コンピュータグラフィックス

8. レンダリング1

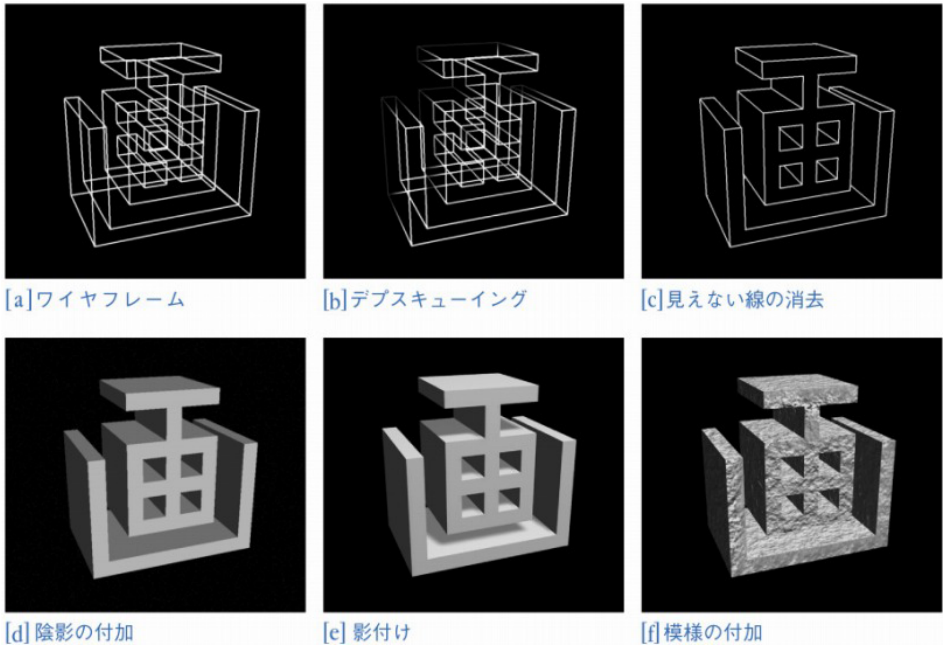
ー 陰面消去 ー

佐藤証 西9-613
akashi.satoh@uec.ac.jp

教科書P.96-116

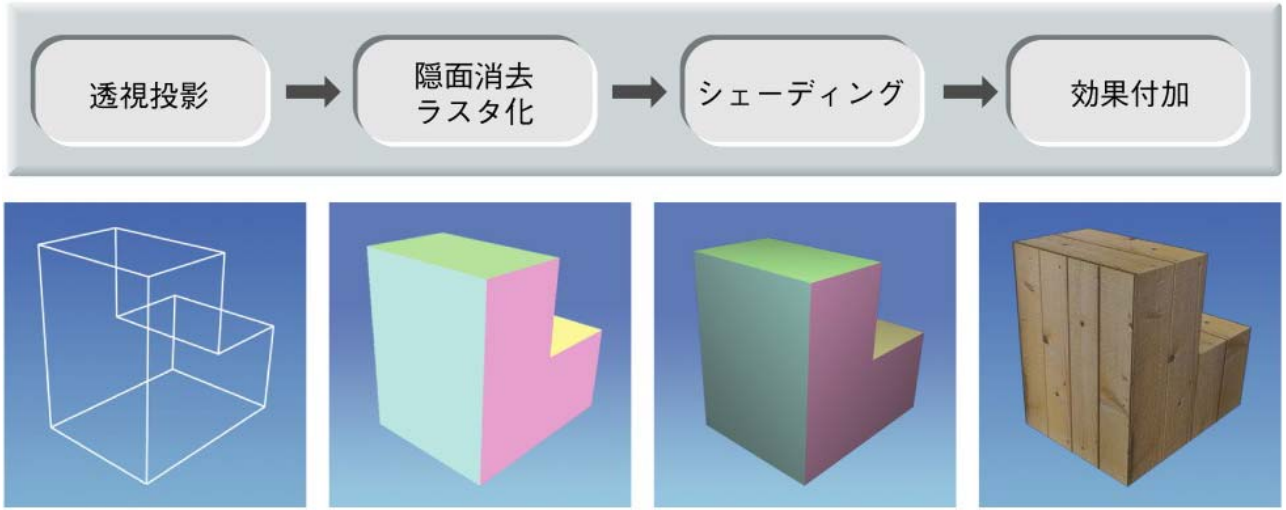
実写的表現法

- 物体の材質感を表現するには，物体表面での光の反射による映り込みや拡散，透過・屈折等の処理が必要となる
- ワイヤーフレーム表示は高速な反面リアリティに乏しく形状の把握が困難なため次のレンダリング処理を行う
  - デプスキューイング表示：遠方の線の輝度を落として奥行を表現
  - 陰線消去：手前の面で見えない線の一部を消去
  - 陰面消去：手前の面で見えない面の一部を消去
  - シェーディング：物体に陰影を付ける
  - 影付：物体に影を付ける
  - テクスチャマッピング：物体表面に模様を張り付ける



実写的表現法

- ３次元レンダリングは以下のような処理で構成される
  - 透視投影：3次元空間で定義された形状モデルを2次元の投影面(スクリーン)に投影
  - 陰面消去・ラスタ化：手前の面で隠された線や面の一部を消去し，その過程で線分によりベクタ表現されていた図形が画素の集まりのラスタ表現に変換
  - シェーディング：光の物理的な性質に基づいて表示対象の輝度(画素のR,G,B値)を計算
  - 効果付加：テクスチャマッピング等様々な効果を付加

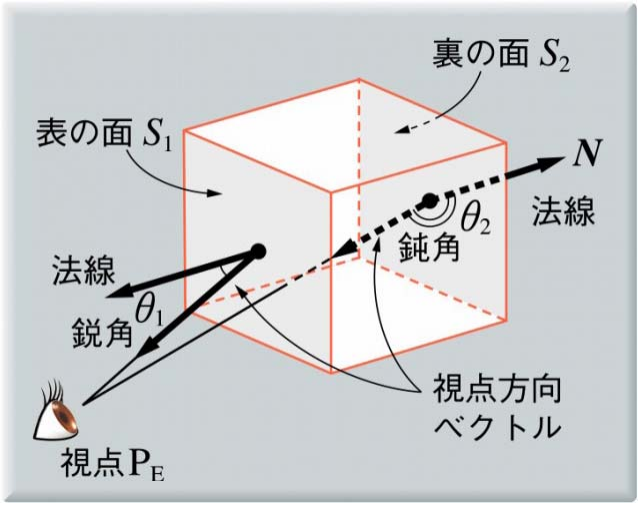


バッファフェースカリング

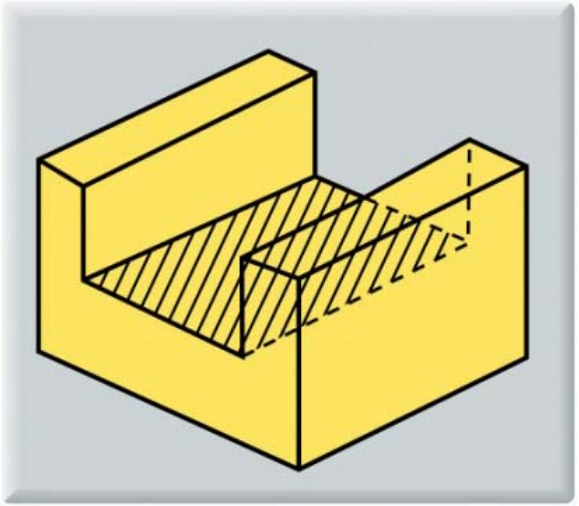
- 視点P_Eから見えない面をあらかじめ除去することをバッファフェースカリングと呼ぶ
- 面の法線ベクトルと視線とのなす角が鈍角のものを除去
  - 法線ベクトルを$\boldsymbol{V}$，視線ベクトルを$\boldsymbol{N}$とする内積$D_f>0$のときは表面，$D_f<0$のときは裏面，$D_f=0$のときは視線に平行な面
  - 面を構成する任意の頂点P_iを用いて表すこともできる

$$D_f = \boldsymbol{V}\cdot\boldsymbol{N} = (\boldsymbol{P_E} - \boldsymbol{P_i})\cdot\boldsymbol{Nc}$$

- 複数の物体がある場合や凹形状の物体に対してはバッファフェースカリングだけでは除去できない面がある



バッファフェースカリング



多面体を構成する面の可視性

法線ベクトルの算出

- 同一直線状にないポリゴンの3頂点P_i,P_j,P_kの3次元座標値をそれぞれ，(x_i, y_i, z_i), (x_j, y_j, z_j), (x_k, y_k, z_k) $(i < j < k)$とし，頂点番号は反時計回りに付けられているとき，法線ベクトル$\boldsymbol{N}$ (n_x, n_y, n_i)は3頂点の外積で求められる

$$\boldsymbol{N} = \boldsymbol{V} \cdot \boldsymbol{N} = (\boldsymbol{P_i} - \boldsymbol{P_j}) \times (\boldsymbol{P_j} - \boldsymbol{P_k})$$

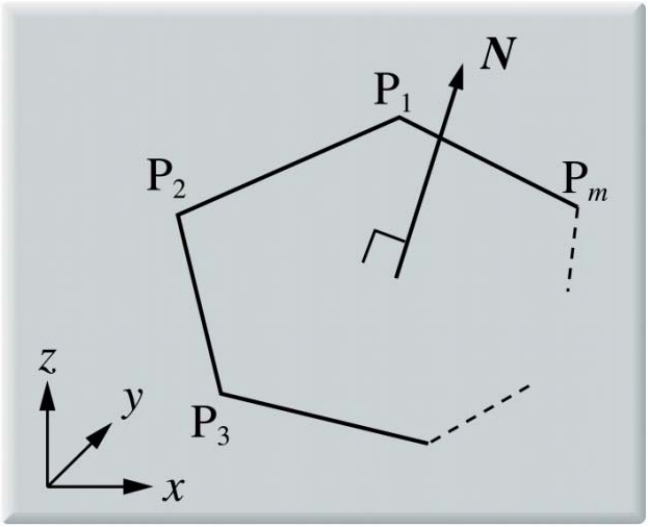
$$n_x = (y_i - y_j)(z_j - z_k) - (z_i - z_j)(y_j - y_k)$$

$$n_y = (z_i - z_j)(x_j - x_k) - (x_i - x_j)(z_j - z_k)$$

$$n_z = (x_i - x_j)(y_j - y_k) - (y_i - y_j)(x_j - x_k)$$

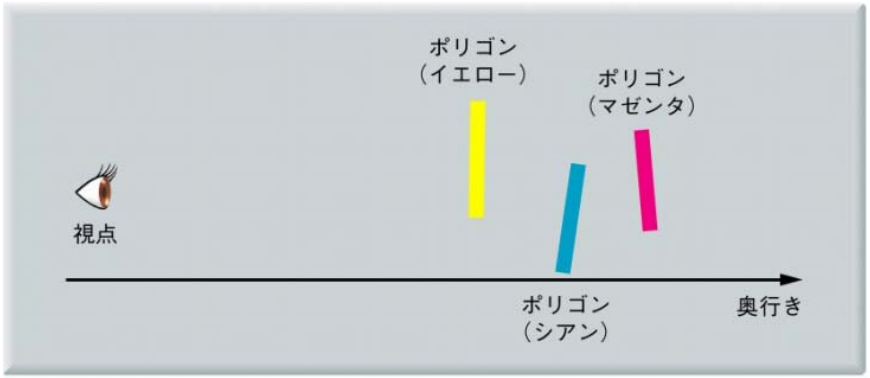
- 単位法線ベクトル

$$\widehat{\boldsymbol{N}} = \frac{\boldsymbol{N}}{\|\boldsymbol{N}\|} = \frac{\boldsymbol{N}}{\sqrt{n_x^2 + n_y^2 + n_z^2}}$$



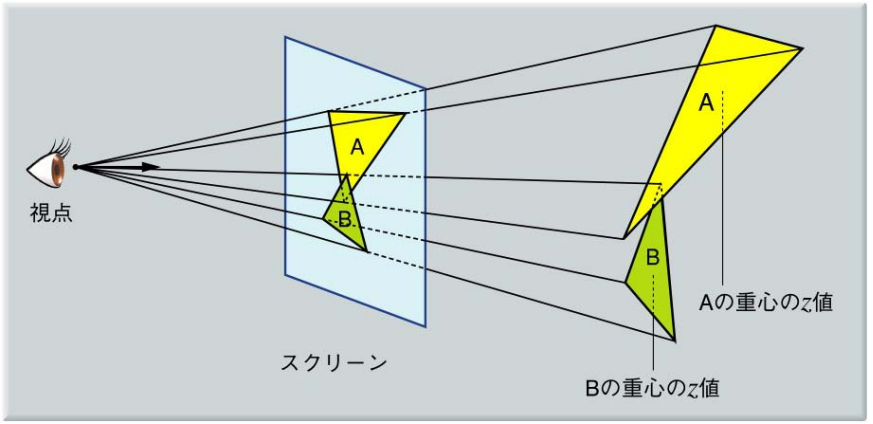
優先順位アルゴリズム

- 画素の色を格納するメモリのフレームバッファに対して奥にある面を先に，手前にある面を先に描くことで陰面消去を行う
- 優先順位を視点とポリゴンの代表点(重心位置等)との距離で決める単純な方法では陰面消去に失敗する場合がある.



[a] ポリゴンの配置

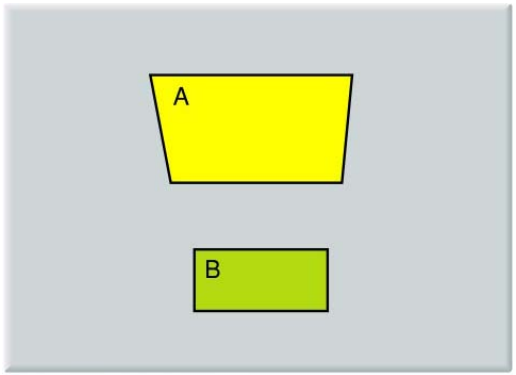
奥行による優先順位のソート



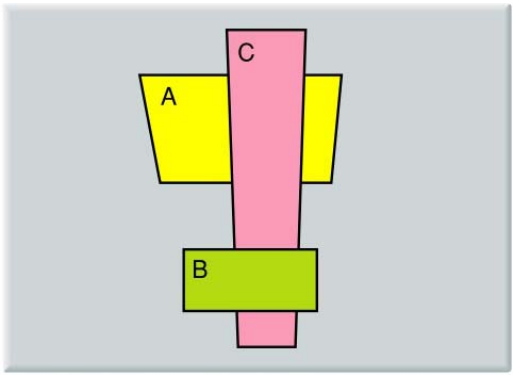
重心によるソートで陰面消去に失敗した例

優先順位アルゴリズム

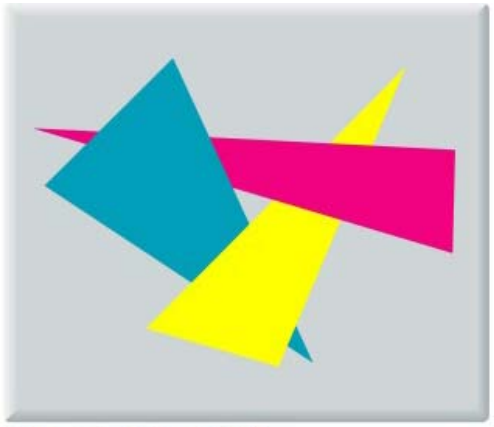
- 2つのポリゴンでも順序を決められない場合が多々ある
  - ポリゴンを分割することで解決する場合もある
- 優先順位の決定に必要な情報を視点が与えられる前に計算して奥方法も開発されている



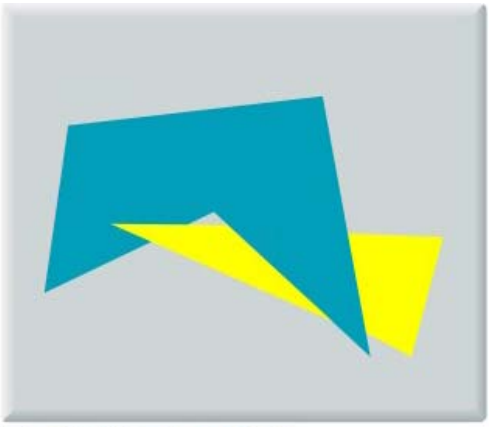
[a] ポリゴンが2枚の場合



[b] ポリゴンが3枚に増えた場合

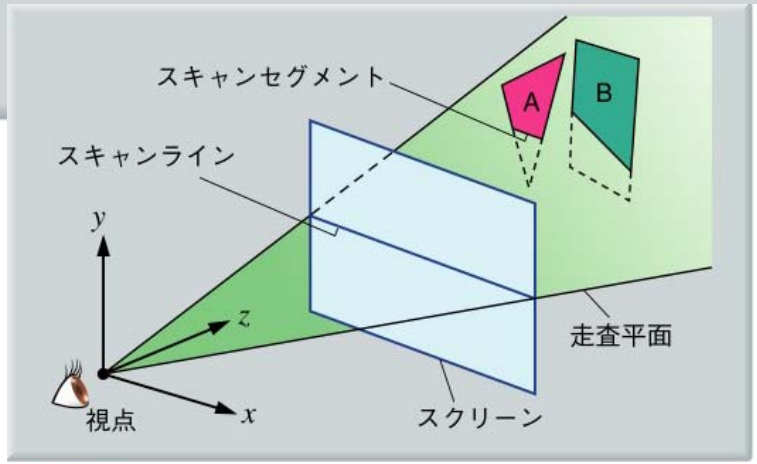

[a] 三すくみの場合

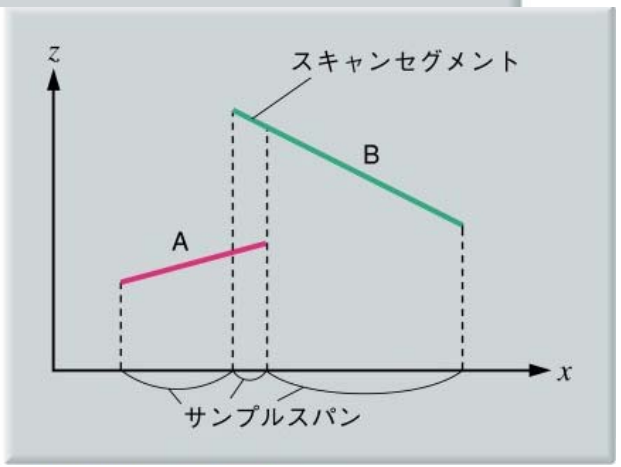

[b] 貫通している場合


[c] 凹ポリゴンの場合

スキャンライン法

```
すべてのポリゴンを透視投影する;
スキャンラインを上から下へ順番に移動させ,
それぞれのスキャンラインについて{
    走査平面とポリゴンとの交差線分(スキャンセグメント)を求める;
    各スキャンセグメントの端点をx座標の小さい順にソートする(x軸ソート);
    スキャンセグメントの端点によりサンプルスパンを決定する;
    それぞれのサンプルスパンについて{
        奥行きの最も小さいスキャンセグメント(可視セグメント)を求める;
        そのサンプルスパンに可視セグメントの色を塗る;
    }
}
```





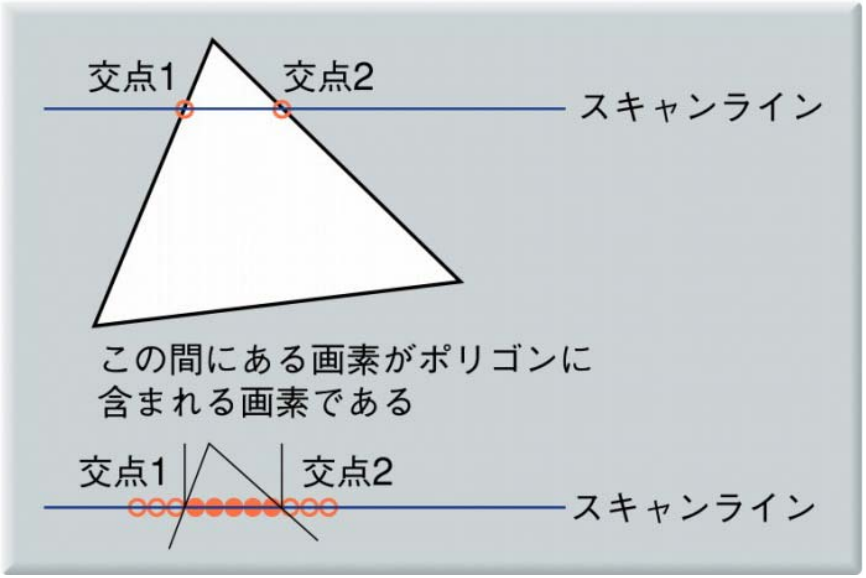
- 作業メモリはスキャンライン一本分で済むが，光の反射や映り込み等の表現ができない

ポリゴンの操作変換

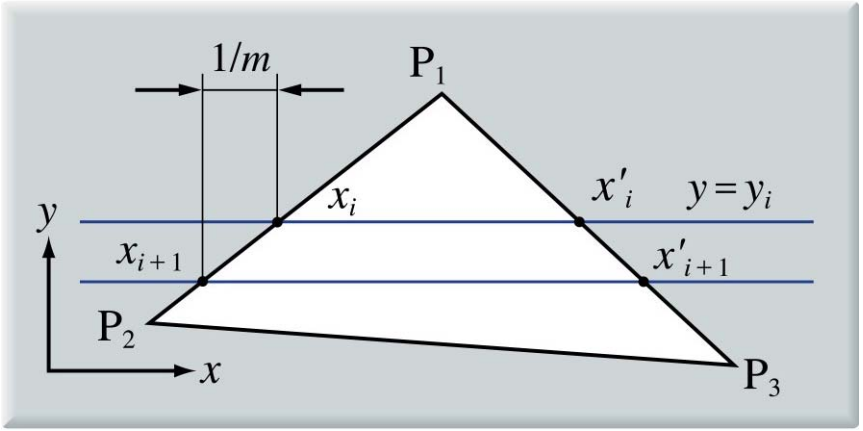
- ポリゴンの辺とスキャンラインとの2つの交点の内側がポリゴン内部の画素であり，これを全てのスキャンライン塗りつぶすことでラスタ化できる
- ラインを上から順に走査するとき，交点x_iの次のラインとの交点x_{i+1}は次式の増分法で求められる

$$x_{i+1} = x_i + \Delta x = x_i - \frac{1}{m}$$
$$(y = mx)$$

- 辺の端点のy座標が大きい順にソートしておくことで高速化が可能



三角形の操作変換



増分法による交点の算出

Zバッファ法

- フレームバッファと同じ解像度で，画素ごとに奥行き(z値)を格納するZバッファを用意して，交点のz値を比較
- ポリゴン以外の曲面にも適用できる
- アルゴリズムが簡単でハードウェア化しやすいためGPU(Graphics Processing Unit)で採用されている

```
フレームバッファ F(i, j)のすべての画素を背景色で初期化する;
Zバッファ Z(i, j)のすべての画素を最遠点(無限大)で初期化する;
すべてのポリゴンを任意の順番でとり出し,
それぞれのポリゴンについて {
    ポリゴンを透視投影する;
    ポリゴンを走査変換する;
    ポリゴン内部の各画素(i, j)について {
        画素(i, j)に対応する点でのポリゴンの奥行き(z値)を求める;
        ポリゴンのz値とZ(i, j)を比較して,
        z < Z(i, j)を満たすならば {
            画素(i, j)に対応する点でのポリゴンの色をF(i, j)に格納する;
            Z(i, j)をポリゴンのz値で更新する;
        }
    }
}
```

z値の計算

ポリゴンの法線ベクトルを$\boldsymbol{N}\,(n_x, n_y, n_i)$，いずれかの頂点を$\mathrm{P}_i\,(x_i, y_i, z_i)$とすると，ポリゴンを含む平面上の点$\mathrm{P}\,(x, y, z)$は

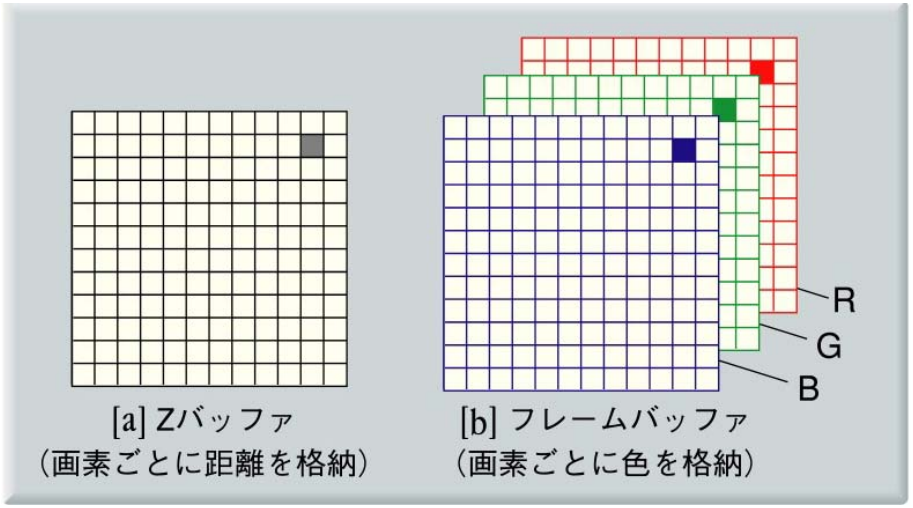
$$(\boldsymbol{P} - \boldsymbol{P_i}) \cdot \boldsymbol{N} = 0$$

各座標成分での表記は

$$n_x x + n_y y + n_z z + d = 0$$

$$d = -\boldsymbol{P_i} \cdot \boldsymbol{N}$$
$$= -(n_x x_i + n_y y_i + n_z z_i)$$

$$\therefore z = -\frac{n_x x + n_y y + n_z z + d}{n_z}$$



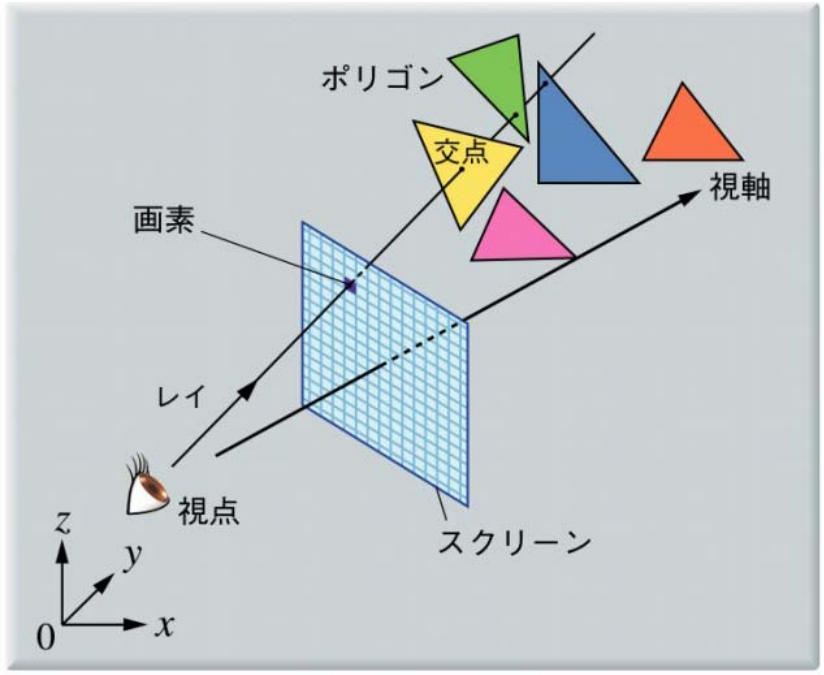
Zバッファとフレームバッファ

左の画素でのｚ値に奥行きの増分　$z = -n_x/n_z$
を加算する増分法でｚ値を高速に求まる

Zバッファ法の描画過程

レイトレーシング法

- ワールド座標系で視点からスクリーンの画素に向うレイと物体の交点を計算
- レイを交差した物体の反射や透過･屈折方向に分けて処理を繰り返す




```
スクリーンのすべての画素を順番にとり出し,
それぞれの画素について{
    視点から画素に向かうレイを決定する;
    レイとすべての物体との交差判定を行う;
    レイと交差する物体が1つ以上存在するならば{
        すべての交点のなかで最も視点に近いもの(可視点)を求める;
        その画素に可視点での物体の色を塗る;
    }
    そのほか(レイと交差する物体が存在しない)の場合{
        その画素に背景色を塗る;
    }
}
```

レイと球との交差判定

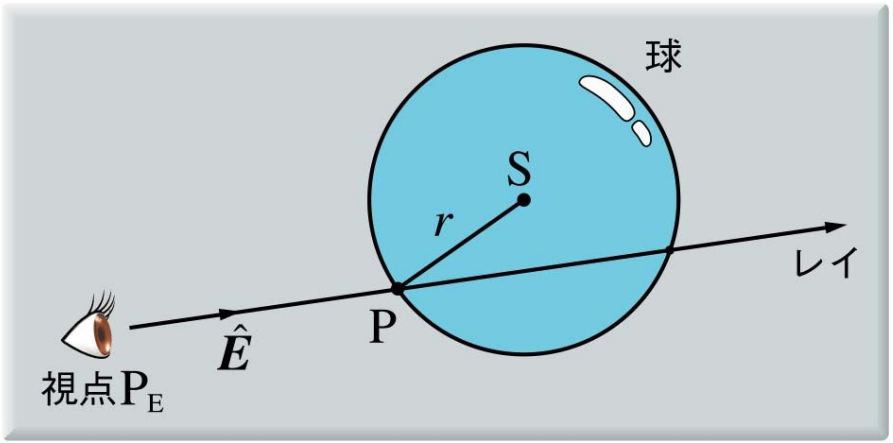
レイの単位方向ベクトルを$\hat{\boldsymbol{E}}$，球の中心を$\mathrm{S}(x_\mathrm{S}, y_\mathrm{S}, z_\mathrm{S})$，半径を$r$，
レイ上の点を$\mathrm{P}(x, y, z)$するとき，視点からの距離のパラメータt
を用いて視点を通るレイの方程式と球の方程式は次式となる

$$\text{レイ}：\boldsymbol{P} = \hat{\boldsymbol{E}}t + \boldsymbol{P}_\mathrm{E}$$

$$\text{球}：\|\boldsymbol{P} - \boldsymbol{S}\| = r$$

球の式の両辺を自乗して内積で表すと

$$(\boldsymbol{P} - \boldsymbol{S}) \cdot (\boldsymbol{P} - \boldsymbol{S}) - r^2 = 0$$



レイの式に代入して整理

$$ar^2 + 2bt + c = 0$$

$$a = \left\|\hat{\boldsymbol{E}}\right\|^2 = 1$$

$$b = \hat{\boldsymbol{E}} \cdot (\boldsymbol{P}_\mathrm{E} - \boldsymbol{S})$$

$$c = \|\boldsymbol{P}_\mathrm{E} - \boldsymbol{S}\|^2 - r^2$$

判別式$D = b^2 - ac > 0$でtが正の解を持てば交点が存在
交点Pでの球面の単位法線ベクトルは

$$\hat{N} = \frac{\boldsymbol{P} - \boldsymbol{S}}{r}$$

レイとポリゴンの交差判定

- ポリゴンを含む平面とレイとの交点のパラメータtは次式となる

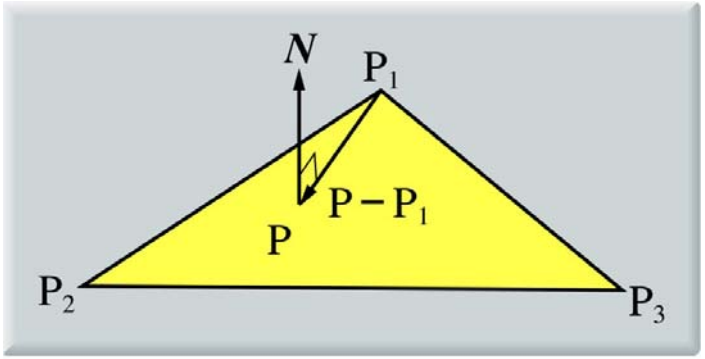
$$t = \frac{(\boldsymbol{P}_{\mathrm{E}} - \boldsymbol{P}_i) \cdot \boldsymbol{N}}{\hat{\boldsymbol{E}} \cdot \boldsymbol{N}}$$

$t > 0$：視点の前方に交点がある

$t < 0$：視点の後方に交点がある

$t = 0$：視点が平面上に存在

$\hat{\boldsymbol{E}} \cdot \boldsymbol{N} = 0$：レイと平面が平行(交点なし)

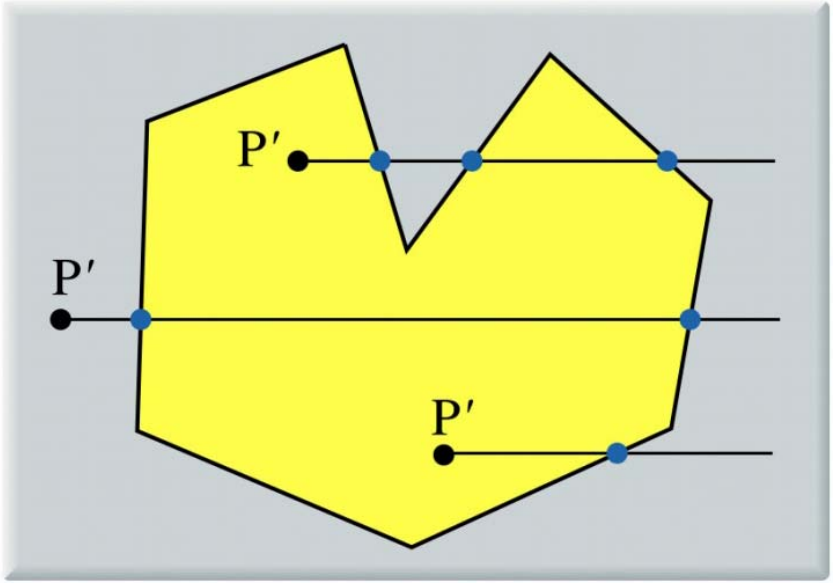


三角形を含む平面での交点算出

- 次に求めた交点Pがポリゴン内部に存在するかどうか判定する
- 一般に3次元空間で判定を行うより，2次元平面に射影した後で判定を行う方が計算量や計算誤差の点で有利

半直線との交点数を用いた点の内外判定

- 射影後の交点をP′，射影後のポリゴンの頂点をP'_1, P'_2, P'_3とするとき，点P′から任意の方向に引いた半直線とポリゴンとの交点数を数える
  - 交点数が奇数：P′ はポリゴン内部
  - 交点数が偶数：P′ はポリゴン外部
- 半直線が頂点を通過する場合に交点とするかどうか判断が必要

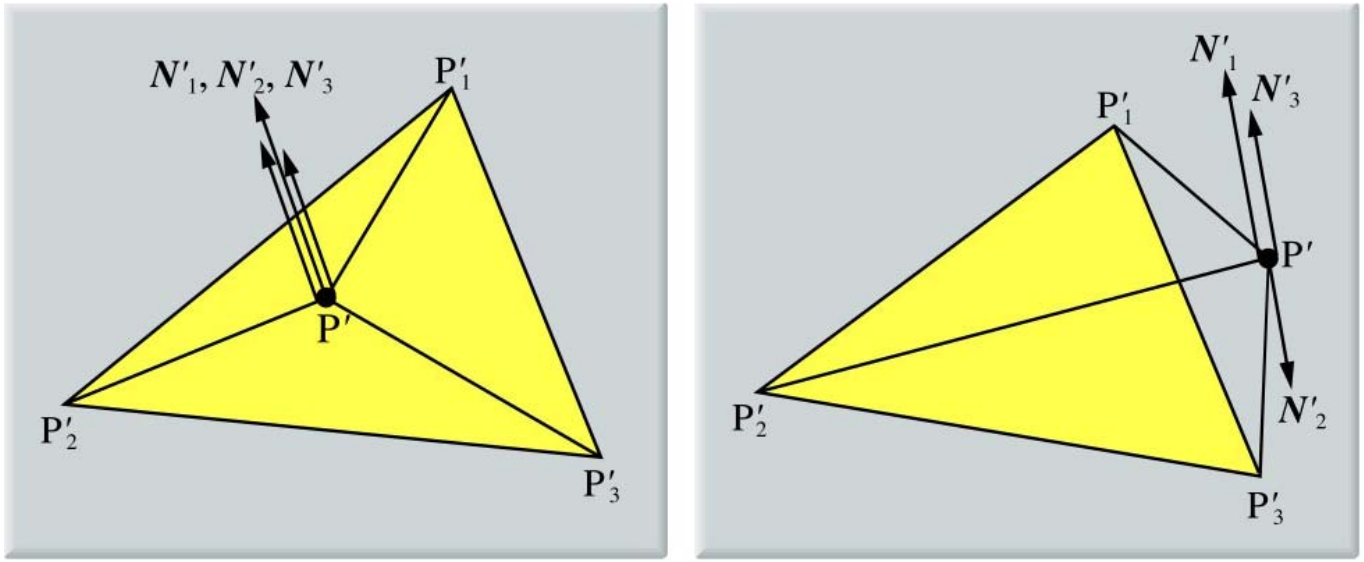


外積の方向を用いた点の内外判定

- 頂点P′とポリゴンの各頂点を結ぶベクトルを用いて次式の外積を求める

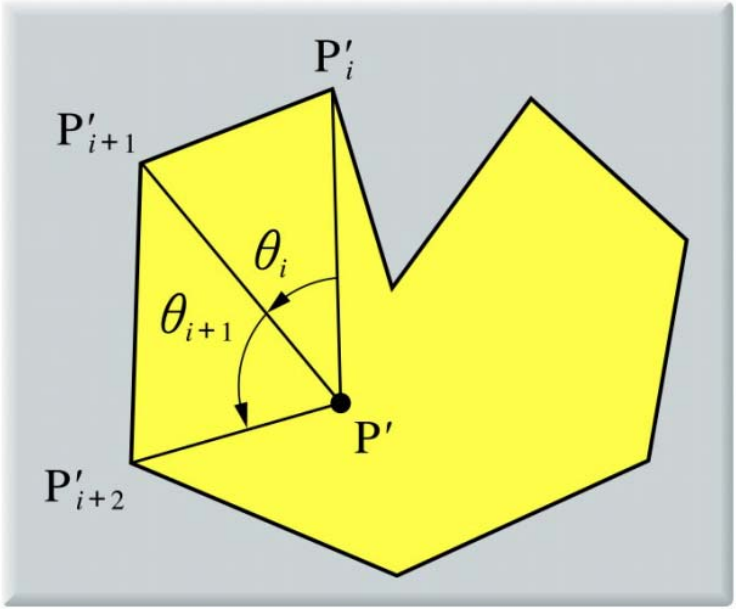
$$\boldsymbol{N}'_{\boldsymbol{i}} = (\boldsymbol{P}'_{\boldsymbol{i+1}} - \boldsymbol{P}') \times (\boldsymbol{P}'_{\boldsymbol{i+2}} - \boldsymbol{P}')$$

  - $\boldsymbol{N}'_{\boldsymbol{i}}$が全て同じ方向：P′ はポリゴン内部
  - $\boldsymbol{N}'_{\boldsymbol{i}}$が一つでも異なる方向：P′ はポリゴン外部
- 判定は凸型のポリゴンに限定される



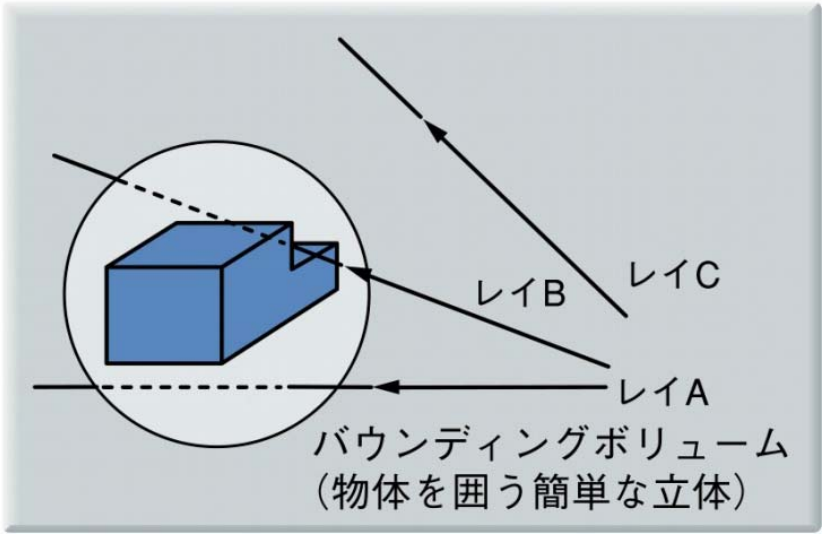
角度の和を用いた内外判定

- 頂点P'とポリゴンの隣接する各頂点とを結ぶベクトルのなす角度の総和により判定を行う
  - 総和が2π：P' はポリゴン内部
  - 総和が0：P' はポリゴン外部
- 計算量が多いのが欠点

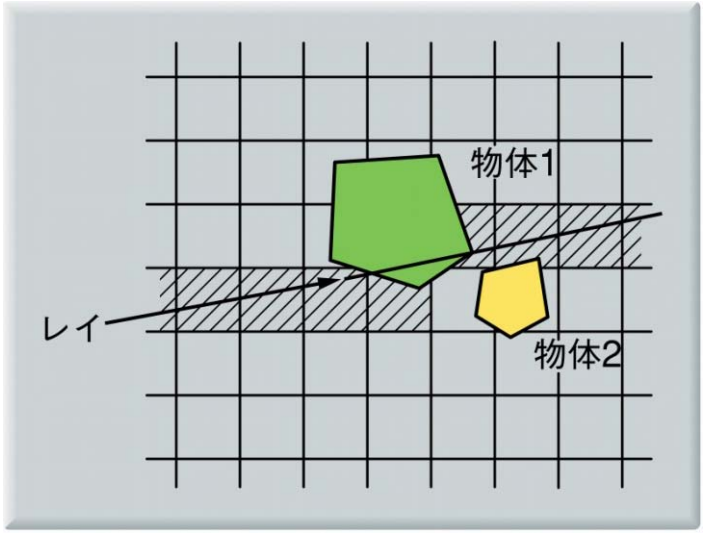


レイトレーシング法の高速化

- 物体を球や直方体等の簡単なバウンディングボリュームで囲いまずレイとバウンディングボリュームとの交点計算を行う
  - バウンディングボリュームを階層化してさらなる高速化が可能
- 空間分割法は3次元空間を立方格子状のボクセルに分割し，ボクセルに存在する物体を記憶して，レイが通過するボクセルの中の物体とだけ交点計算を行う
- 各画素の処理が独立しているので複数のCPUで並列化する



バウンディングボリュームによる高速化



空間分割法による高速化