(p)+C(p)$の線分をTに挿入（線分の順序はpのすぐ下の走査線と交差順序に対応）
7. ($C(p)$の線分を削除して再び挿入すると、順序が逆転)

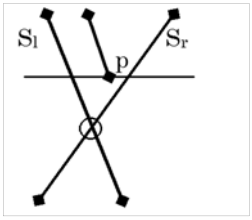
HandleEventPoint(p)続き



8. if $U(p)+C(p)=0$ （下端点の場合）
   then
   Tにおいてpの左隣の線分s_lと右隣の線分s_r
9. FindNewEvent(s_l, s_r, p)
   else （上端点又は交差点の場合）
   Tにおいて$U(p)+C(p)$の左端の線分＝s_L、
   s_Lの左隣の線分s_l
10. FindNewEvent(s_l, s_L, p)
    Tにおいて$U(p)+C(p)$の右端の線分＝s_R、
    s_Rの右隣の線分s_r
11. FindNewEvent(s_R, s_r, p)

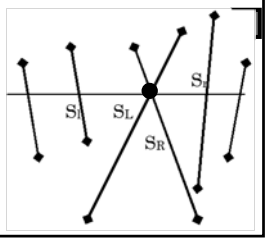
下端点の場合

1. 下端点pの左隣にある線分Slと右隣にある線分Srの交差判定を行う
2. 交差する場合→新しいイベントとしてイベントキューに追加する



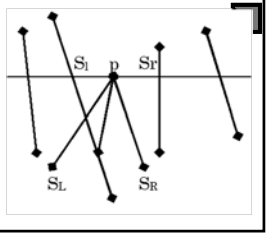
交差点の場合

1. $C(p)$の左端の線分s_Lと s_Lの左隣の線分s_lの交差判定を行う
2. $C(p)$の右端の線分s_Rとs_Rの右隣の線分s_rの交差判定を行う
3. 交差する場合→
   - 新しいイベントとしてイベントキューに追加すべきかどうかを定める



上端点の場合

1. $U(p)$の左端の線分s_Lと s_Lの左隣の線分s_lの交差判定を行う
2. $U(p)$の右端の線分s_Rとs_Rの右隣の線分s_rの交差判定を行う
3. 交差する場合→
   - 新しいイベントとしてイベントキューに追加すべきかどうかを定める

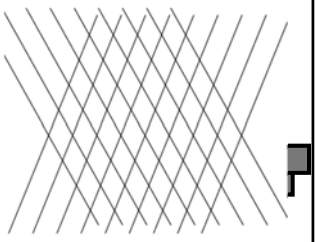


FindNewEvent(s_l, s_r, p)

1. if
   - ①s_lとs_rが走査線より下で交差する、又は、
   - ②走査線上で p の右で交差する、且つ、Qにこの交差点はまだ入っていない
2. then
   - この交差点をイベント点としてQに挿入

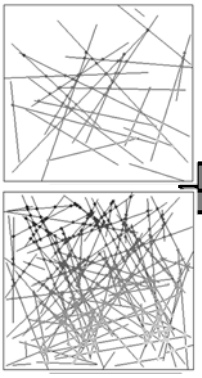
計算量

- 線分n本、交差点k個
- 交差判定 $O(1)$
  - n上端点 2回
  - n下端点 1回
  - k交差点 2回
- 線分リストの操作 $O(\log n)$
  - 線分挿入 n回
  - 線分削除 n回
  - 線分交換 k回
- イベントリストの操作 $O(\log(n+k))$
  - イベントの追加と削除 $2n+k$回
- 総計算量$O((2n+k)\log(n+k))\rightarrow O((n+k)\log(n))$←交点数$k$に依存



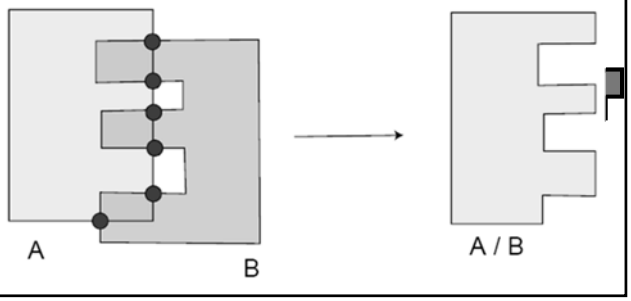
応用

- CAD (Computer Aided Design)
  - 機械部品の自動設計
- ロボット自動走行
  - 障害物との衝突判定
- パターン認識
  - 線形分離と凸多角形の交差
- VLSI回路設計
  - 部品配置と配線交差のチェック
- コンピュータグラフィックス
  - 隠面・隠線除去



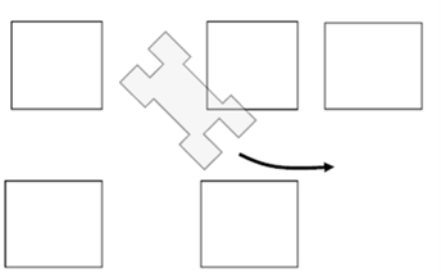
CAD

- 機械部品の自動設計



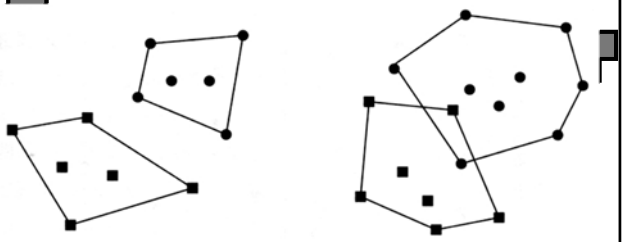
ロボット自動走行

- 障害物との衝突判定



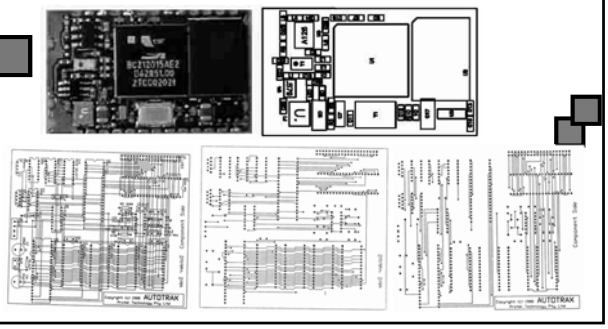
パターン認識

- 線形分離と凸多角形の交差
  - パターン分類が可能かどうかを判断する



VLSI回路設計

- 部品配置、配線チェック



コンピュータグラフィックス

- 隠面・隠線処理
  - どの部分が見えるか、どの部分が隠れるかを判断する