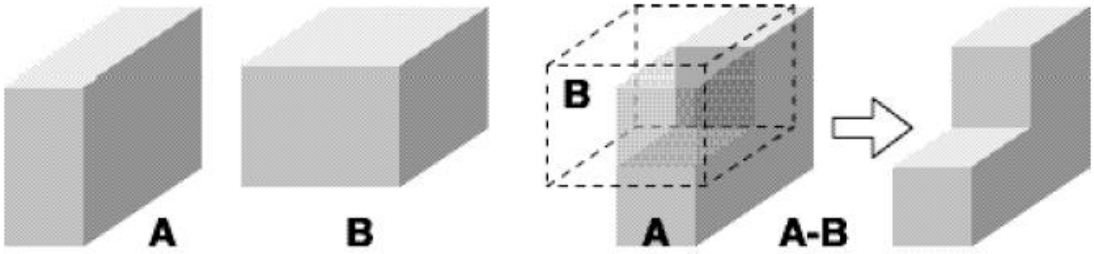
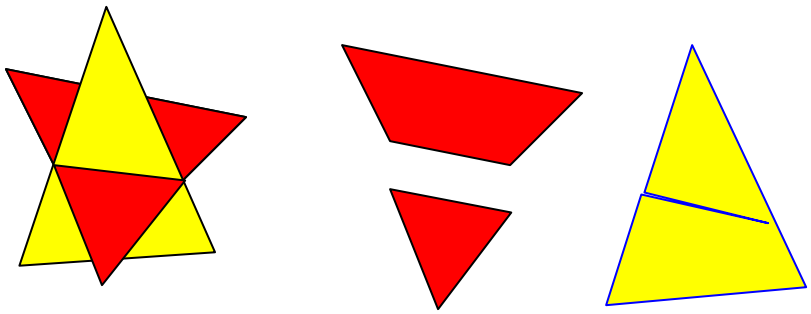
形状として基本立体（プリミティブ）から集合演算を行うデータの表現形式を CSG (Constructive Solid Geometry) という。

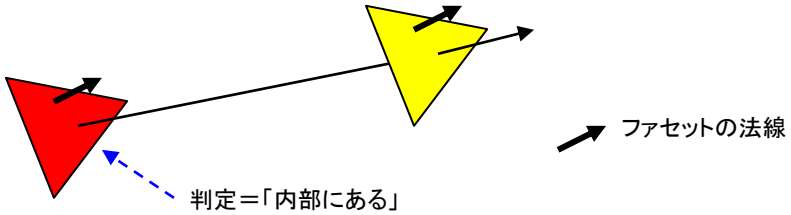


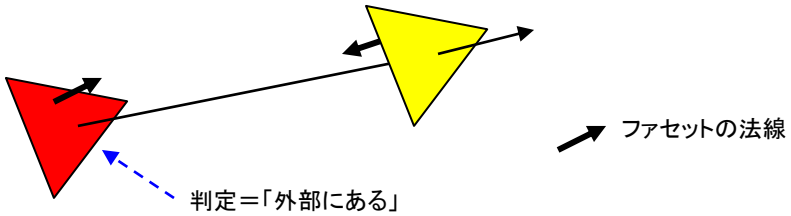
ソリッドを扱うため Winged Edge Data 構造等を使うのが美しいと思うがいつも位相が保たれていなければ成り立たない。浮動小数点で計算する事を考えるとかなり厄介である。しかもレベルの高いコード記述能力が必要というのがさらに問題。
もっと安直なのは位相については知らん顔して演算できないか？
それで単純に思いつくのは以下の方法（まぁ誰でも最初に思いつくだろうけど、、）
ソリッド（ポリゴン）各ファセット（三角形）をもう一方のソリッド各ファセットで交差関係をチェックしてファセットをスプリットしてやる。
この時点でソリッド A、ソリッド B が分離可能な状態になっている。



後は各ファセットがソリッド内部か外部にあるのか判定して集合演算子に従って取捨選択すれば良い。この方法は安直なのでソリッドの位相が多少は崩壊していても集合演算できる点だが逆にアルゴリズムからすぐに分かるがバラバラのファセット集合になって結果が出力される点だ。

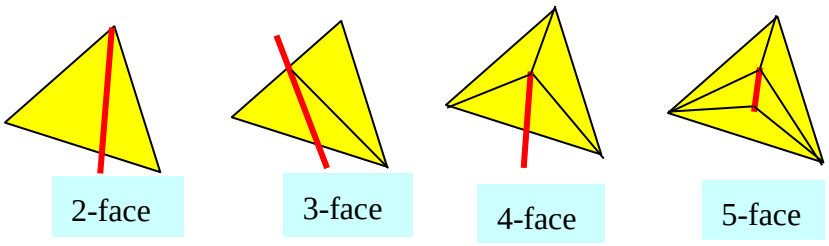
各ファセットがソリッド内部か外部にあるのか判定は単純だ。



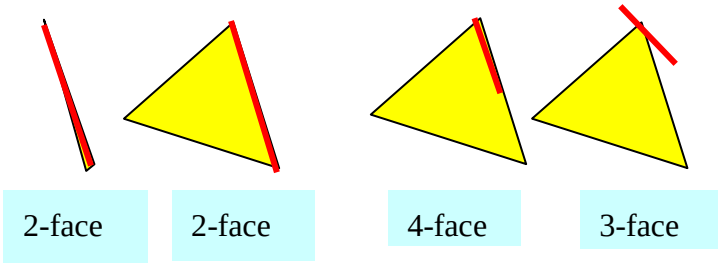


法線方向に Ray を飛ばして突き刺さったファセットの法線との向きを比較すれば良い。
表に突き刺さるなら「外」にあり、裏に突き刺さるなら内部にあると判定できる。
当然だけど複数のファセットと突き刺さるが一番手前に突き刺さるファセットと比較する。
だから Ray を飛ばすという（光を遮るところを見つける）。

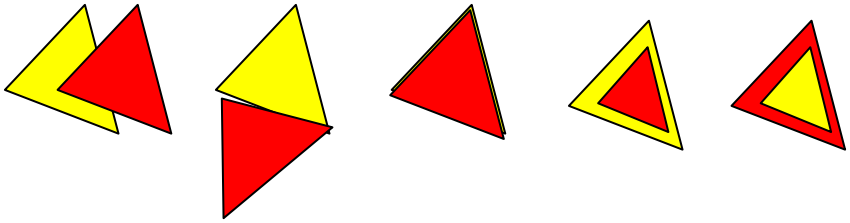
ここまではアルゴリズム。
実に簡単な実装で可能になるのだが問題は沢山ある。
まず、ファセットのスプリット処理。これは基本パターンは４パターンしかない。これによってファセット（黄色）をどんな風にスプリットすれば良いのかは予め決めておける。
N－face は何個にスプリットされるかを表している。



さて、これはあくまで基本パターンだ。厄介なのは以下のパターン。



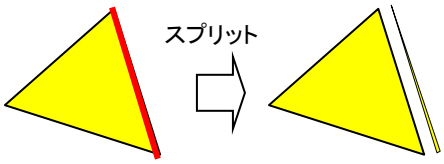
さらに厄介なのは次のパターン



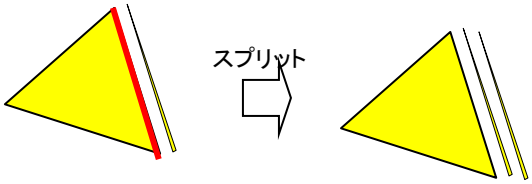
これらは交差せずにペタリと接触するケース。
ただ、エッジとファセットの交差関係に着目して分類すれば上で述べた４＋１パターンのどれかに分類できる。つまりスプリットは可能になるのだが問題は内外判定だ。
ペタリと接触しているので Ray を飛ばすことが出来ないが互いの法線ベクトル見ることで背中合わせかどうかはチェックできるため内外判定も可能だ。

そしてもっとも厄介なのは基本４＋１パターンなのかそれ以外なのかどのパターンなのかを判定する閾値だ。厳密なら単純にパターンわけ出きるが例えば距離が０なら接触それ以外なら交差とか、、、しかし計算は浮動小数点で行うためそんな判定は不可能。
つまり０．０００１以下なら接触それ以外なら交差とかいう処理になる。
しかし、０．０００１０００００１は？
詳しくは書けないがこういう時の工夫でアルゴリズムの良し悪しが決まってくる。

しかし、どうやっても対処できないケースに出くわすのが常だ。その時のためにエラー訂正の仕組みも入れておくべき。問題は何を持ってエラーとすべきなのかだ。
上記で述べたアルゴリズムはパターンの分類分けに誤っても実装上の破綻は無いのでそ知らぬ顔をして結果を返してくる。
なので処理が旨く行く間はエラー訂正の仕組みは入れないで出力結果に異常があったときにどうして異常なのかを調査してその条件を抽出してエラー訂正の仕組みを入れた。



本来はこれでスプリット完了だがまだスプリットされないかをチェックすると再びスプリットされてしまう。



つまりこの処理は無限ループとなってしまう。