

# Java Applet等による 立体パズルの再現と その自動解法

ソフトウェア情報学部

和島 茂

<http://www.aomori-u.ac.jp/staff/wajima/>

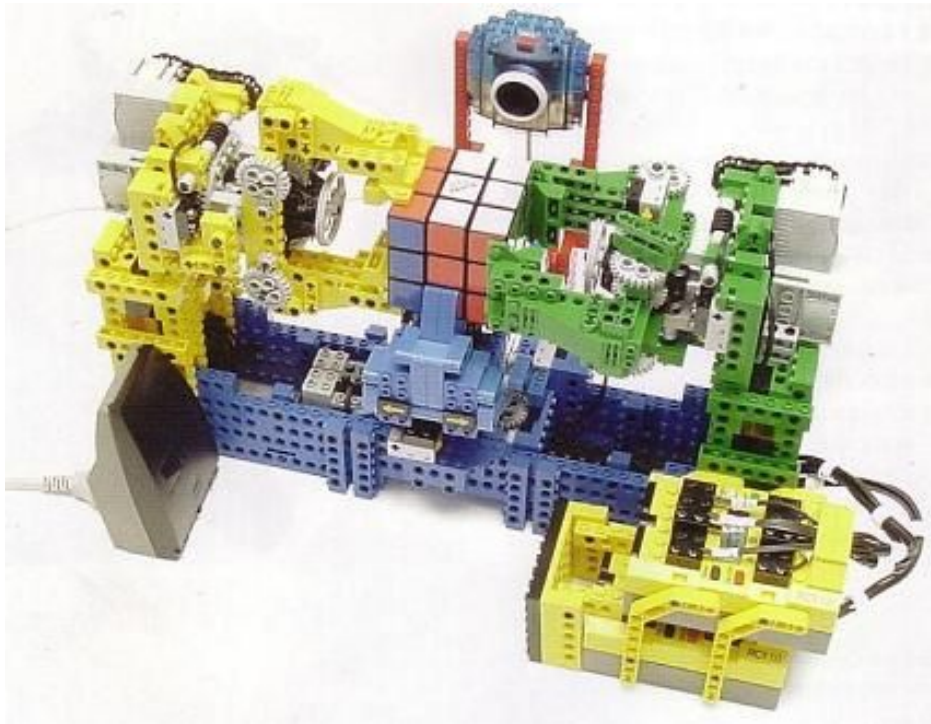
# 目次

---

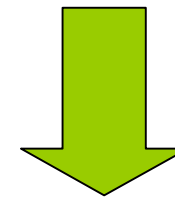
- 発端
- VBAでの解法ルーチン開発
- $4 \times 4 \times 4$ のキューブから一般化へ
- Visual C#・Java Appletへの移植
- 他のパズルへの応用
- 今後の課題

# 発端

---



- MindStormsにルービックキューブを解かせている？



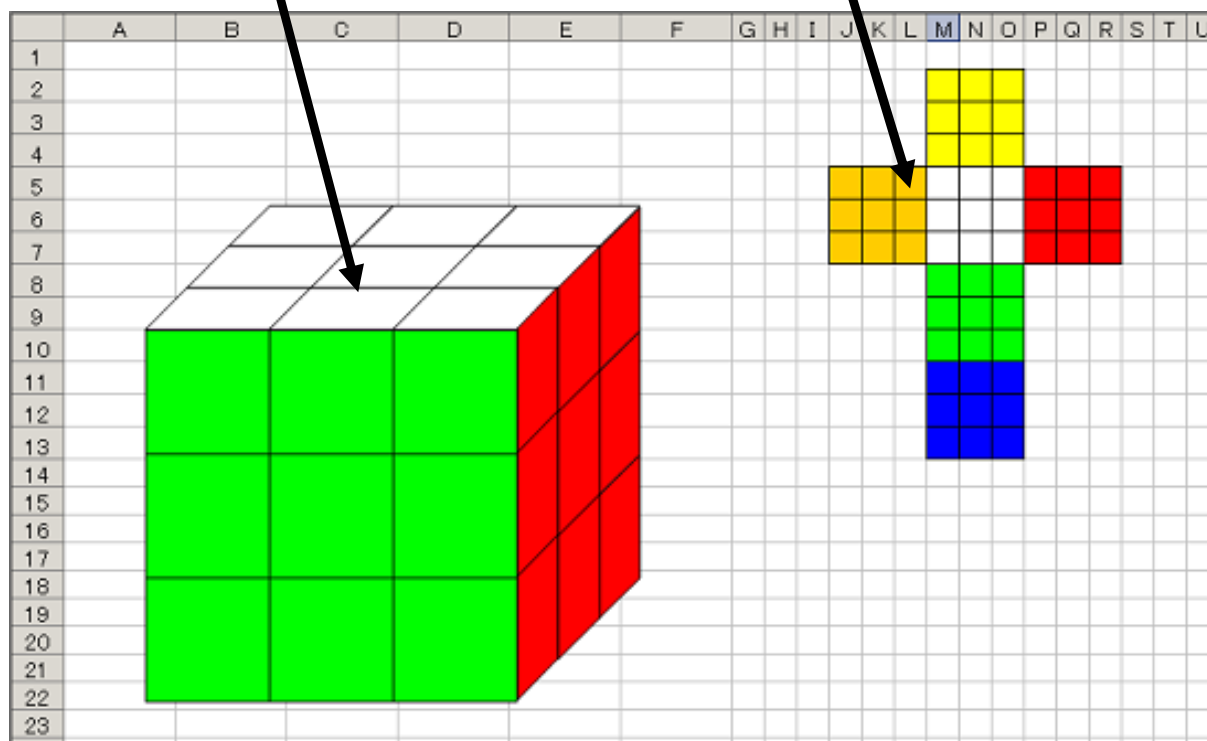
- とりあえずVBAで解法のルーチンを

「Make:」誌より

# VBAでの表示

オートシェイプの  
平行四辺形

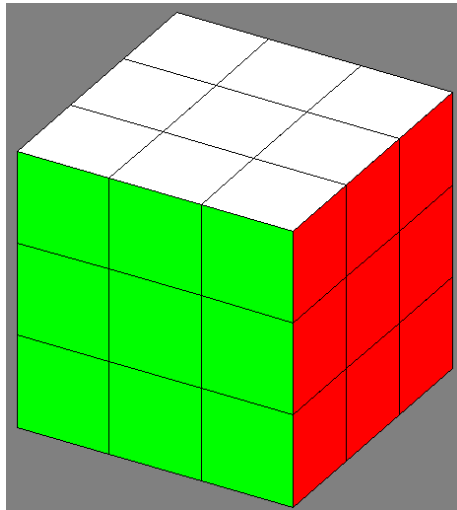
正方形に整形したセル



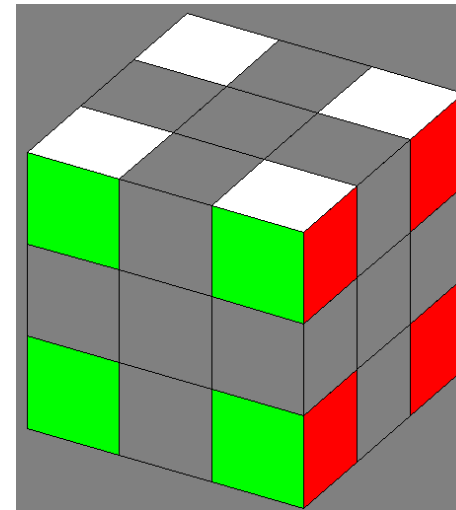
オートシェイプとセルの色を  
変えればキューブが回転  
しているように見える



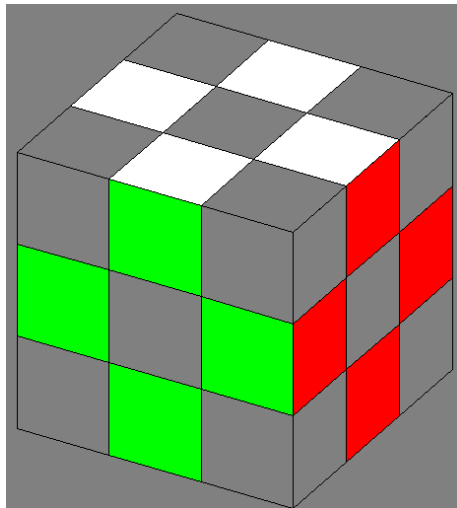
# ルービックキューブ



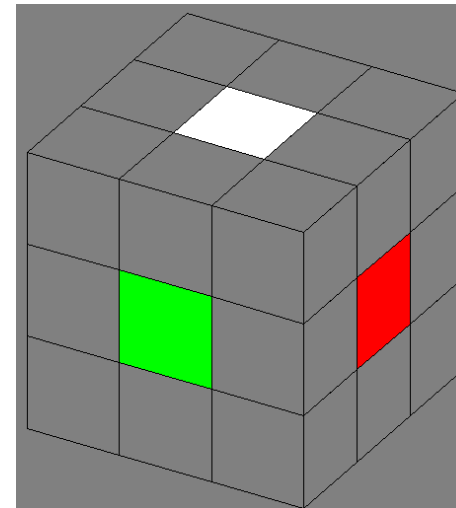
完成状態



コーナーピース  
(8個)



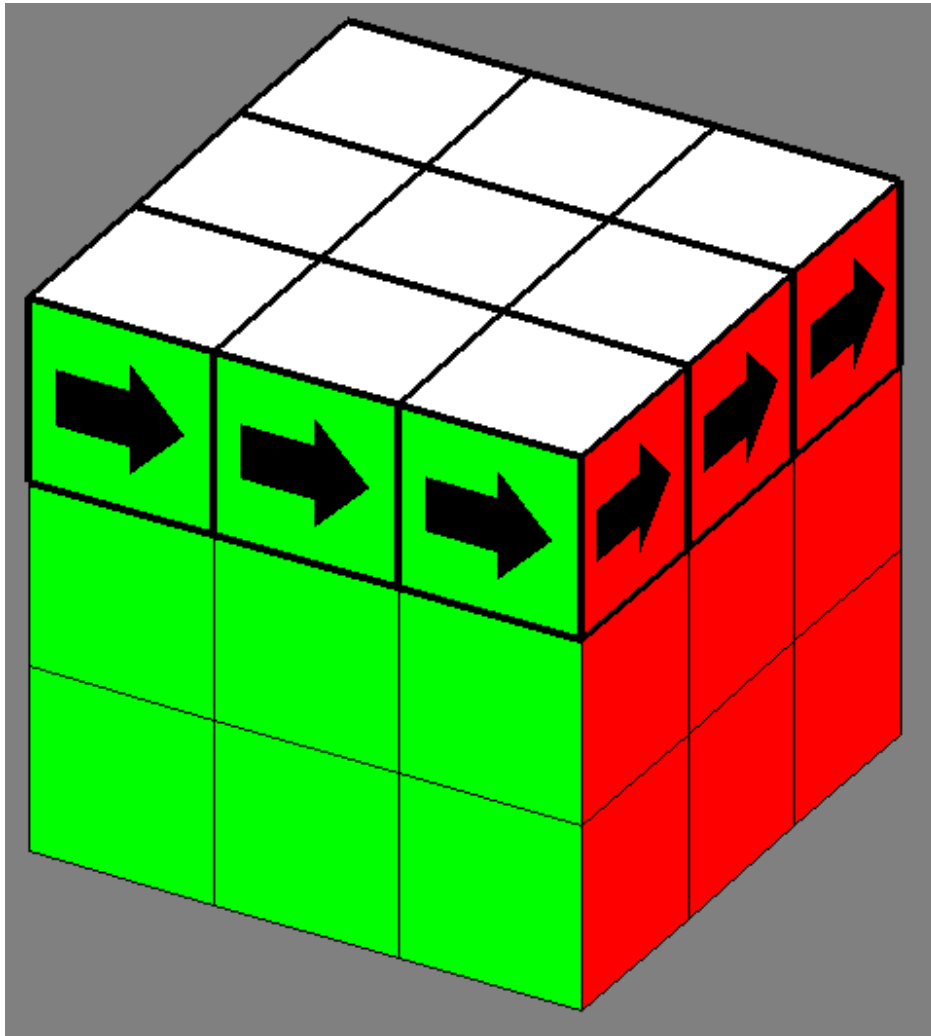
エッジピース  
(12個)



センターピース  
(6個)

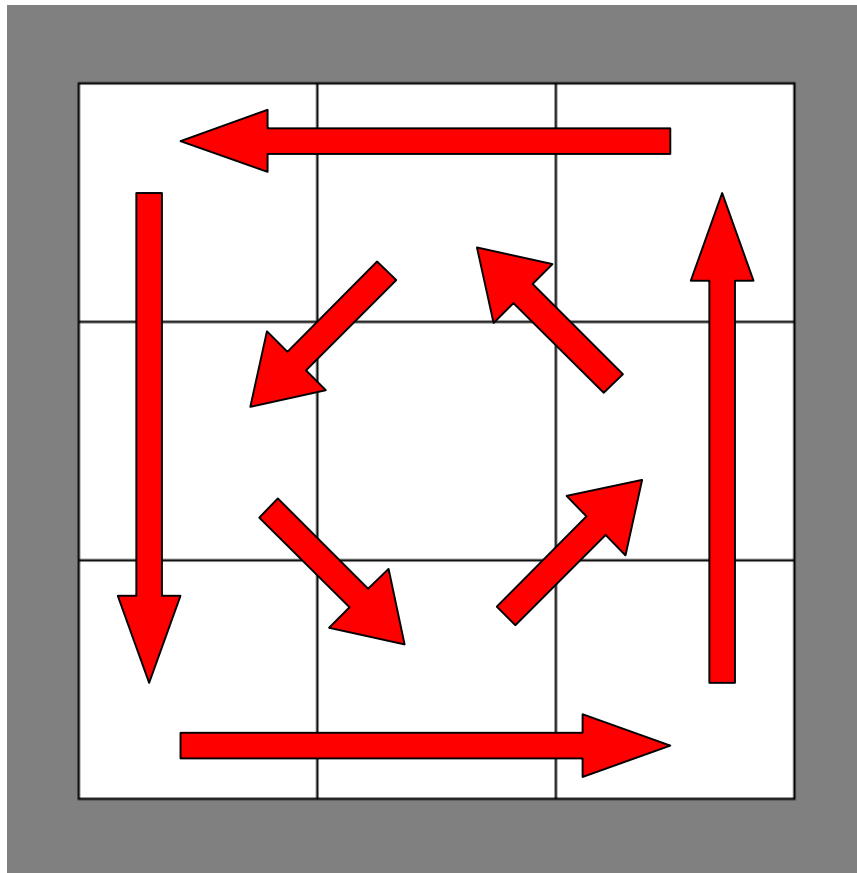
# 回転の最小単位

---



- 層単位で90度回転
- 9個のピースが同時に動く
- ピースの位置が変化
- ピースの向きが変化

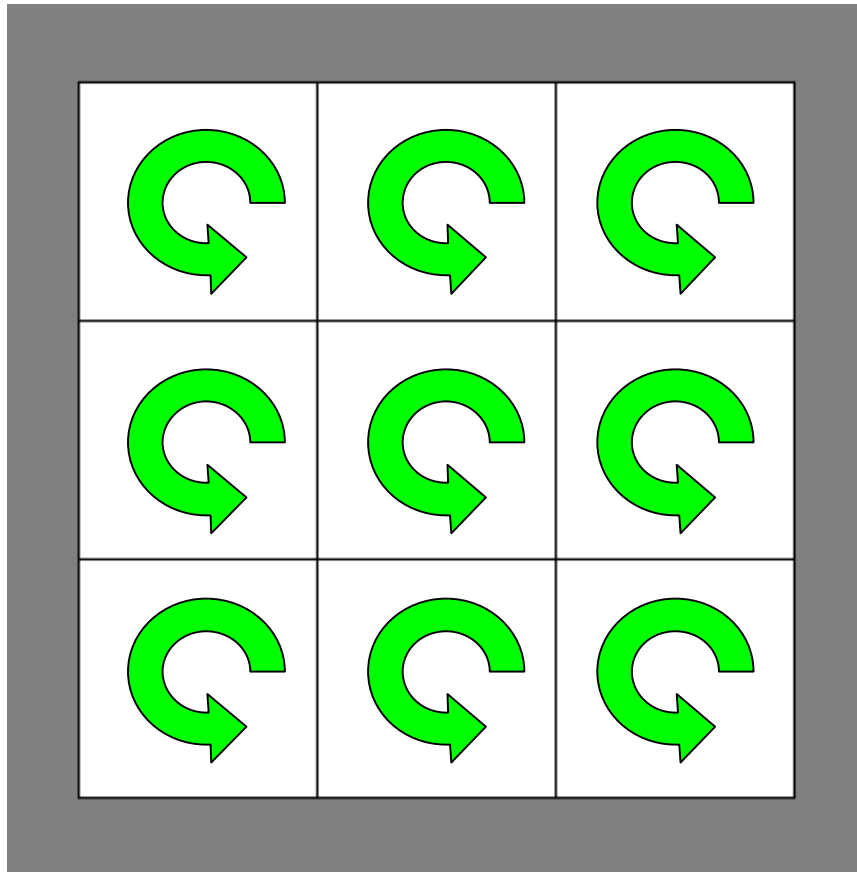
# 位置の移動



- 8個のピースが図のように移動する
- どのようにしてもセンターピース同士的位置関係は変わらない

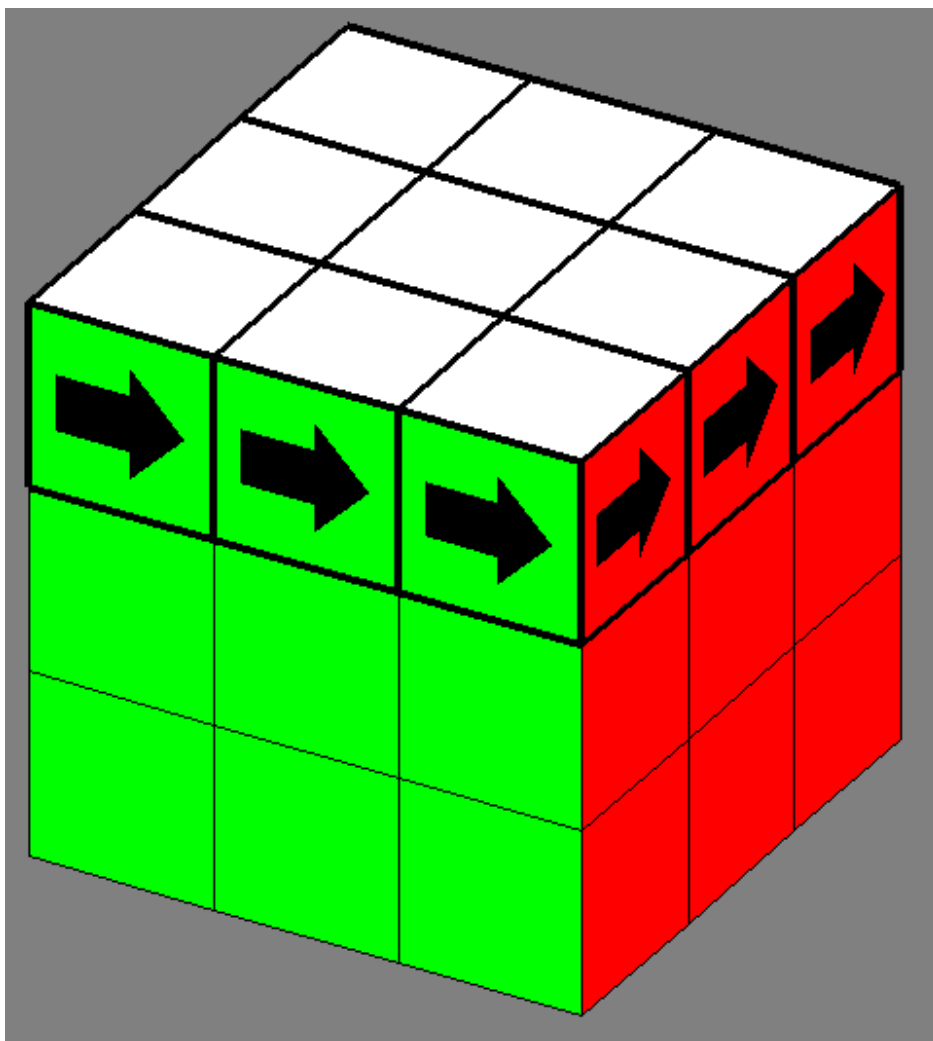
# 方向の変化

---



- 9個(中間層では8個)のピースの向きが変化する

# 回転



## □ 1回の回転がもつ情報

- 回転軸
- 回転方向
- 層番号

`rot.Axis = "y"`

`rot.Direction = 1`

`rot.Layer = 2`

(任意のキューブの状態) = (完成状態) × (回転1) × (回転2) × ...

# 自動解法

---

## □ 計算機的な解法

- 与えられた状態を表す等価な回転の積のなかで最短のものを推論

## □ 人間的な解法

- ピースをある規則に従い順番に揃えていく
- 序盤以外ではそれまでに揃えた部分を崩さない手を使う

# 解法ルーチンに必要な情報

---

- 26個のピースの位置と方向を正しく入れれば完成

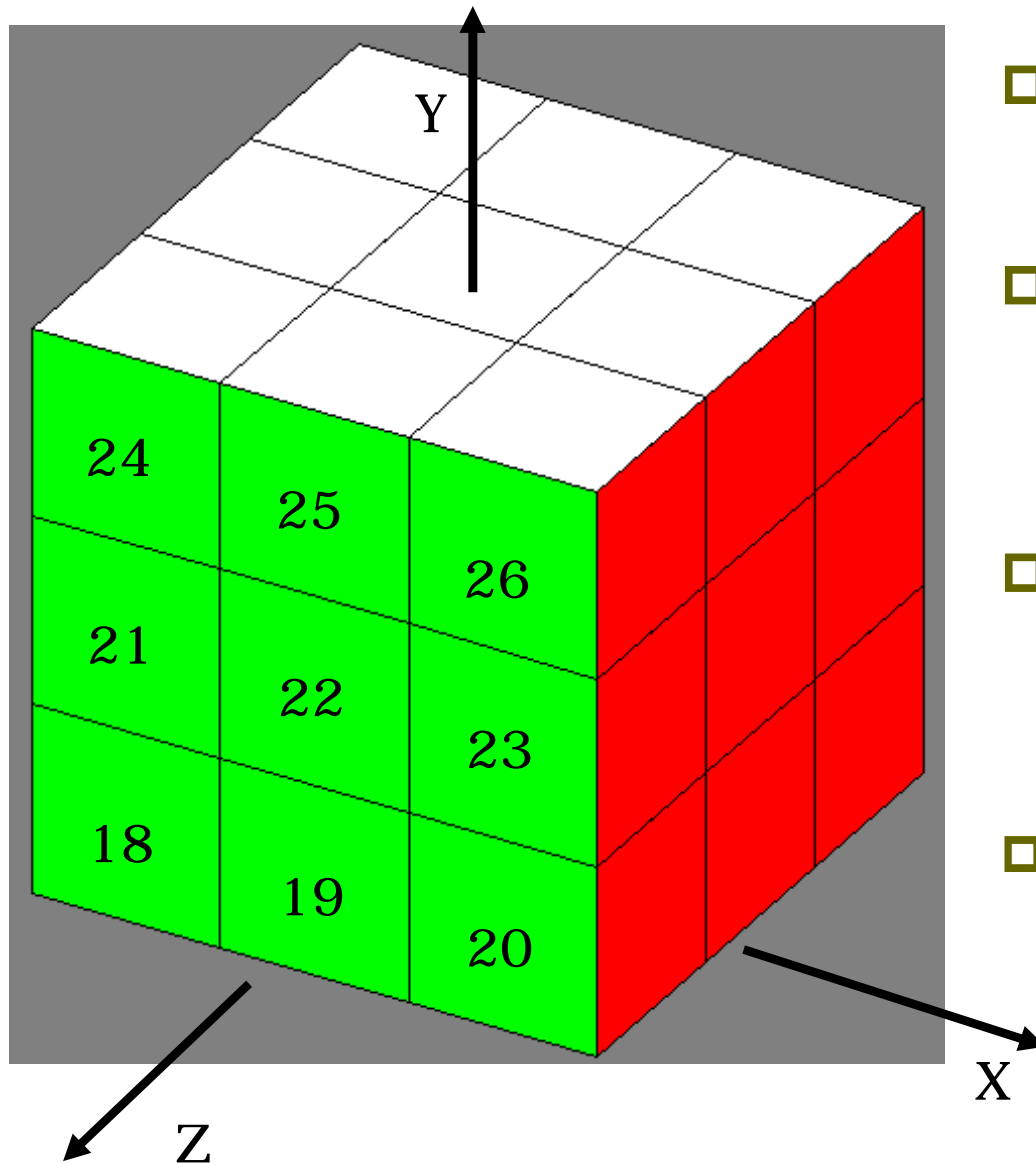
ピースの位置・方向の情報

```
graph TD; A[ピースの位置・方向の情報] --> B[表示]; A --> C[解法の手]
```

表示

解法の手

# ピースの位置

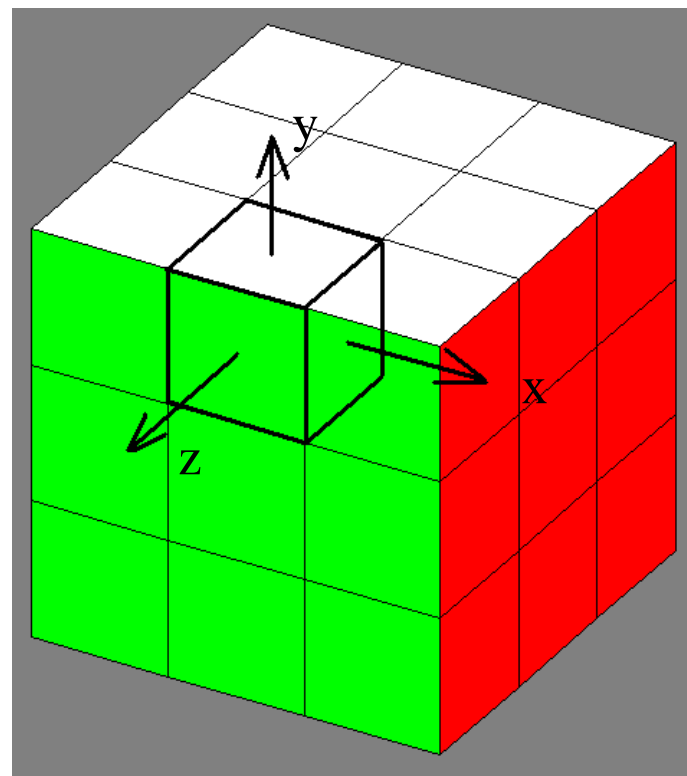


- ピースを配列で表わす
- ピースに0 ~ 26の固有番号を振る
- $(i, j, k)$ にあるピースのindexは  $i + 3(j + 3k)$
- 配列のindexと固有番号が一致 正しい位置

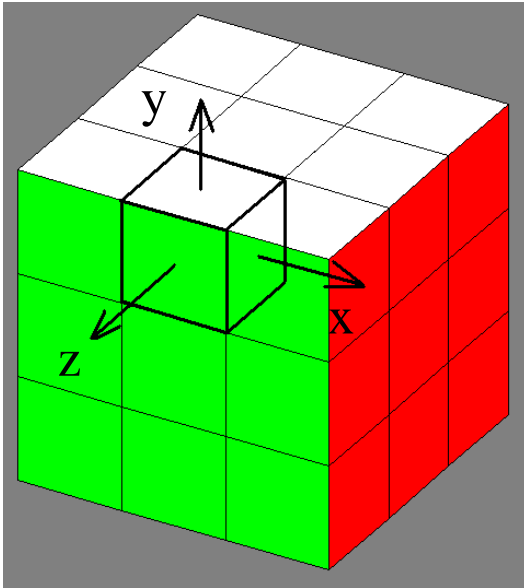
# ピースの方向

- ピースはx, y, zの方向ベクトルをもつ
- ピースの方向は3 × 3の行列で表される

$$\begin{array}{c} \text{x} \\ \text{y} \\ \text{z} \end{array} \begin{array}{ccc} \text{X} & \text{Y} & \text{Z} \\ \left( \begin{array}{ccc} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{array} \right) \end{array} \text{方向ベクトルx}$$



# 状態の変化



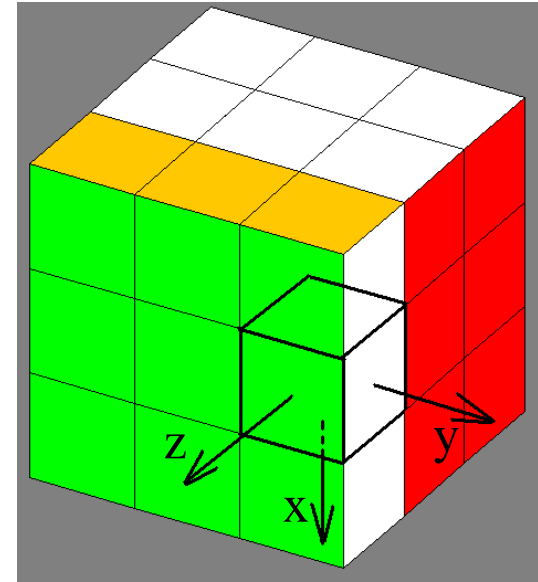
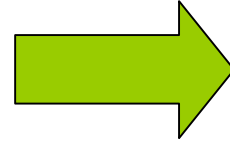
位置(1, 2, 2) 25

piece(25).Number = 25

piece(25).xX = 1

piece(25).yY = 1

piece(25).zZ = 1



位置(2, 1, 2) 23

piece(23).Number = 25

piece(23).xY = -1

piece(23).yX = 1

piece(23).zZ = 1

# ピースの回転

---

- 90度単位で回転する
- (回転後の状態) = (回転前の状態) × (回転行列)

$$X_+ = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{pmatrix} \quad Y_+ = \begin{pmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \quad Z_+ = \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$X_- = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix} \quad Y_- = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{pmatrix} \quad Z_- = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

# ピース1個が持つ情報

---

□ Number 固有番号 = 収まるべき位置

□ xX

□ xY

□ xZ

□ yX

□ yY

□ yZ

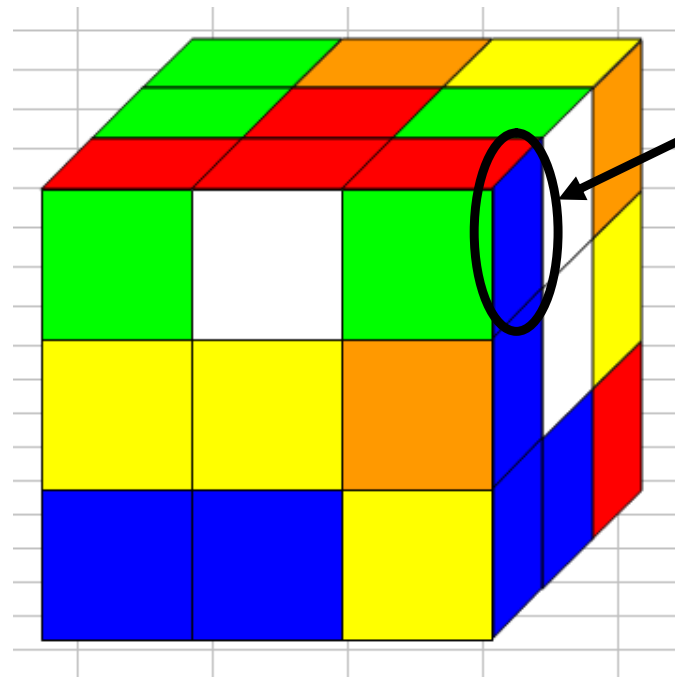
□ zX

□ zY

□ zZ

} 向き

# 色の決定



位置(2, 2, 2)の X(正方向)面

piece(26)の方向行列

$$\begin{array}{c} \text{x} \\ \text{y} \\ \text{z} \end{array} \begin{pmatrix} & \text{X} & \text{Y} & \text{Z} \\ \begin{array}{|c|} \hline 0 \\ \hline \end{array} & 1 & 0 \\ \begin{array}{|c|c|c|} \hline -1 & 0 & 0 \\ \hline \end{array} \\ \begin{array}{|c|} \hline 0 \\ \hline \end{array} & 0 & 1 \end{pmatrix}$$

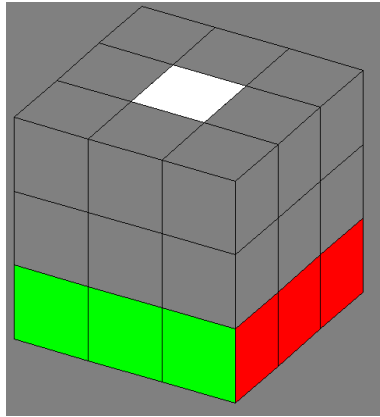
ピースの面の色

$x_+$ : 赤     $y_+$ : 白     $z_+$ : 緑

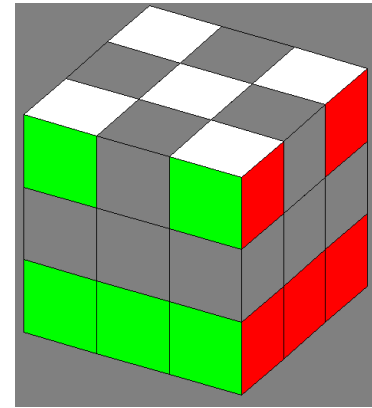
$x_-$ : 橙     $y_-$ : 青     $z_-$ : 黄

yベクトルが-X方向    ピースの $y_-$ 面の色 = 青

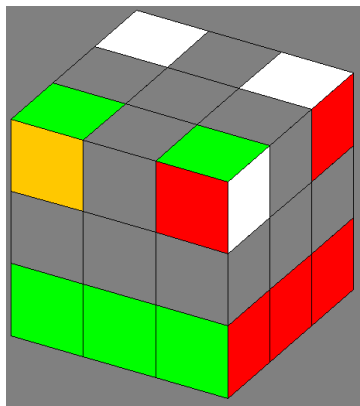
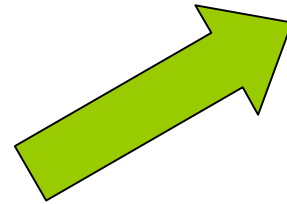
# ツクダ式解法(人間)



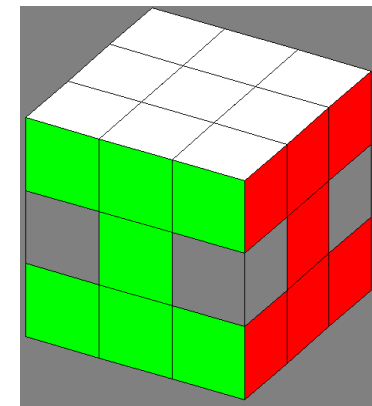
下の面のピースを位置・  
方向とも正しく入れる



上面のコーナーピース  
の方向を修正する



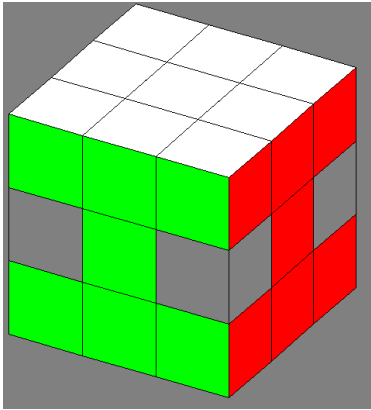
上面のコーナーピースの  
位置を正しく並べ替える



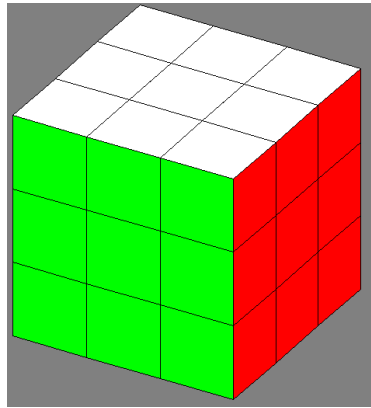
上面のエッジピースを  
正しく入れる

# ツクダ式解法(人間)

---

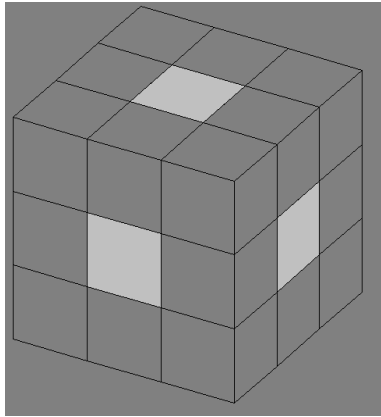


側面のエッジピースの  
位置を合わせる

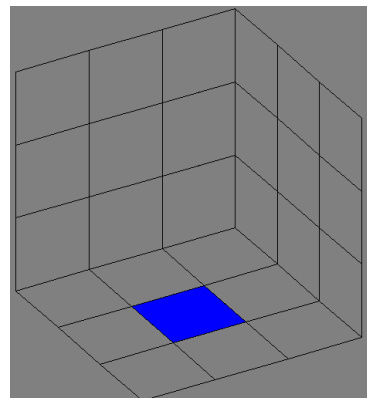
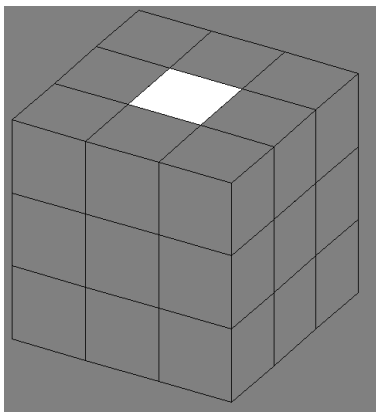
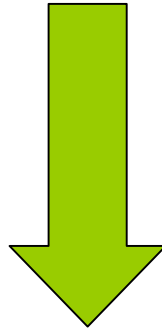


側面のエッジピースの  
方向を合わせて完成

# センターピースの移動

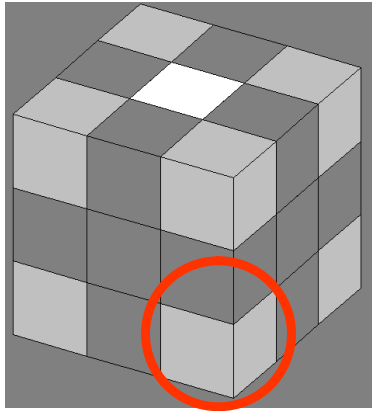


6ヶ所のセンターピースのNumber(あるべき位置)  
を調べる



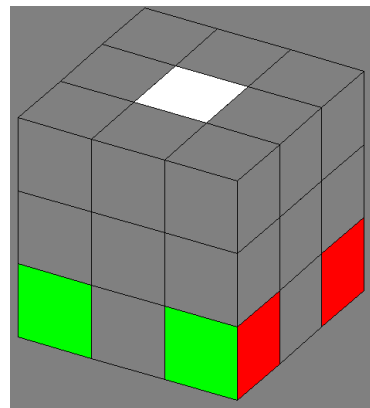
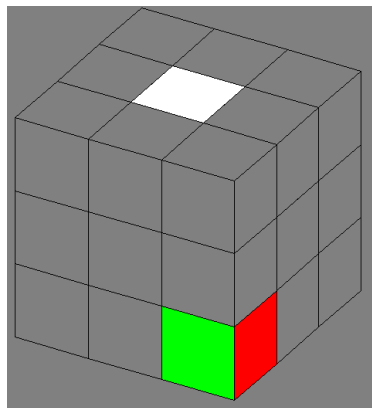
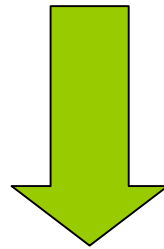
下にあるべきピースをみつけたら、  
全体回転でそのピースを下に移動  
させる

# コーナーピース合わせ



8ヶ所のコーナーピースのNumberを調べ、右下手前にあるべきピースを探す

同時にそのピースの方向も調べる



見つけたら規定の手順で正しい位置・方向に移動させる

他の3つも同様

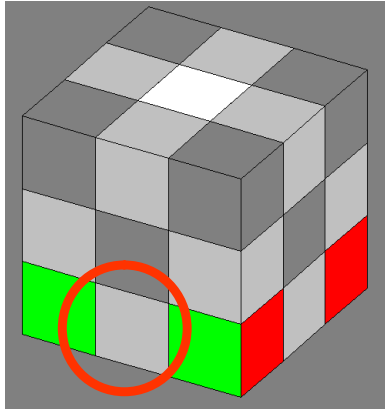
# (手順の実行)

```
' ックダ式
' 青面コーナーキューブ合わせ用
' 右下前に正しいピースを-y面が-Y方向を向くように入れる
dataBCorner(0) = "No.0,2,0,2,Y,"
dataBCorner(1) = "No.1,2,0,2,-X,x2' y2' x2 y2d z2 y2' z2' "
dataBCorner(2) = "No.2,2,0,2,-Z,z2 y2 z2' y2d x2' y2 x2' "
dataBCorner(3) = "No.3,2,0,0,Y,x2 y2 x2' z2 y2d z2' "
dataBCorner(4) = "No.4,2,0,0,-X,x2 y2 x2' x2' y2d x2' "
dataBCorner(5) = "No.5,2,0,0,Z,z0 z2 y2' z0' z2' "
dataBCorner(6) = "No.6,0,0,0,Y,x0 y2d x0' y2' x2' y2 x2' "
dataBCorner(7) = "No.7,0,0,0,X,x0 y2' x0' z2 y2' z2' "
dataBCorner(8) = "No.8,0,0,0,Z,z0' y2 z0 x2' y2 x2' "
dataBCorner(9) = "No.9,0,0,2,Y,x0' y2d x0 z2 y2' z2' "
dataBCorner(10) = "No.10,0,0,2,X,x0' x2' y2 x0 x2' "
dataBCorner(11) = "No.11,0,0,2,-Z,z2' y2' z2d y2d z2' "
dataBCorner(12) = "No.12,2,2,2,-Y,x2' y2 x2 z2 y2d z2' "
dataBCorner(13) = "No.13,2,2,0,-Y,y2' x2' y2 x2 z2 y2d z2' "
dataBCorner(14) = "No.14,0,2,0,-Y,y2d x2' y2 x2 z2 y2d z2' "
dataBCorner(15) = "No.15,0,2,2,-Y,y2 x2' y2 x2 z2 y2d z2' "
dataBCorner(16) = "No.16,2,2,2,-Z,y2' x2' y2 x2' "
dataBCorner(17) = "No.17,2,2,0,-X,y2d x2' y2 x2' "
dataBCorner(18) = "No.18,0,2,0,Z,y2 x2' y2 x2' "
dataBCorner(19) = "No.19,0,2,2,X,x2' y2 x2' "
dataBCorner(20) = "No.20,2,2,2,-X,y2 z2 y2' z2' "
dataBCorner(21) = "No.21,2,2,0,Z,z2 y2' z2' "
dataBCorner(22) = "No.22,0,2,0,X,y2' z2 y2' z2' "
dataBCorner(23) = "No.23,0,2,2,-Z,y2d z2 y2' z2' "
```

```
' 右下前(OK), -y面が-Y方向(OK)
' 右下前(OK), -y面がX方向
' 右下前(OK), -y面がZ方向
' 右下奥, -y面が-Y方向
' 右下奥, -y面がX方向
' 右下奥, -y面がZ方向
' 左下奥, -y面が-Y方向
' 左下奥, -y面がX方向
' 左下奥, -y面がZ方向
' 左下前, -y面が-Y方向
' 左下前, -y面がX方向
' 左下前, -y面がZ方向
' 右上前, -y面がY方向
' 右上奥, -y面がY方向
' 左上奥, -y面がY方向
' 左上前, -y面がY方向
' 右上前, -y面がZ方向
' 右上奥, -y面がX方向
' 左上奥, -y面が-Z方向
' 左上前, -y面が-X方向
' 右上前, -y面がX方向
' 右上奥, -y面が-Z方向
' 左上奥, -y面が-X方向
' 左上前, -y面がZ方向
```

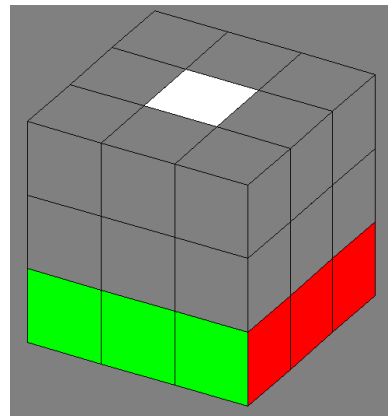
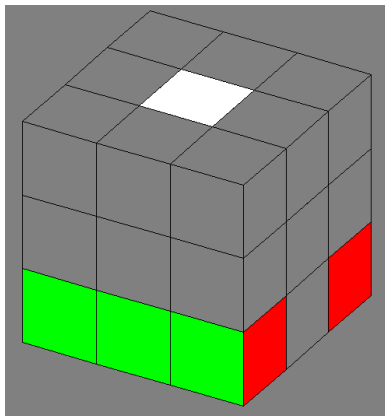
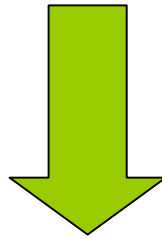
- それぞれのパターンについて実行する手順を決めておく
- 条件にあてはまったら手順を実行する

# エッジピース合わせ



12ヶ所のエッジピースのNumberを調べ、手前下にあるべきピースを探す

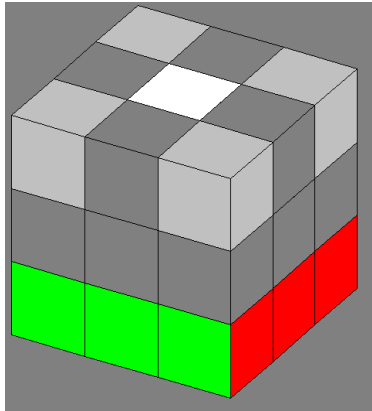
同時にそのピースの方向も調べる



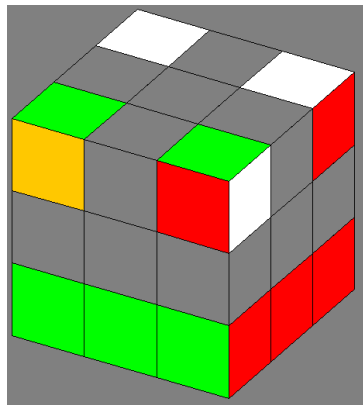
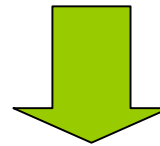
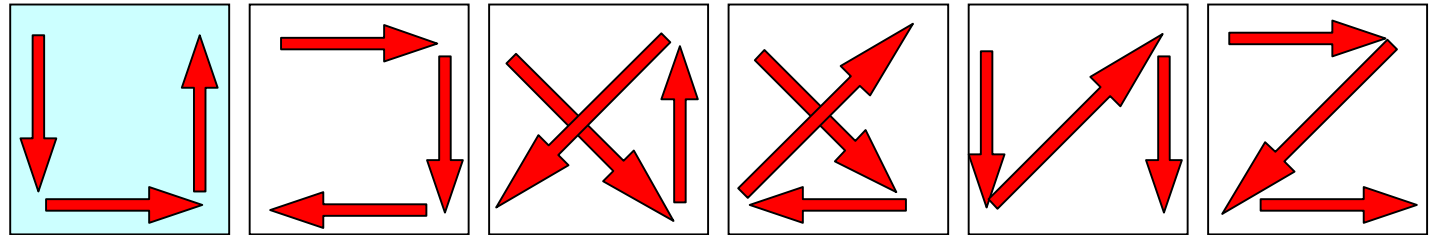
見つけたら規定の手順で正しい位置・方向に移動させる

他の3つも同様

# コーナーピース位置合せ



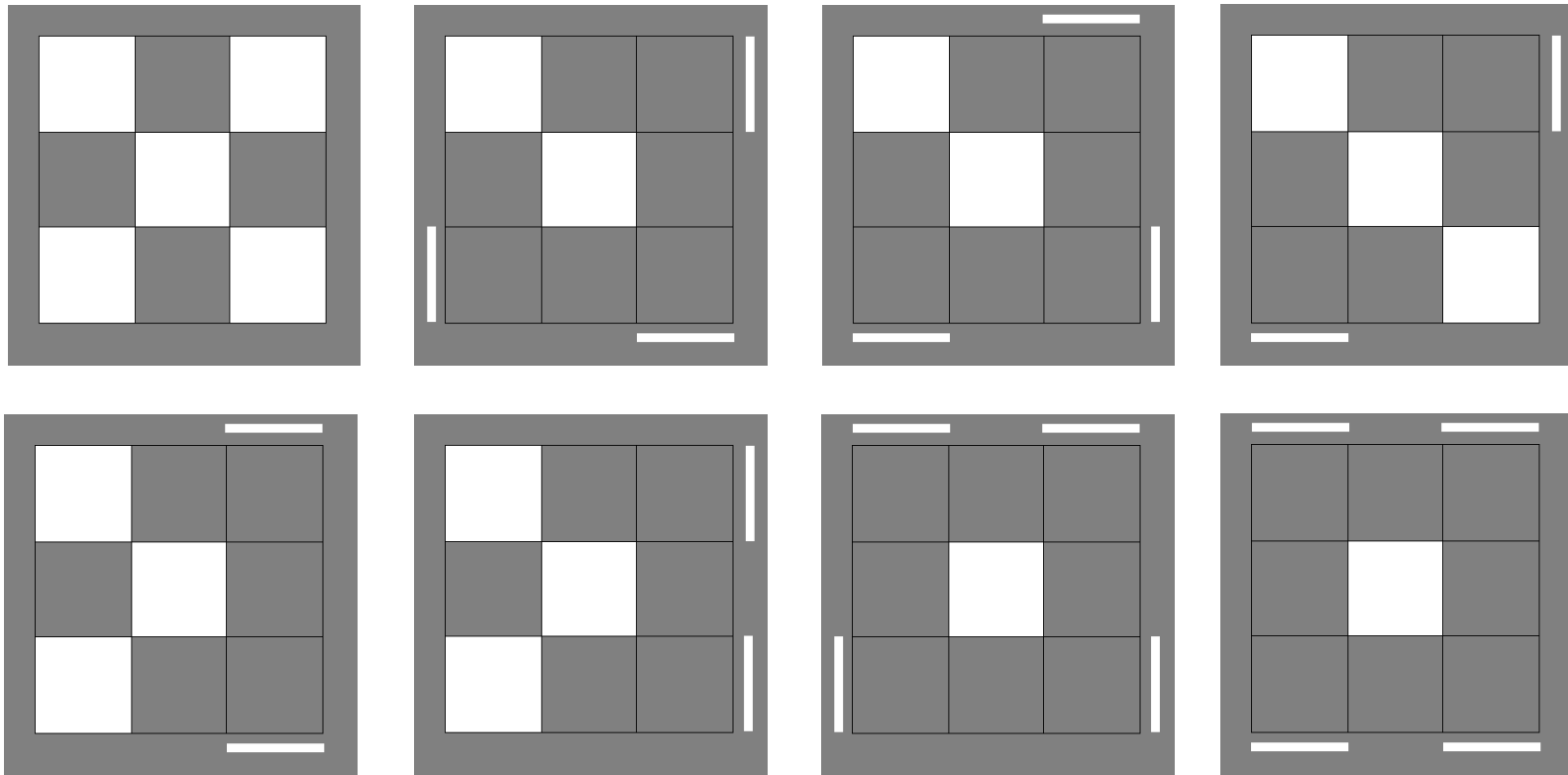
上段の4つのコーナーピースの並び方(円順列)が  
 $3! = 6$ 通りのうちどれであるか調べる



規定の手順でピースの位置関係が正しくなるように  
並べ替える

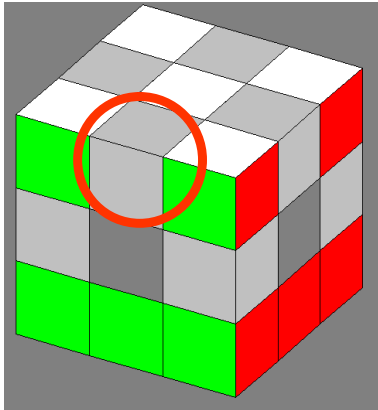
# コーナーピース方向合せ

---



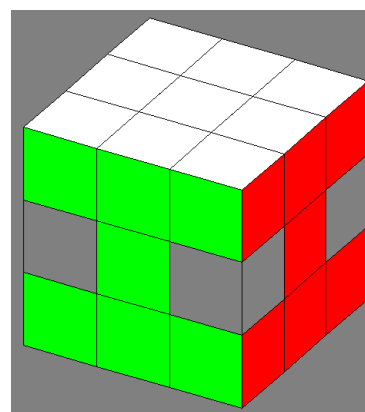
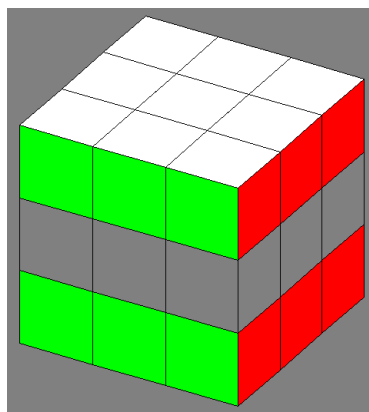
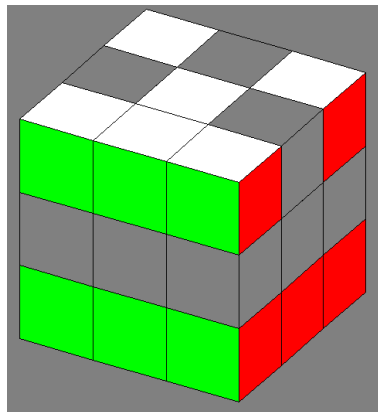
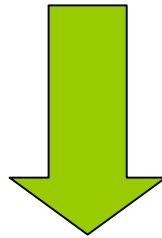
4つのコーナーピースの方向を調べ、8つのパターンのどれであるかに  
応じ、規定の手順で方向を合わせる

# 上エッジピース合わせ



8ヶ所のエッジピースのNumberを調べ、手前上にあるべき  
ピースを探す

同時にそのピースの方向も調べる

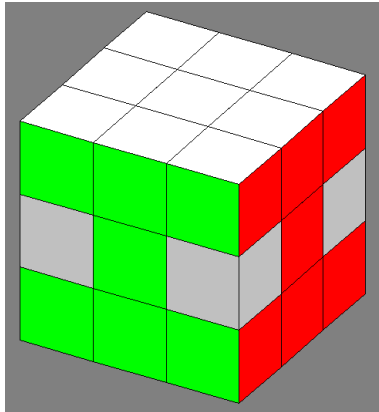


見つけたら規定の手順で正しい位置・方向に移動させる

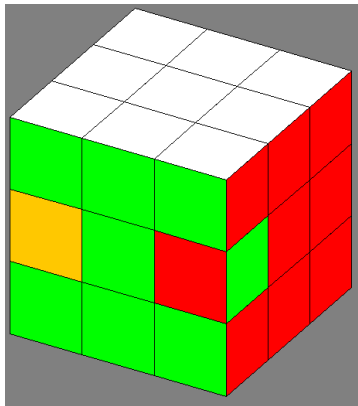
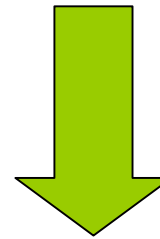
他の3つも同様

最後に2層目を回転させて  
センターピースも合わせる

# 側面エッジピース位置合せ

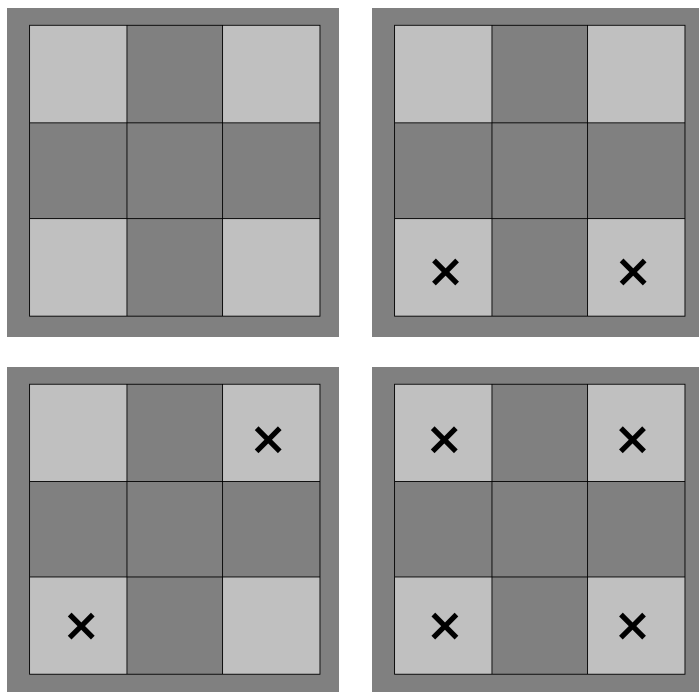


中段の4つのエッジピースの並び方が  
 $4! = 24$ 通りのうちどれであるか調べる



規定の手順でピースの位置が正しくなるように  
並べ替える

# 側面エッジピース方向合せ



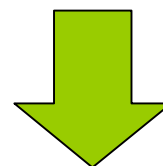
(上から見た図)

( ◻ が正しい向き、× が反対向き)

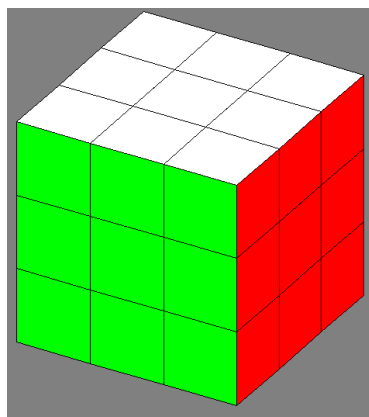
4つのエッジピースの向きが合っているか  
違っているかを調べ、図の4通りのうちどれ  
であるか調べる



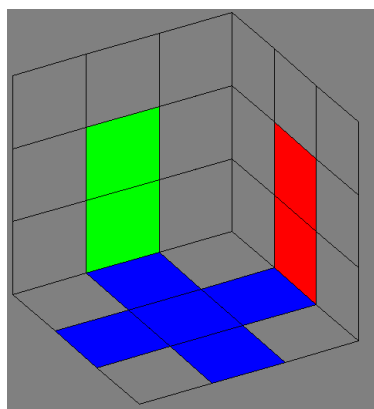
規定の手順でピースの向きが正しくなるように並  
べ替える



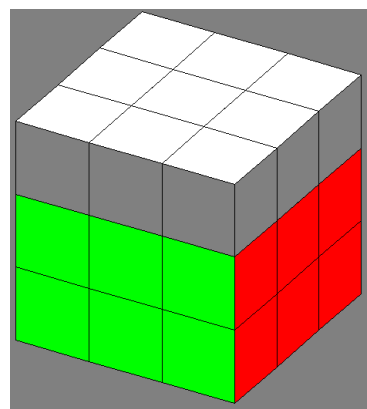
完成！



# LBL(Layer By Layer)法

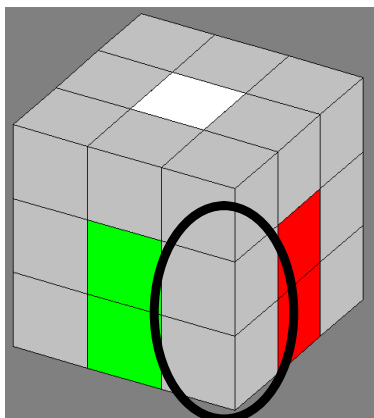
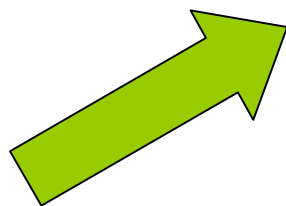
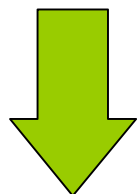


下面のエッジを揃える



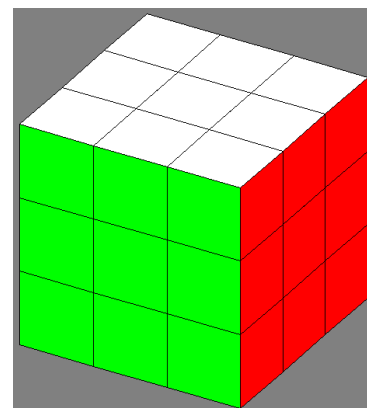
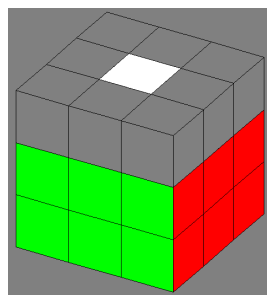
上の面のピースの方向を合わせる

(58通り)



この位置に入る2つの  
ピースの位置・方向の  
全ての組合せについて  
最短の手順を使う

(96通り)



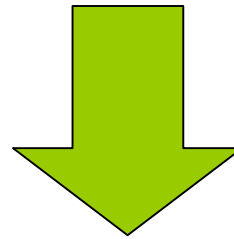
位置を並べ替える

(22通り)

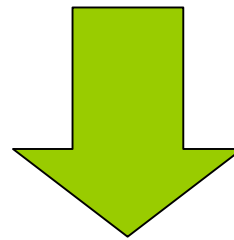
# 絵柄つきルービックキューブ



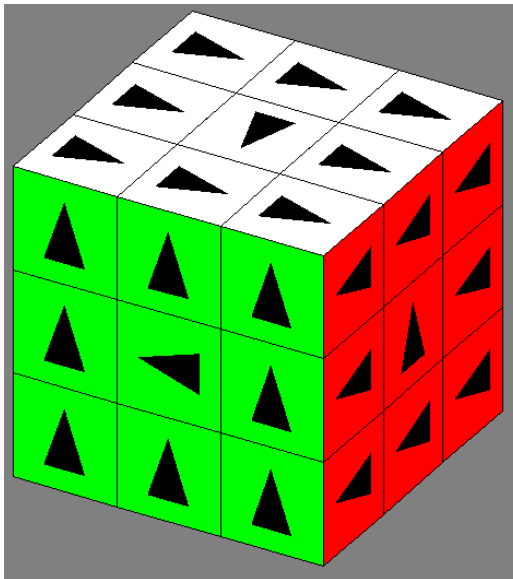
- 絵柄なしの場合のようにして6面を完成させてもセンターピースの方向が狂う



- ここまでの解法では完全な基本状態にはなっていない

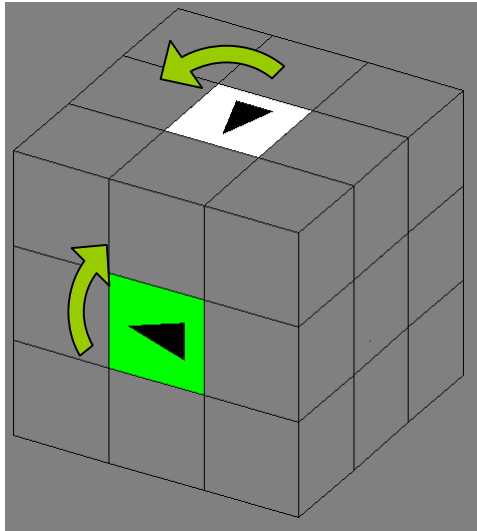


- 方向を合わせるためのルーチンが必要

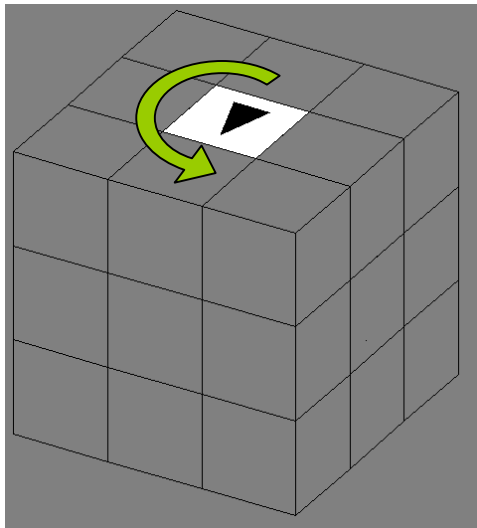


# 可能な移動単位

---

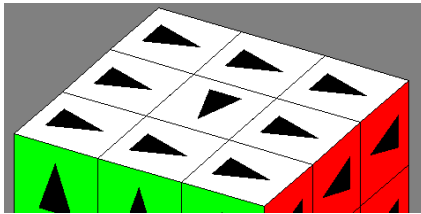


隣り合ったセンターピースを $+90^\circ$ ,  $-90^\circ$  ずつ回す

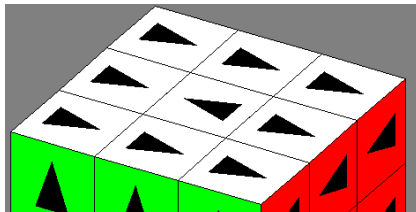


単独で $180^\circ$  回す

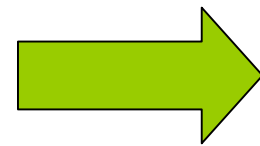
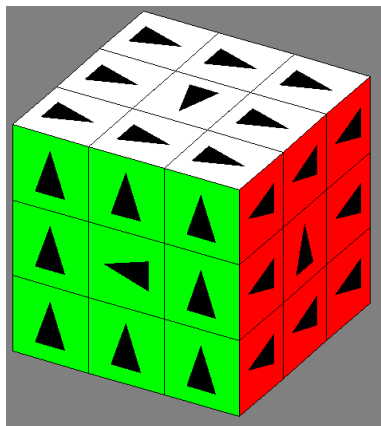
# センターピース方向合わせ



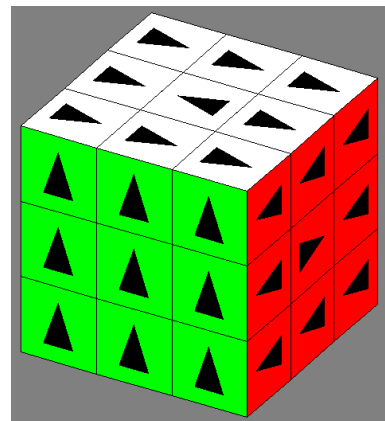
odd : 正しい状態から+90°または-90°ずれた状態



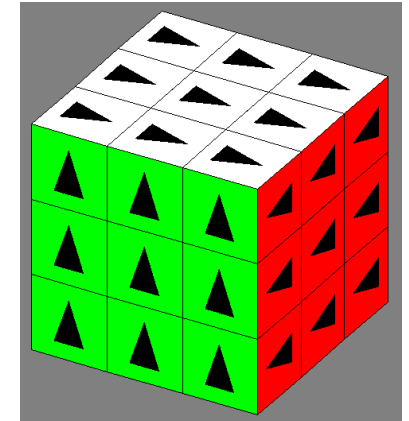
even : 正しい状態から180°ずれた状態



evenだけの状態に

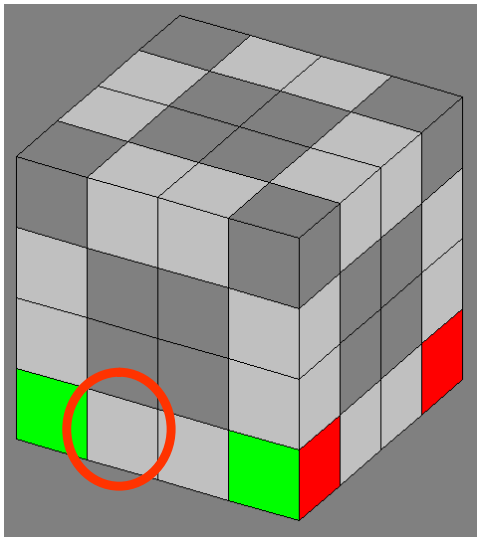
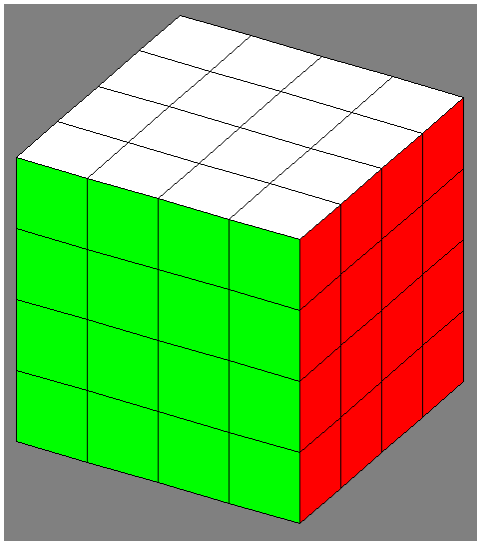


逆向きのピースを反転



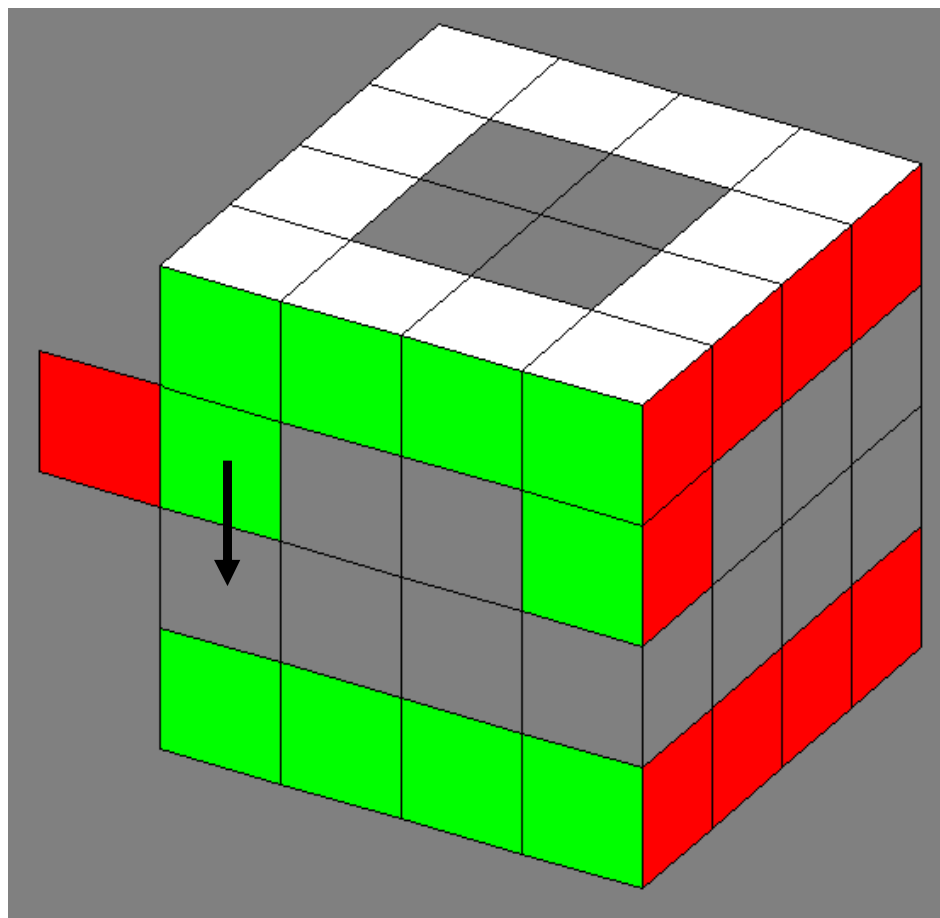
基本状態

# 4×4×4のキューブ



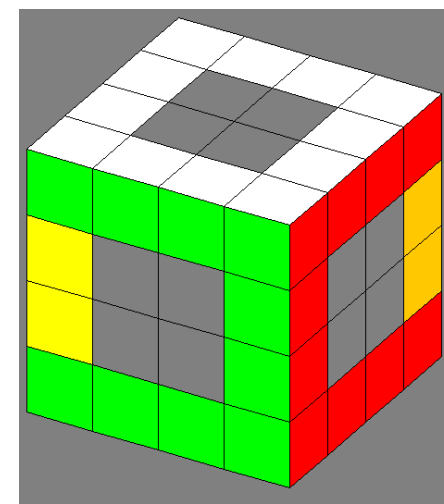
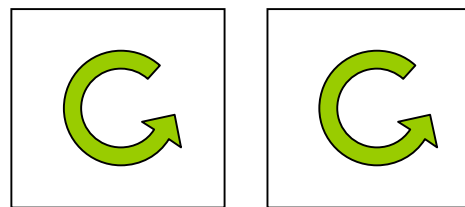
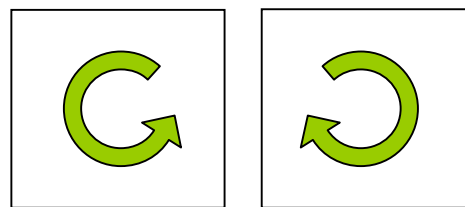
- センターピースがない
- 上下エッジピース合わせ
  - 探すべき位置は前の2倍=24ヶ所
- 横エッジピース合わせ
  - ペアリング
  - 位置合わせ
  - 方向合わせ
- インナーピース合わせ

# ペアリング

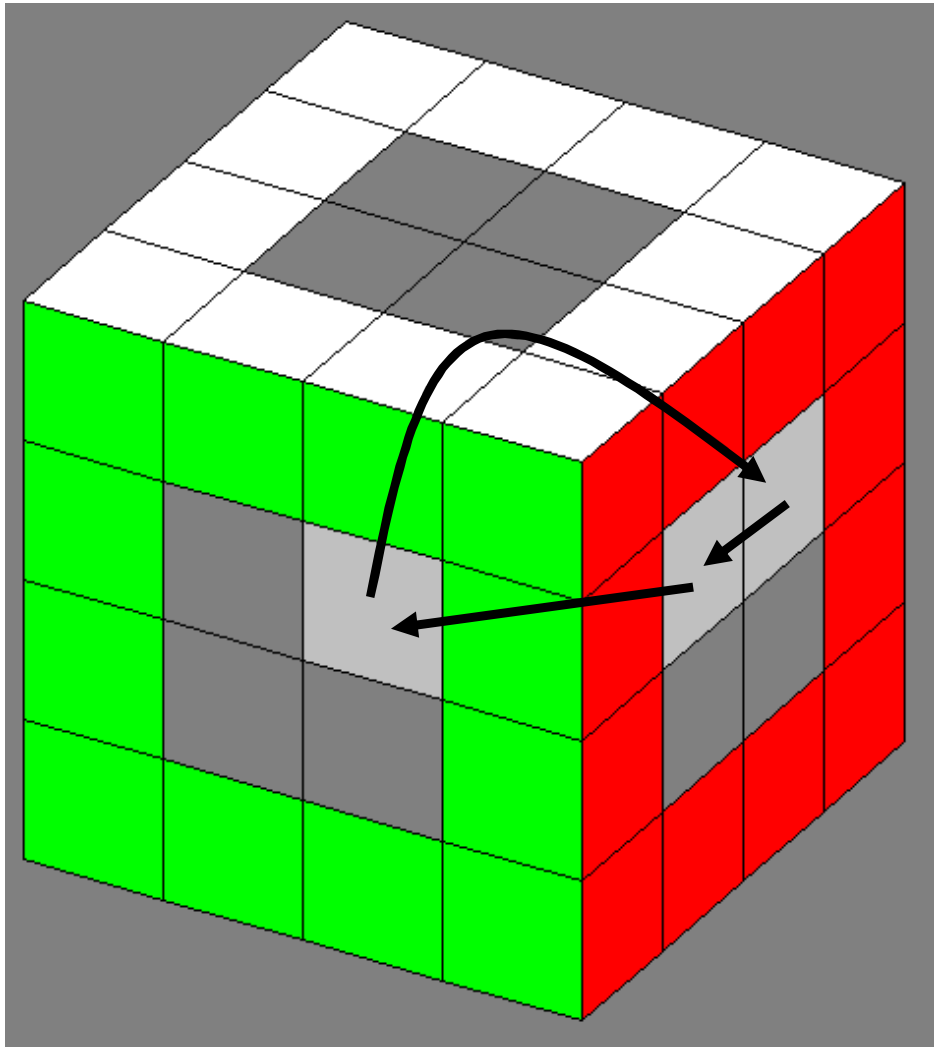


1. 対応するピースが同じ層にあるときは片方を別の層に移動させる

2. 2つの層の順列を一致させる

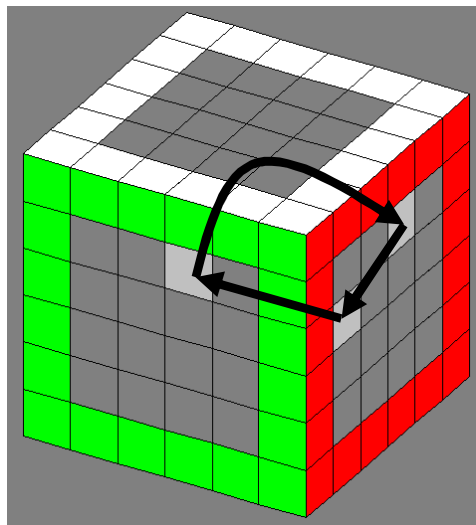
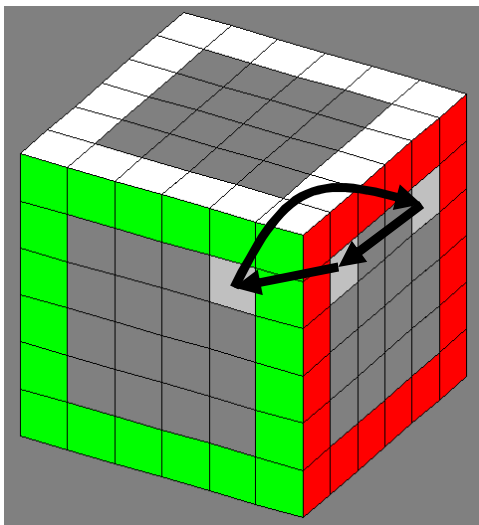
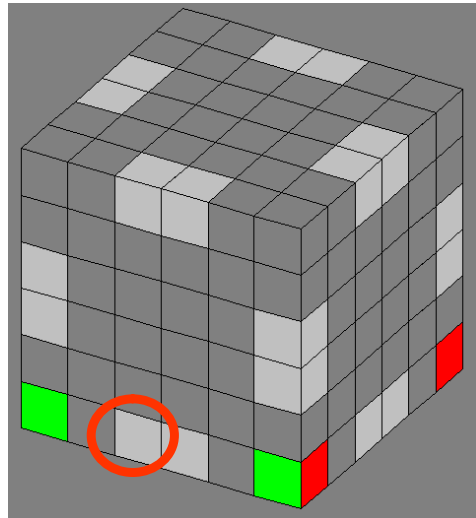
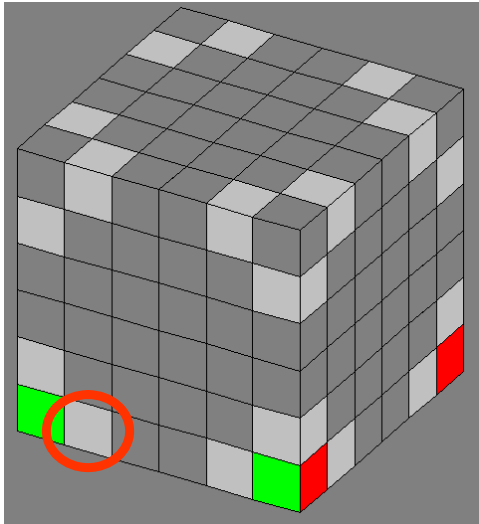


# インナーピース合わせ



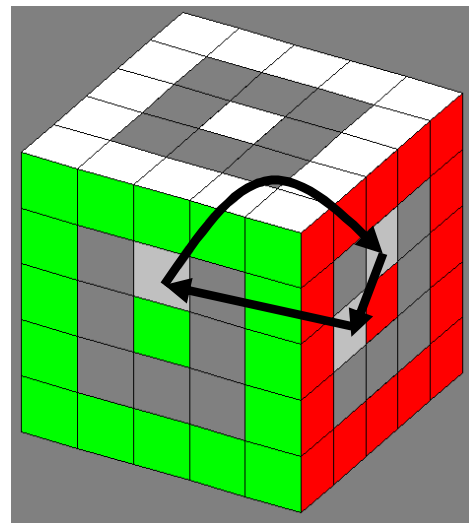
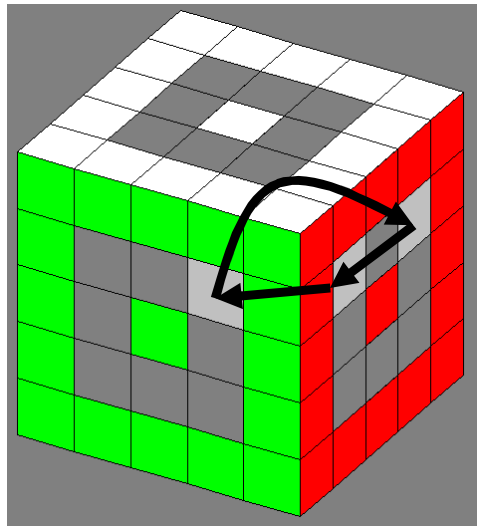
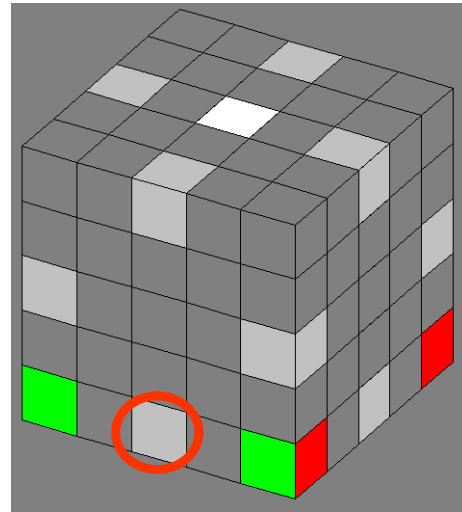
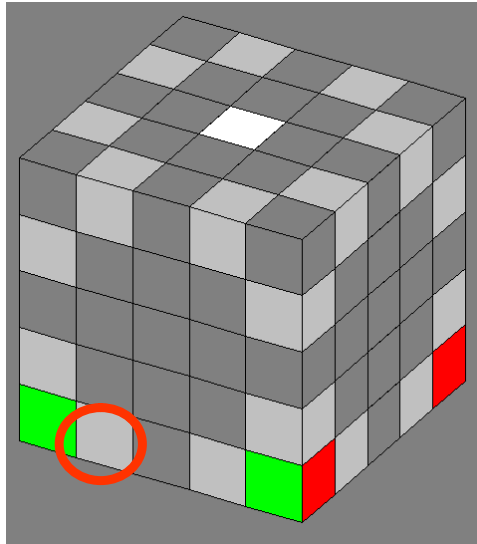
- 規定の手順で3ヶ所のインナーピースをローテーションする
- それぞれの面に同色のピースを入れれば完成

# 一辺のピース数が偶数個の場合



- エッジピースを合わせるときは  $4 \times 4 \times 4$  の時と同様のことを  $N/2 - 1$  回繰り返す
- インナーピースは  $4 \times 4 \times 4$  の時の応用で揃える

# 一辺のピース数が奇数個の場合

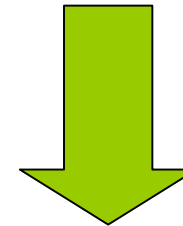


- 中央以外のエッジピースは偶数個のキューブの時と同じ ( $(N-3)/2$ 回)
- 中央のエッジピースは  $3 \times 3 \times 3$ のキューブと同様にして合わせる
- インナーピースは  $4 \times 4 \times 4$ の時の応用で揃える  
一般化

# Visual C#・Java Appletへの移植



- 解法ルーチンは類似のものにも流用できそう
- VBAでは操作・表示系に問題がある



- Visual C#・Java Appletに移植

# 表示系・操作系

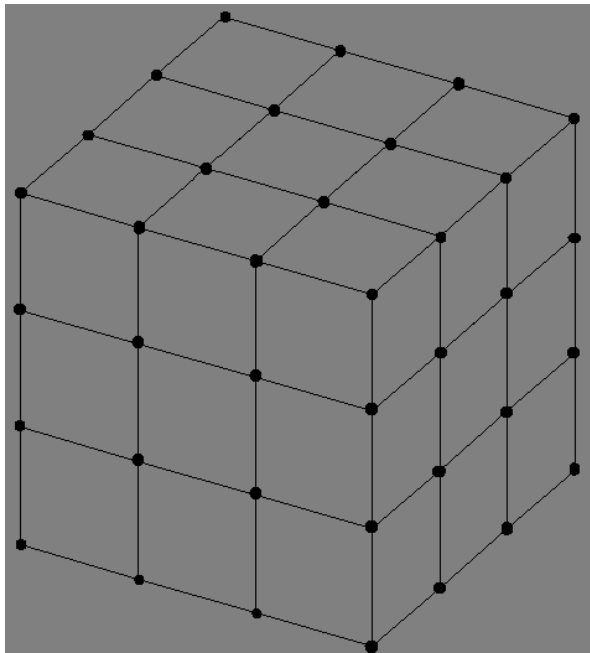
---

- 全体をドラッグして動かせるようにする
- 回転をアニメーション表示する
- ピース上でクリックすると規定の方向に回転
- ピース上にマウスポインタがあるときに矢印で回転方向を表示する
- Visual C#版にはセーブ・ロード機能を付ける

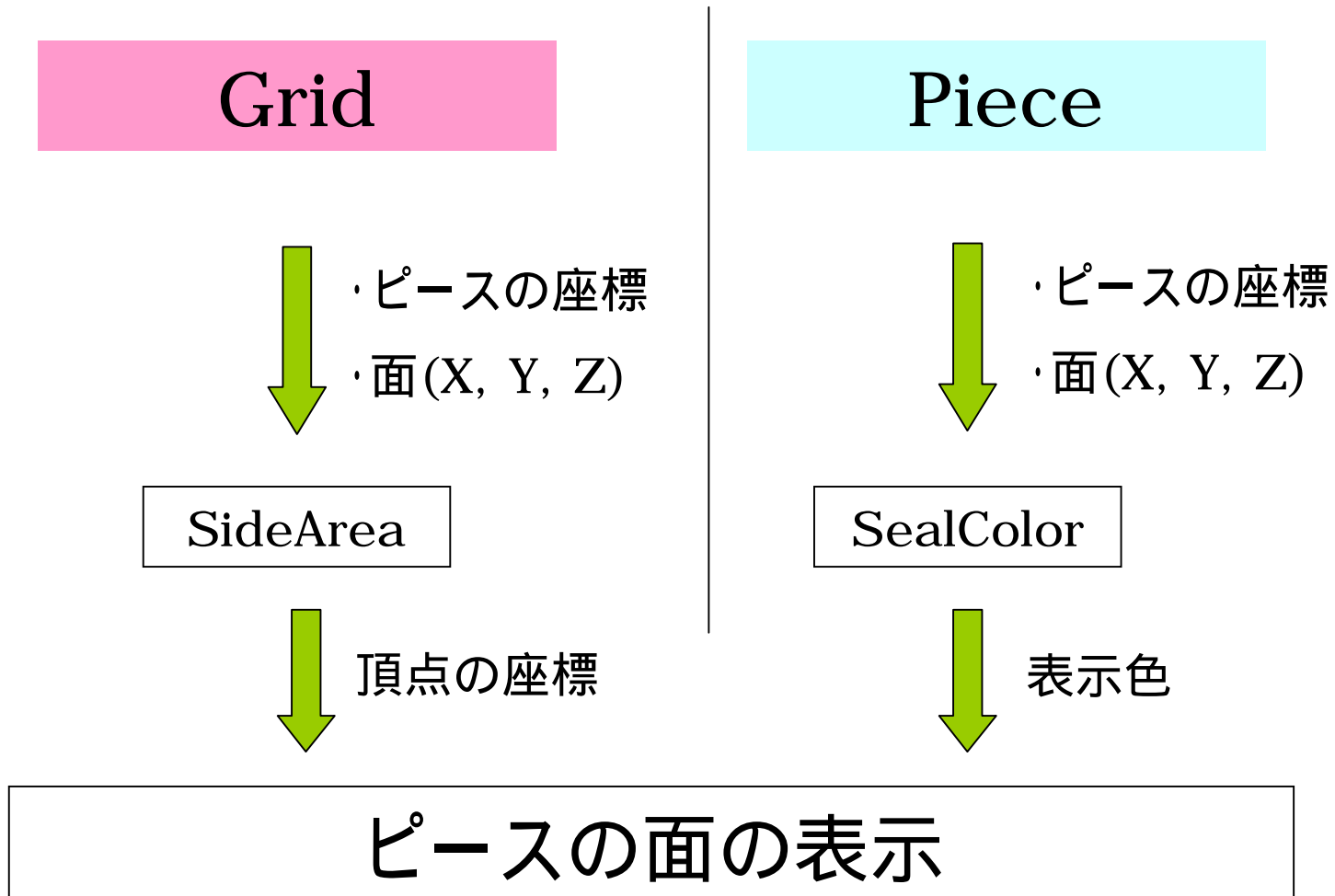
# グリッド

---

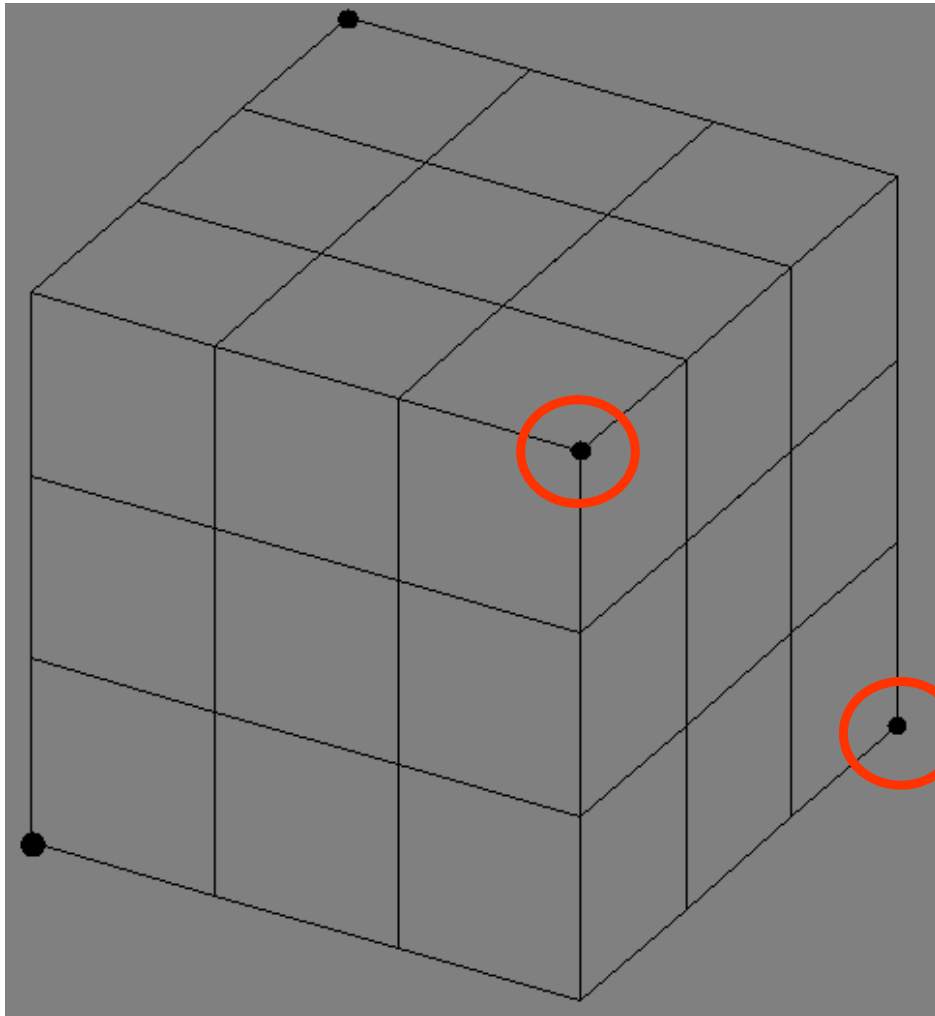
- Grid: 格子点(実数)の配列
- マウスでドラッグしたときなどにGridを書き換える
- ピースの表示はGridの点を結んだPolygonで行う



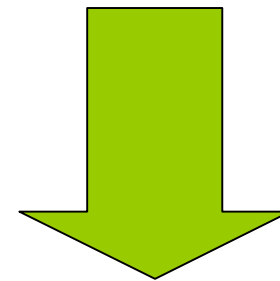
# 面の表示



# 面の前後判定



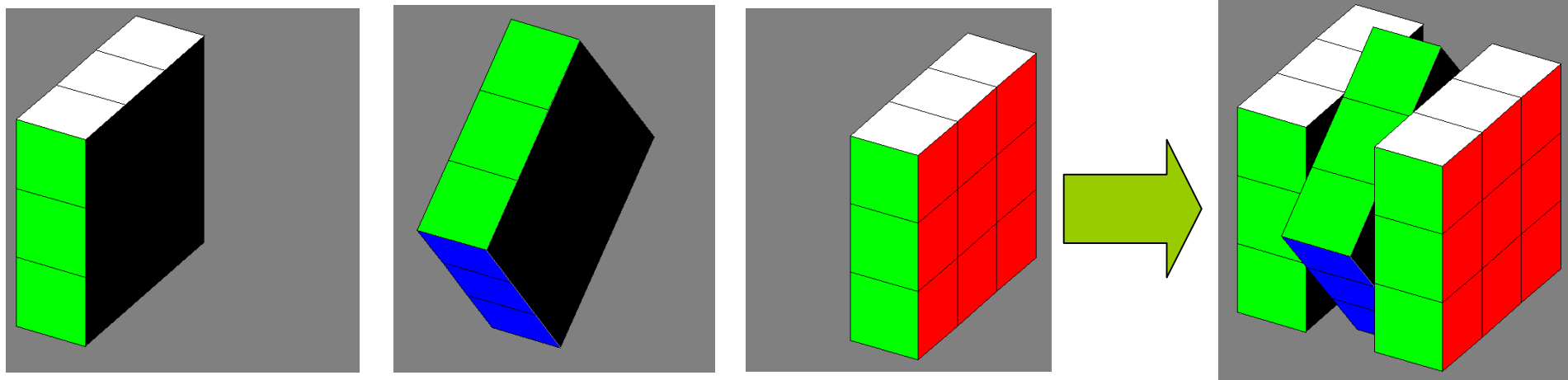
- 回転させると裏側が見える



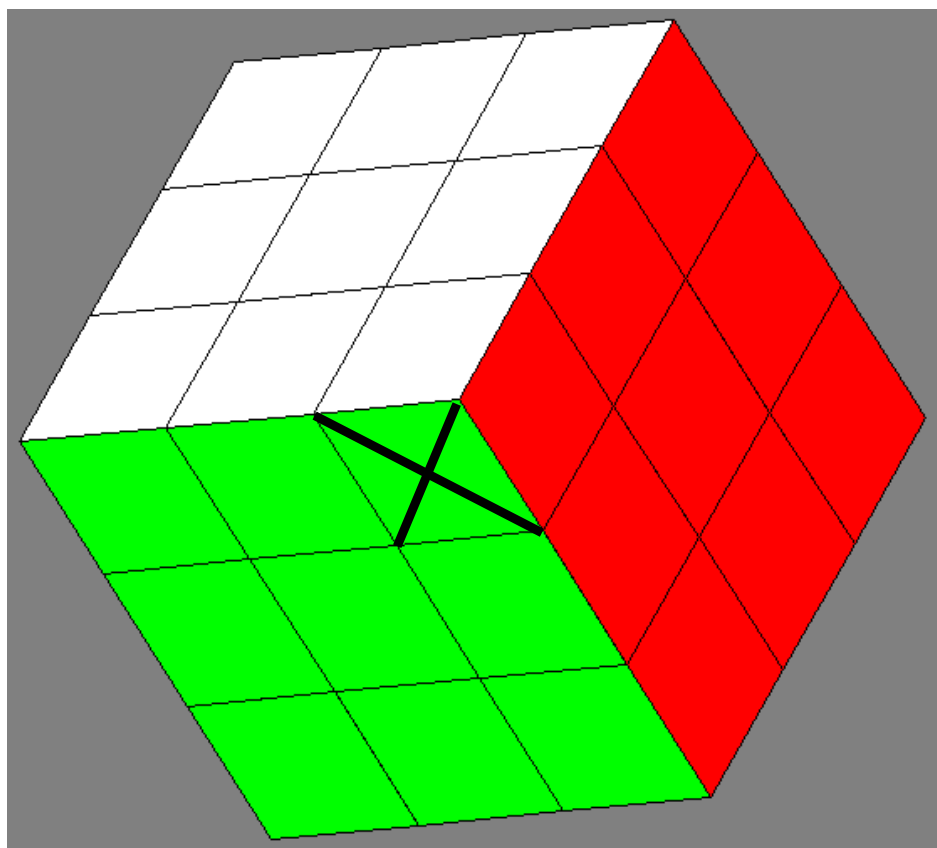
- +X面, -X面のうちどちらを表示するかを決める
  - 2つの点 +X面の法線
  - 法線が手前向きなら+X面
  - 法線が奥向きなら-X面

# アニメーション

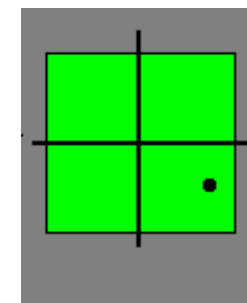
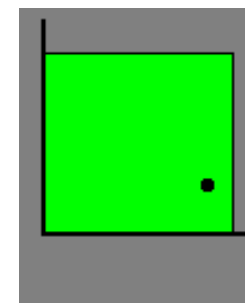
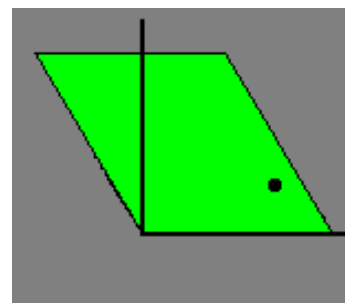
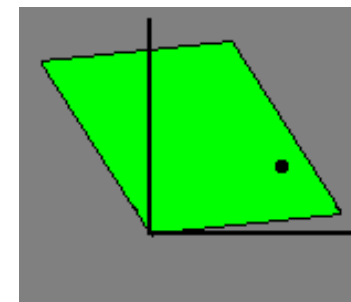
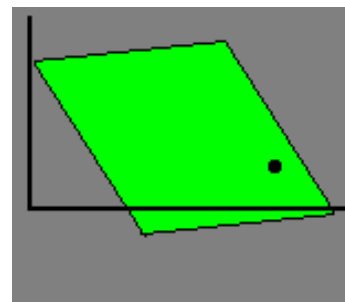
- 後ろ・回転層・前のパーツを順に仮イメージに描き、最後に実イメージに表示する
- 回転層の部分の表示には元のGridを回転させた仮のGridを使う
- 1ターンのアニメーションが終わったらピースのデータを書き換える



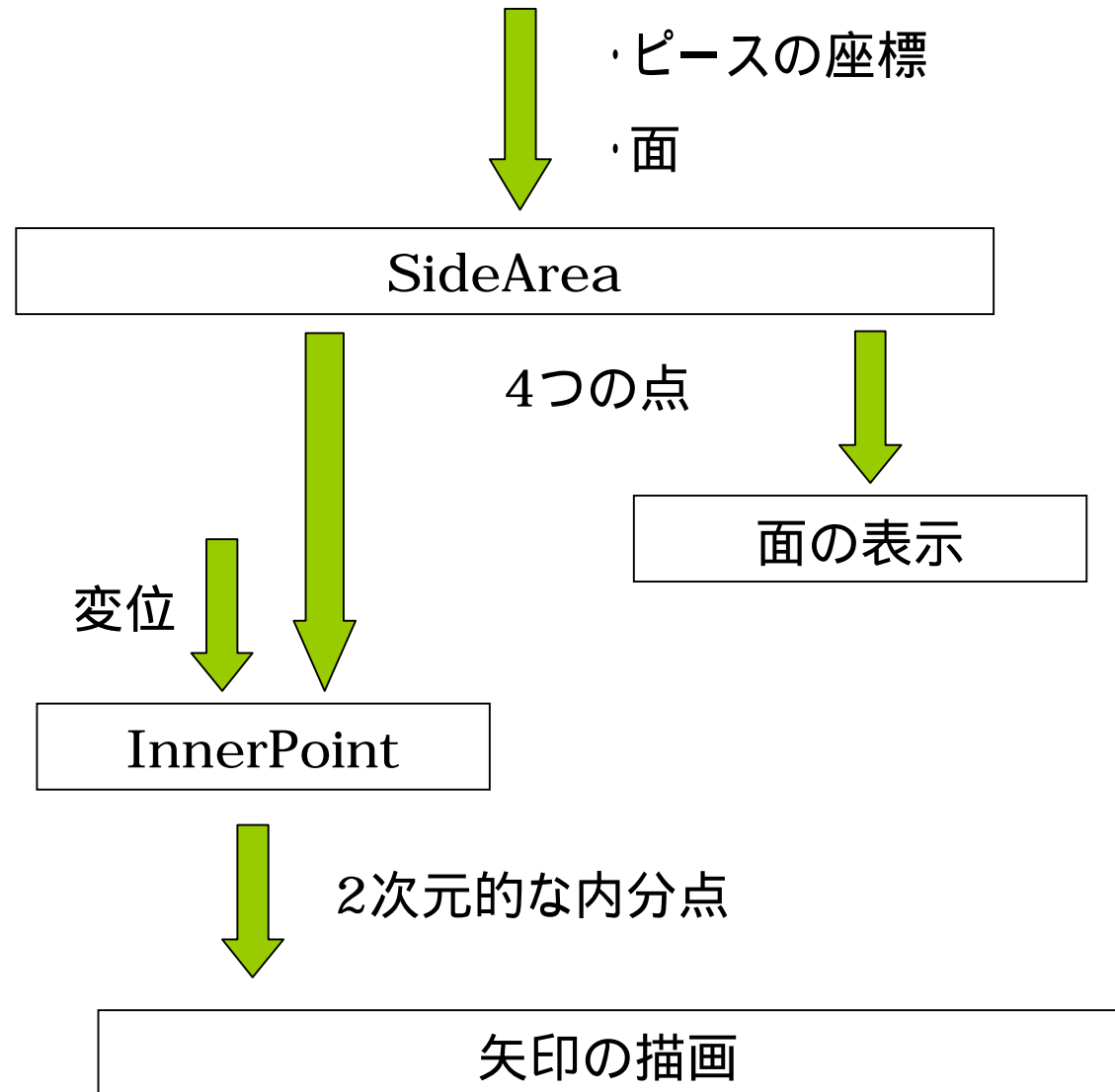
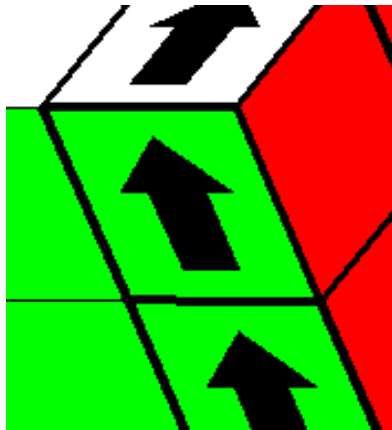
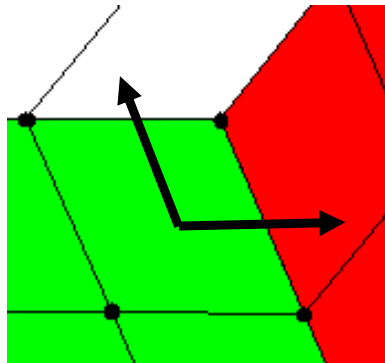
# 回転候補の取得



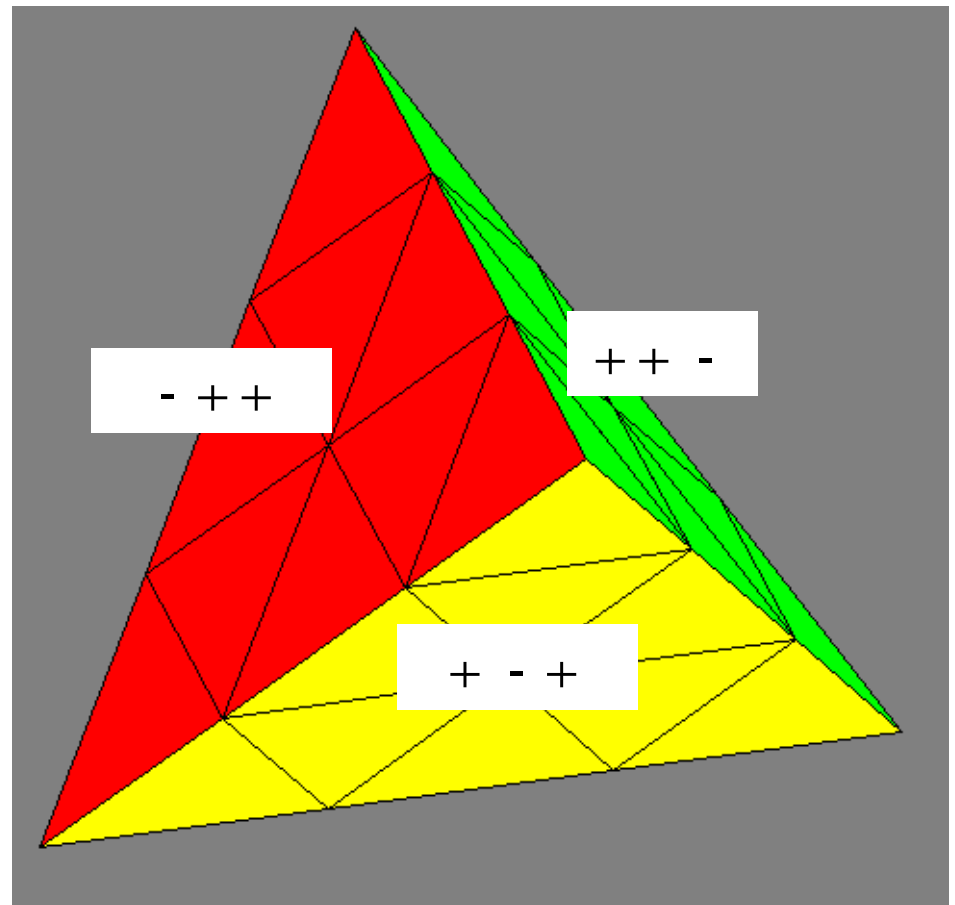
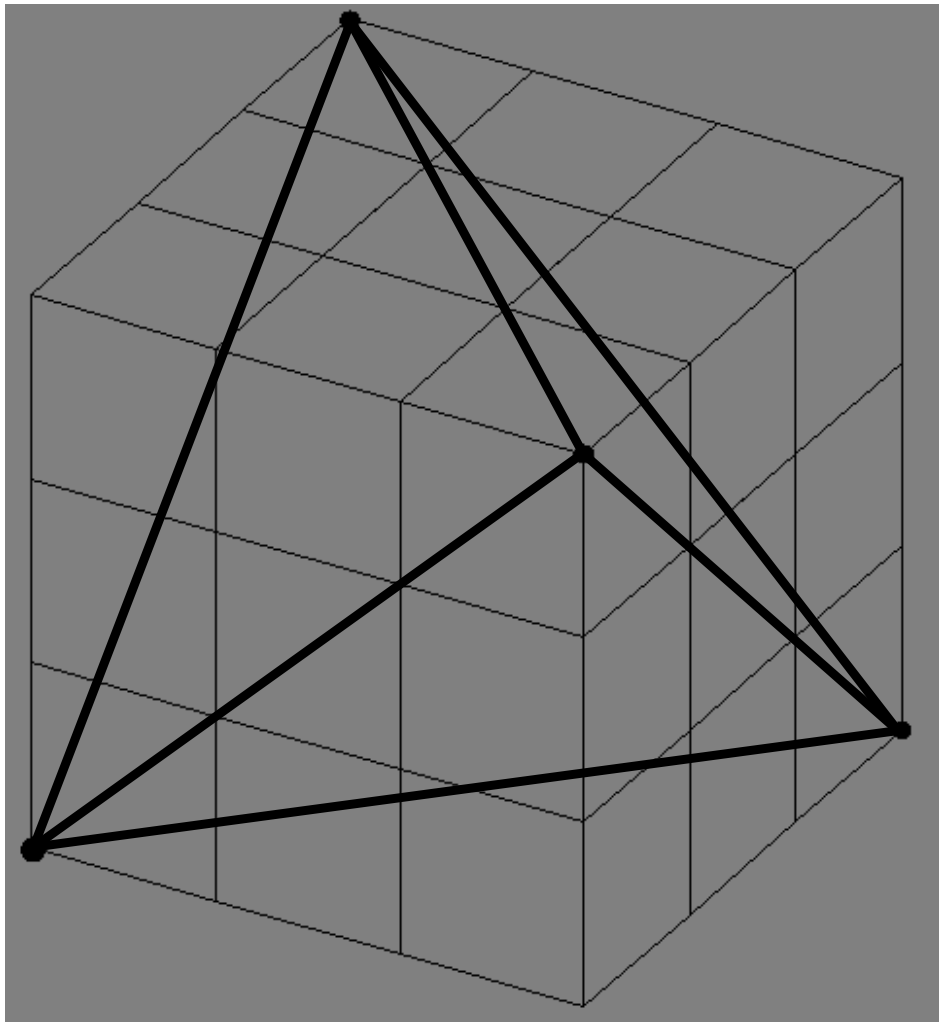
- マウスポインタがピース上にあるとき、4分割した領域のどこにあるかで回転候補を決める



# 矢印の表示

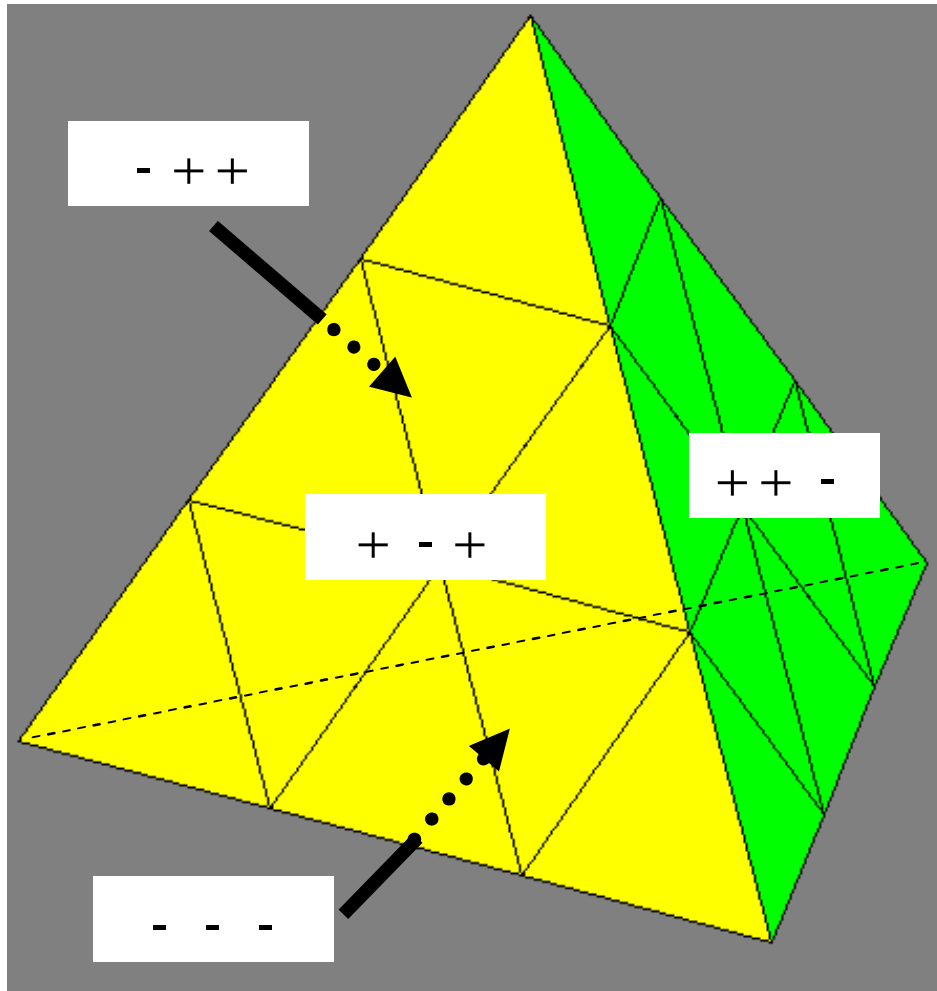


# 四面体パズル



六面体をカットすると四面体になる

# 四面体パズル



- グリッド・ピースの位置は4次元座標系で表す

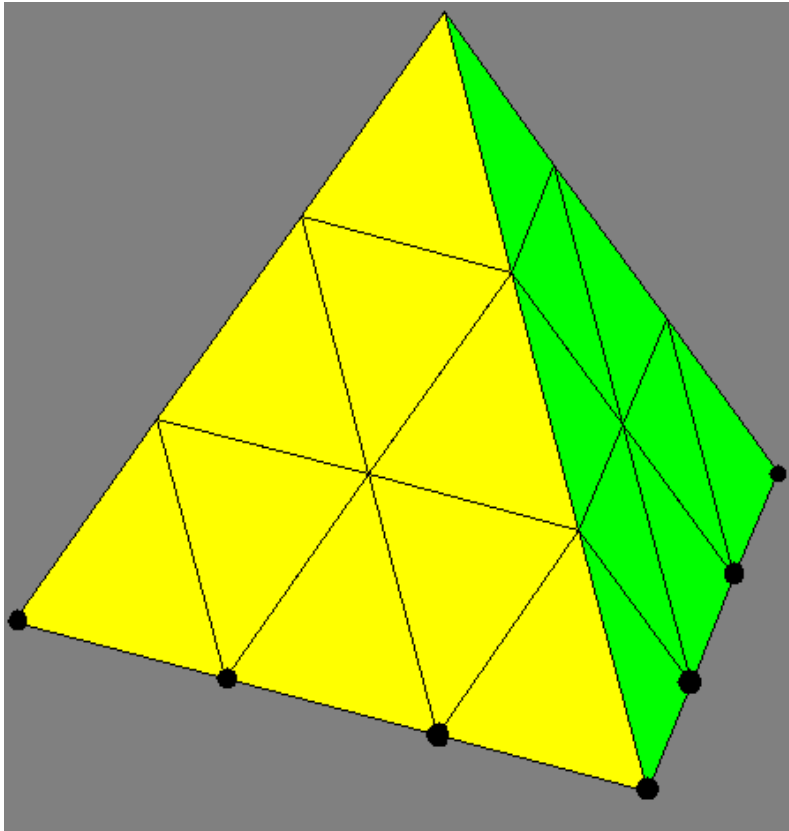
I軸： - - - 面

J軸： - + + 面

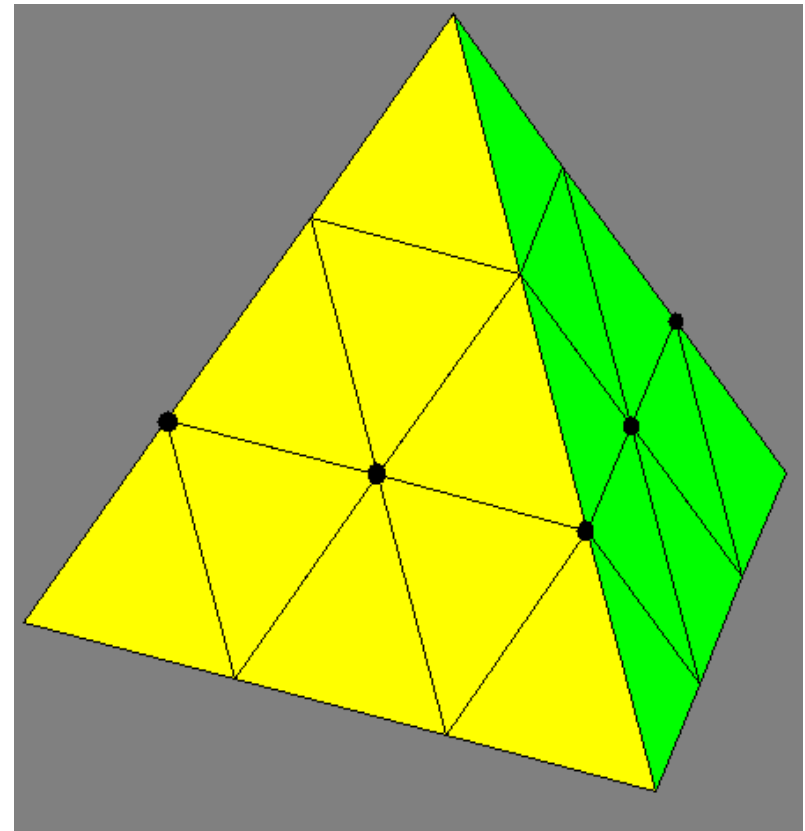
K軸： + - + 面

L軸： + + - 面

# グリッドの座標



$(0, j, k, l)$

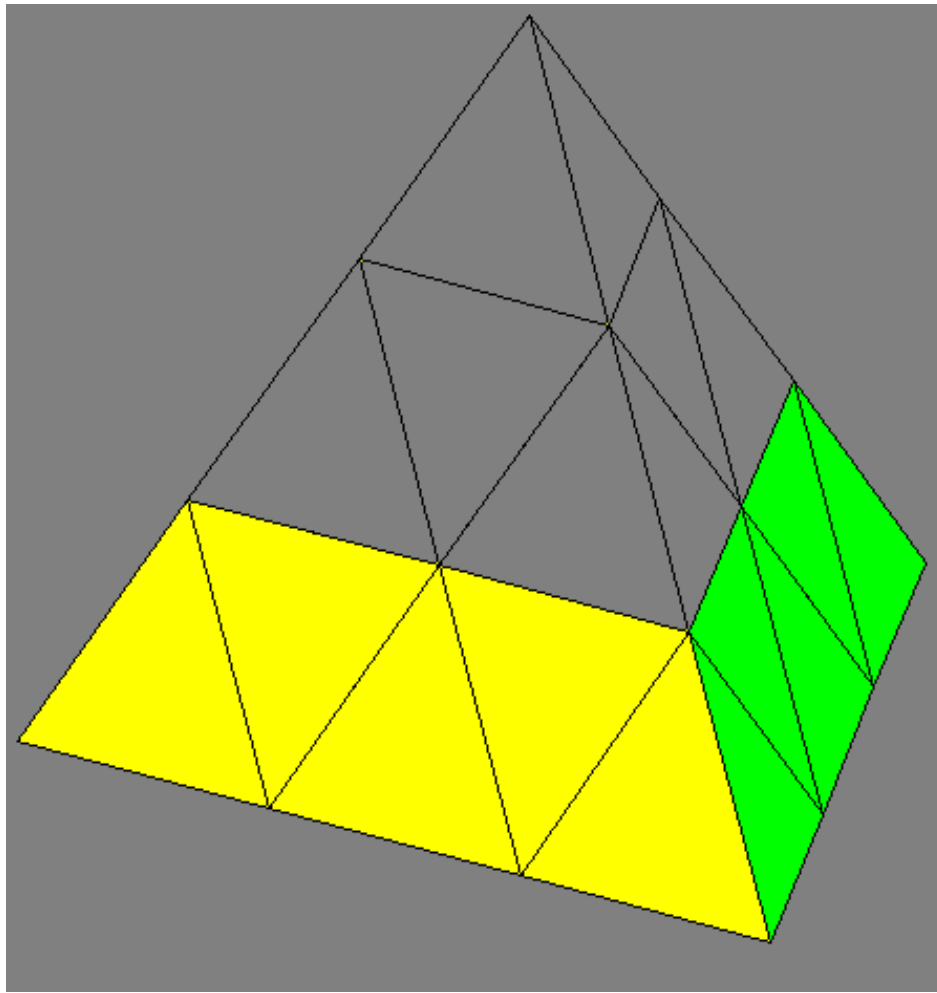


$(1, j, k, l)$

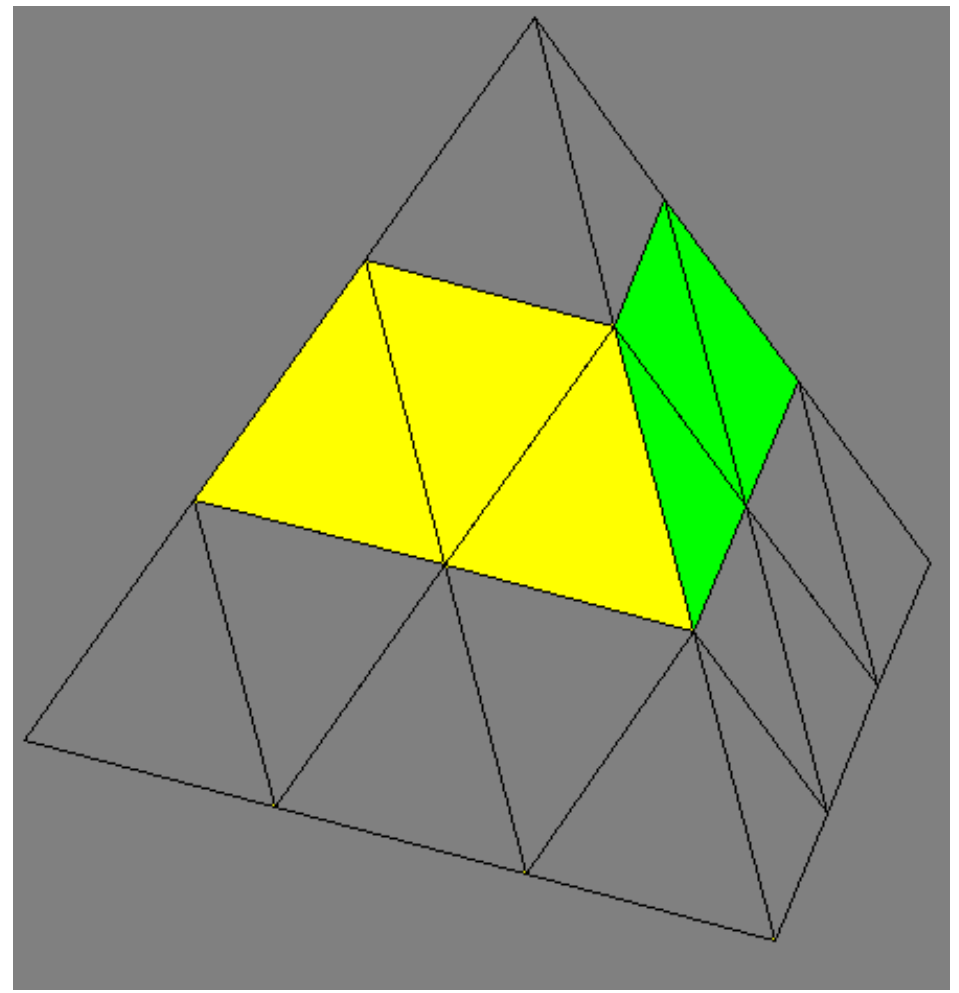
$$i + j + k + l = N$$

# ピースの座標

---



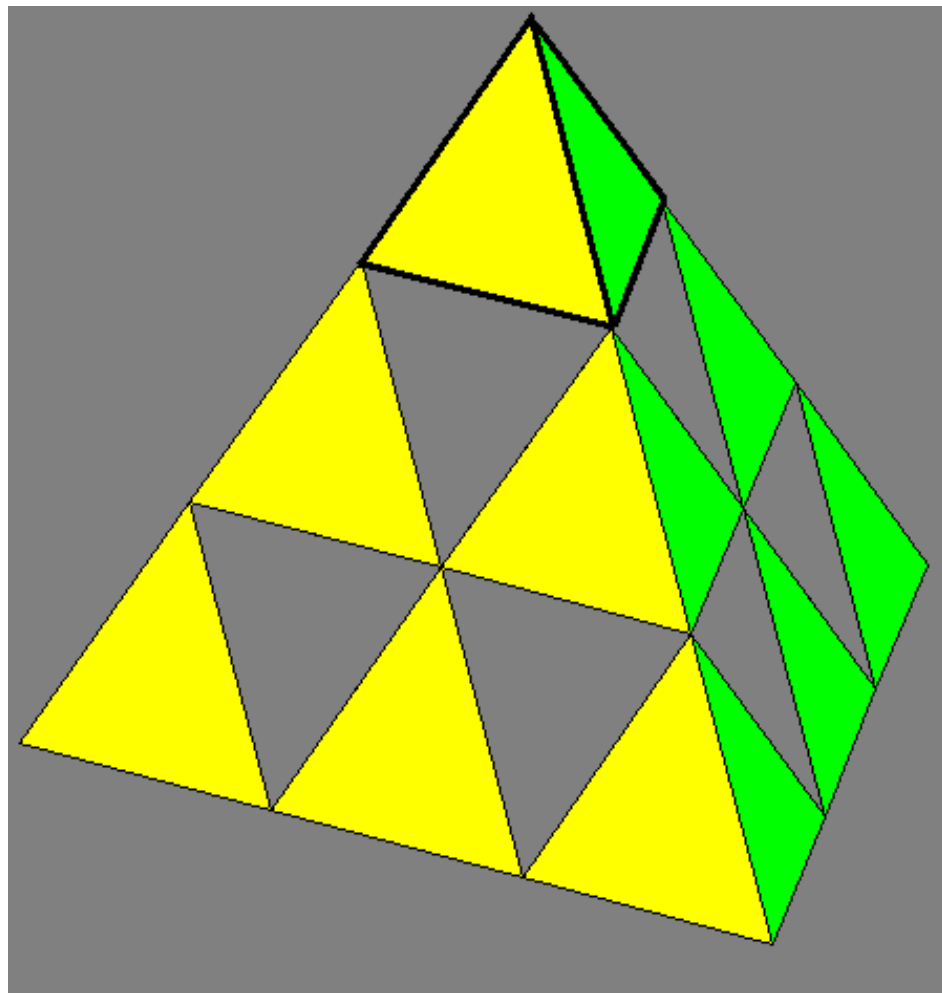
$(0, j, k, l)$



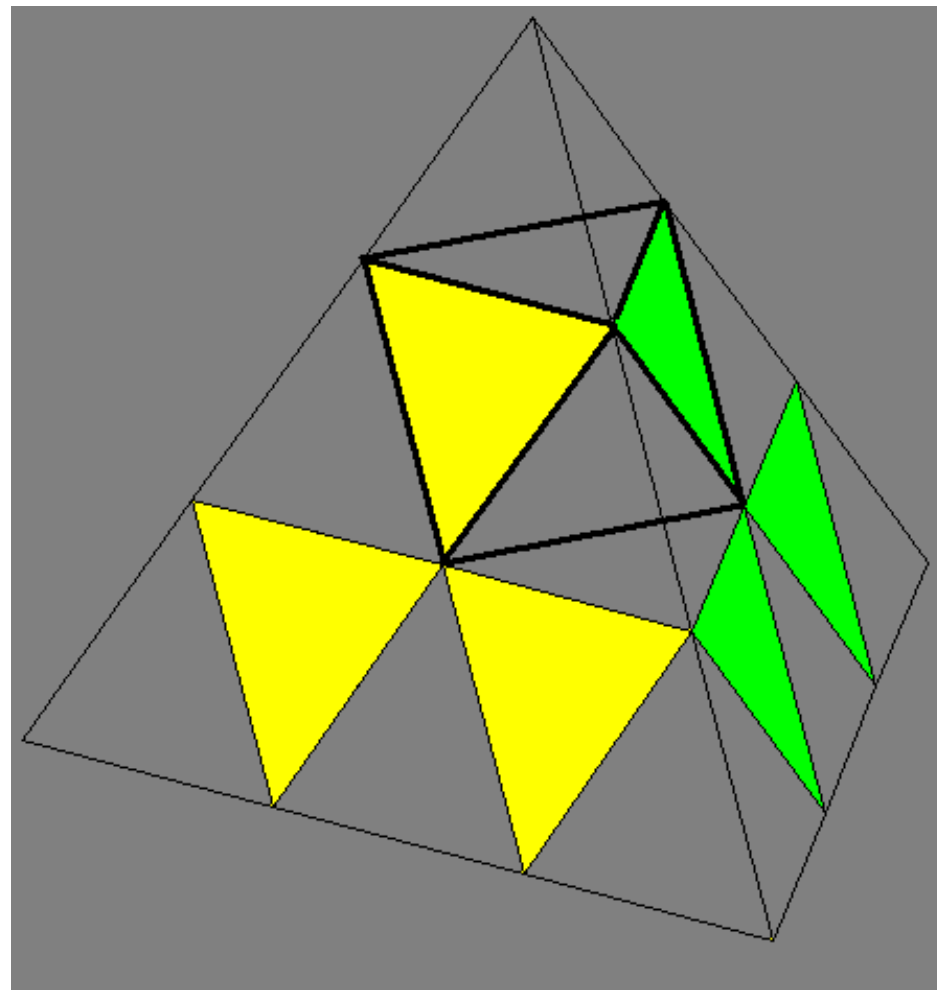
$(1, j, k, l)$

# 順ピースと逆ピース

---



$$i + j + k + l = N - 1$$



$$i + j + k + l = N - 2$$

# 方向データと回転

グローバル  
I J K L

ローカル  
i j k l

$$\begin{pmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & 1 \end{pmatrix}$$

- ピースの方向は4×4の行列で表される
- 120度単位で回転する

$$I_+ = \begin{pmatrix} 1 & & & \\ & & 1 & \\ & & & 1 \\ & 1 & & \end{pmatrix}$$

$$J_+ = \begin{pmatrix} & & & 1 \\ & 1 & & \\ & & & \\ 1 & & & \end{pmatrix}$$

$$K_+ = \begin{pmatrix} & 1 & & \\ & & & 1 \\ & & 1 & \\ 1 & & & \end{pmatrix}$$

$$L_+ = \begin{pmatrix} & & & 1 \\ & 1 & & \\ & & & \\ & & 1 & \end{pmatrix}$$

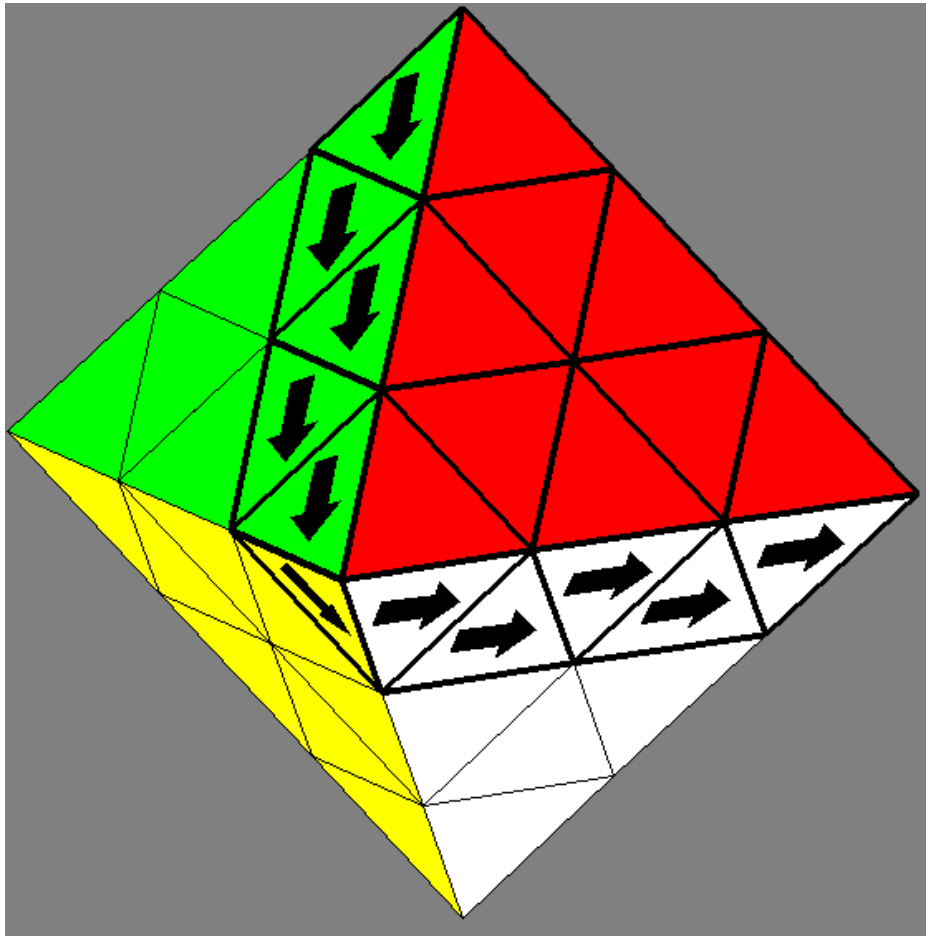
$$I_- = \begin{pmatrix} 1 & & & \\ & & 1 & \\ & & & 1 \\ & 1 & & \end{pmatrix}$$

$$J_- = \begin{pmatrix} & & & 1 \\ & 1 & & \\ & & & \\ 1 & & & \end{pmatrix}$$

$$K_- = \begin{pmatrix} & 1 & & \\ & & & 1 \\ & & 1 & \\ 1 & & & \end{pmatrix}$$

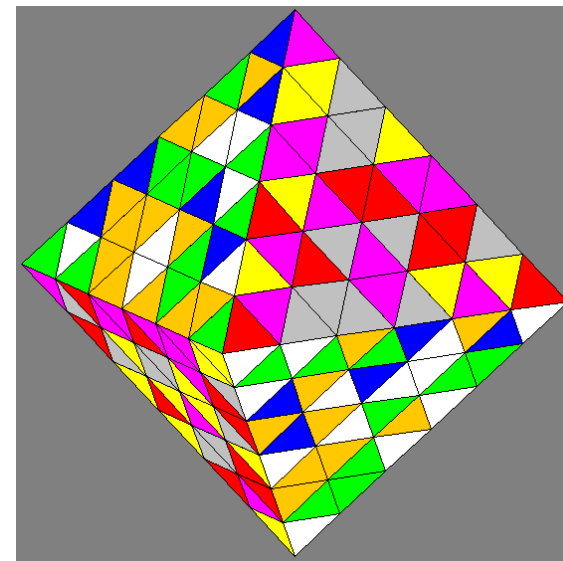
$$L_- = \begin{pmatrix} & & & 1 \\ & 1 & & \\ & & & 1 \\ & & 1 & \end{pmatrix}$$

# 八面体パズル(4軸)



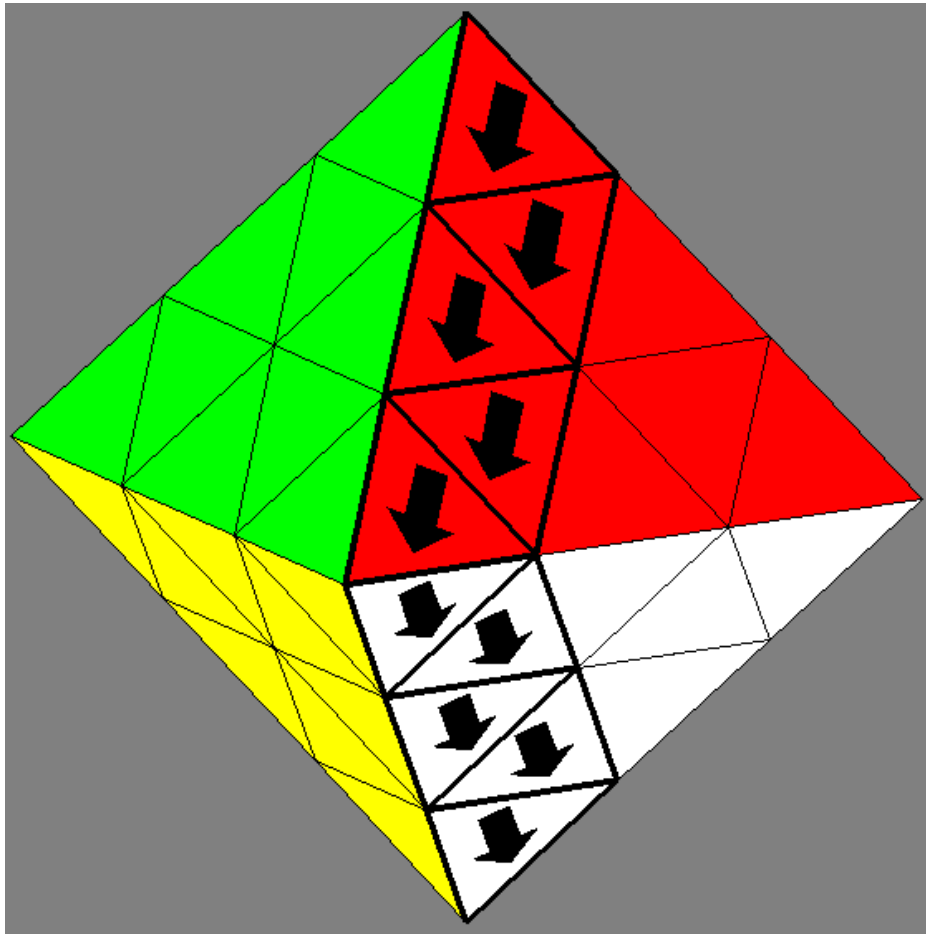
I軸まわり 正方向 第2層の回転

- 回転軸 : I, J, K, L
- 座標系: 4次元
- 単位回転角 :  $120^\circ$
- 方向行列 :  $4 \times 4$



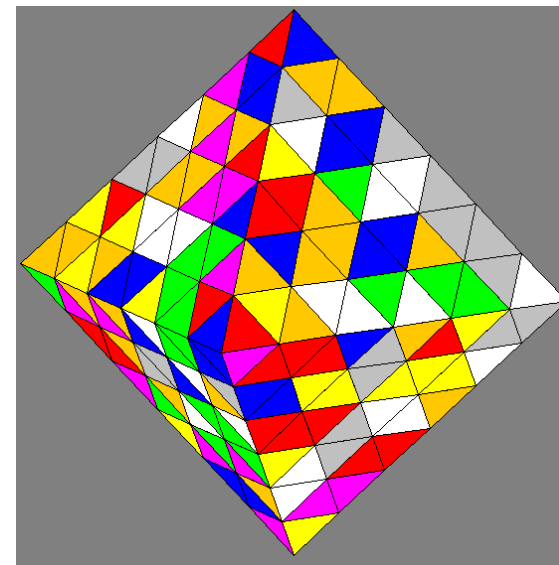
「表」と「裏」は混ざらない

# 八面体パズル(3軸)



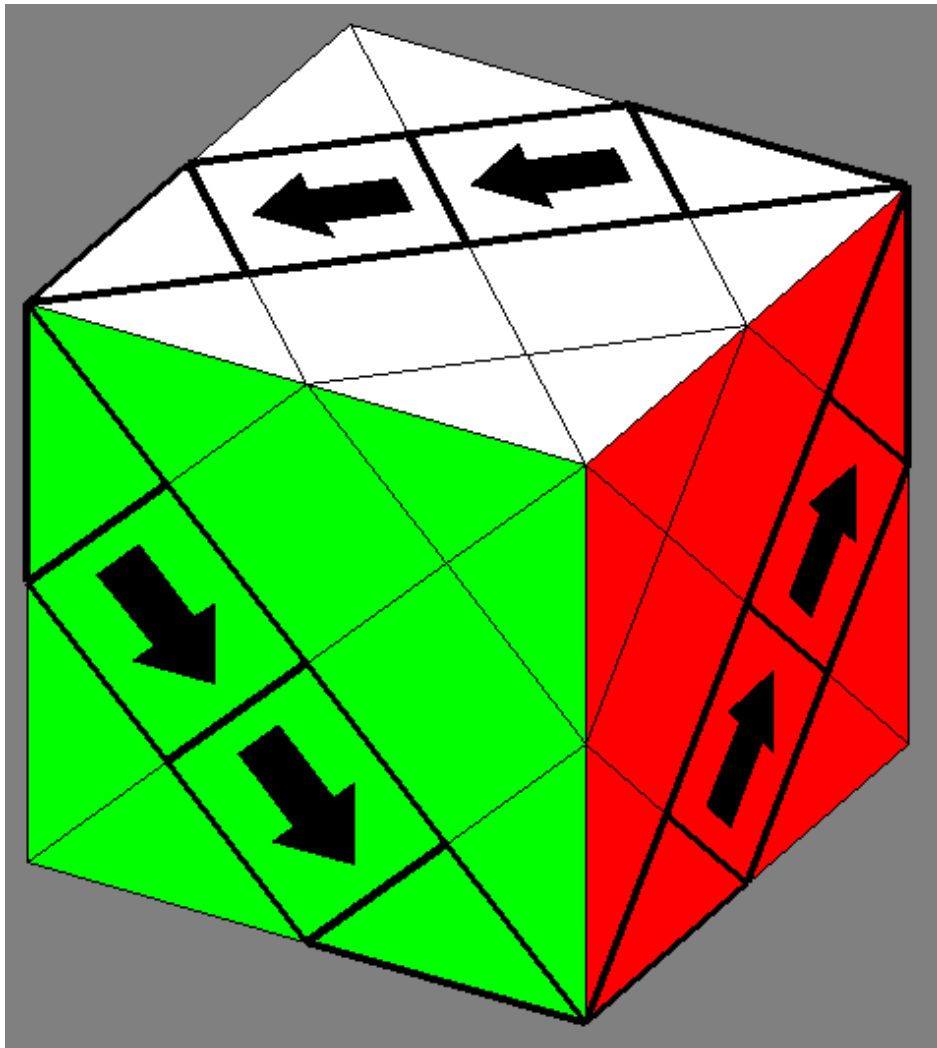
X軸まわり 正方向 第3層の回転

- 回転軸 : X, Y, Z
- 座標系: 3次元
- 単位回転角 :  $90^\circ$
- 方向行列 :  $4 \times 4$



ピースはどの面にも移動できる

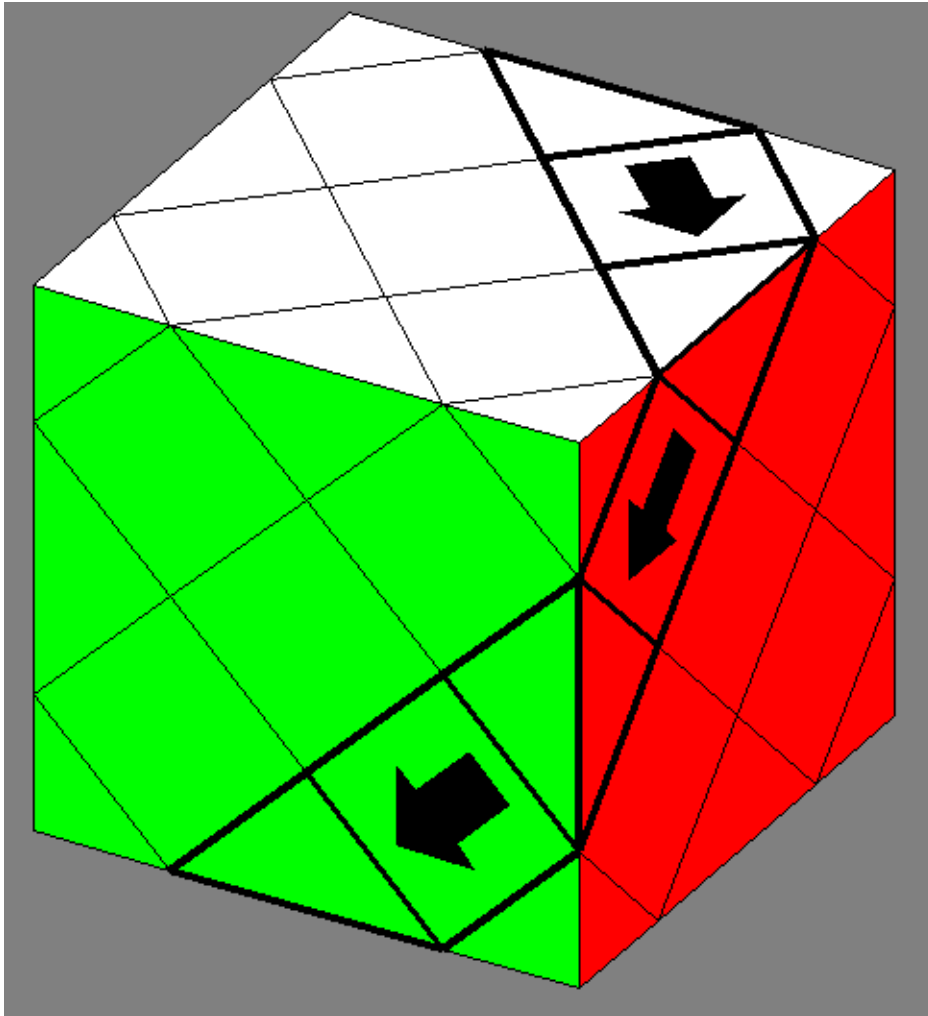
# 六面体・4軸のパズル(1)



I軸まわり 正方向 第3層の回転

- 回転軸: I, J, K, L
- 座標系: 3次元
- 単位回転角:  $120^\circ$
- 方向行列:  $3 \times 3$
  
- 層の切り方は等間隔

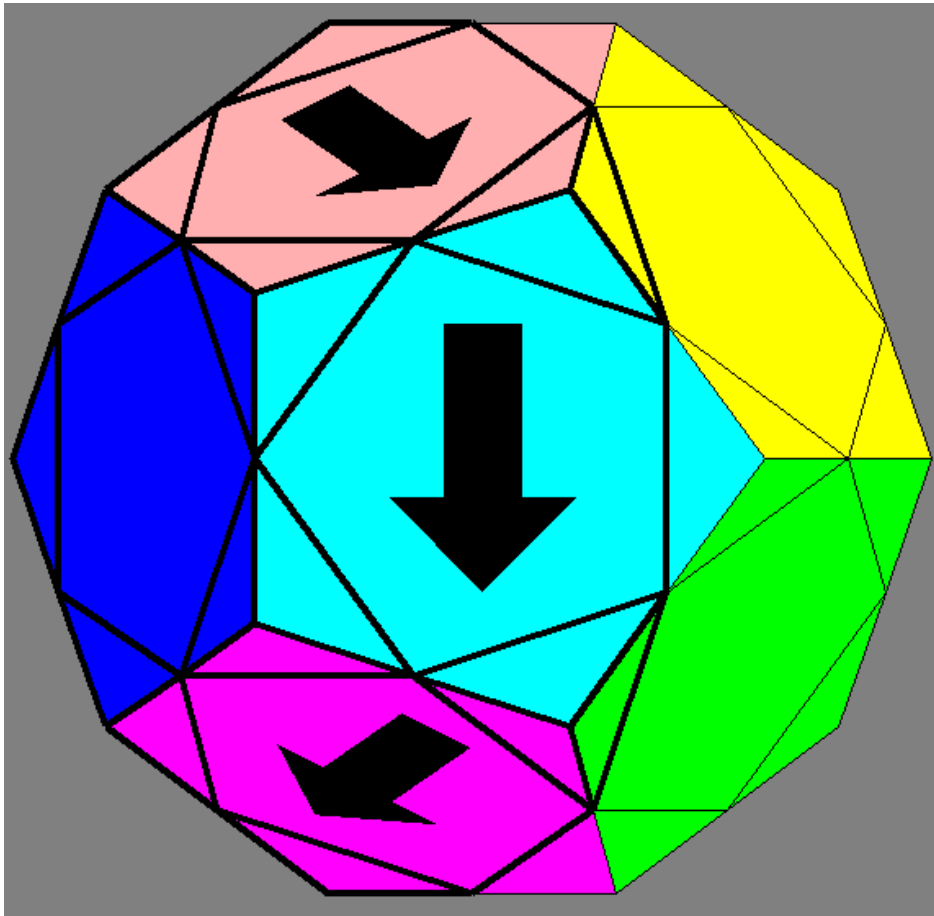
# 六面体・4軸のパズル(2)



J軸まわり 正方向 第3層の回転

- 回転軸: I, J, K, L
- 座標系: 3次元
- 単位回転角:  $120^\circ$
- 方向行列:  $3 \times 3$
  
- コーナーの部分の層の厚さはその他の半分

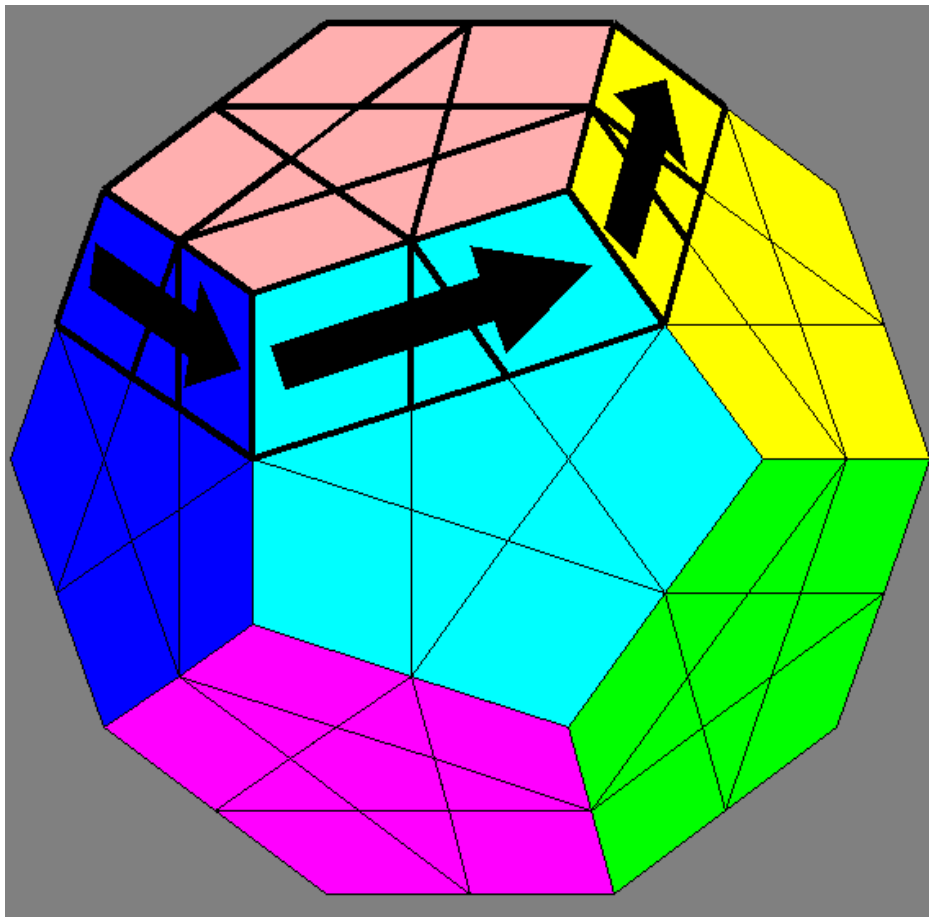
# 十二面体のパズル(1)



I軸まわり 正方向 第0層の回転

- 回転軸 : I, J, K, L, M, N
- 座標系 : 6次元
- 単位回転角 :  $72^\circ$
- 方向行列 :  $6 \times 6$
  
- 層数 : 2
  
- J, L軸 : XY平面上
- K, M軸 : YZ平面上
- I, N軸 : ZX平面上

# 十二面体のパズル(2)



- 回転軸 : I, J, K, L, M, N
- 座標系 : 6次元
- 単位回転角 :  $72^\circ$
- 方向行列 :  $6 \times 6$
  
- 層数 : 3

M軸まわり 正方向 第3層の回転

# まとめ

---

- 正六面体
  - 一辺のピース数が一般の場合に適用できる解法
  - 一辺のピース数が3の場合に特化したより手数の少ない解法
  - 一辺のピース数が3の場合の完全解法
- 正四面体
  - 一辺のピース数が2, 3の場合に付いての解法
- 正八面体・正十二面体
  - 再現のみ
- ユーザーフレンドリーな操作系
  - 回転候補のプレビュー
  - Undo, Redo機能
  - セーブ・ロード機能

# 今後の課題

---

- 群論的な解法
- $4 \times 4 \times 4$ 以上の完全解法
- テクスチャの使用 (Java 3D, Direct3D)
- 二十面体・準正多面体のパズル
- 現実のキューブを解くロボット