

# Rubik's 4-Cube

Takashi Yoshino

Rubik's cube has a natural extension to four-dimensional space. This article constructs the basic concepts of the puzzle and implements it in a program. The well-known three-dimensional Rubik's cube  $R_3$  consists of 27 unit subcubes. Each face  $F$  of  $R_3$  determines a set  $S$  of nine subcubes that have a face in the same plane as  $F$ . The set  $S$  can be rotated around the normal through the center of  $F$ . Rubik's 4-cube (or 4D hypercube)  $R_4$  consists of 81 unit 4-subcubes, each containing eight 3D subcubes. Each 3-face  $G$  of  $R_4$  determines a set  $T$  of 27 4-subcubes that have a cube in the same hyperplane as  $G$ . The set  $T$  can be rotated around the normal (a plane) through the center of  $G$ . Projecting the whole 4D configuration to 3D exhibits Rubik's 4-cube as a four-dimensional extension of Rubik's cube. Starting from a random coloring of the 4-cube, the goal of the puzzle is to return to the initial coloring of the 3-faces.

## ■ Basic Concepts

### □ Hypercubes in Low Dimensions

To understand the 4D hypercube, it helps to first see how its lower-dimensional analogs relate to each other. The zero-dimensional hypercube (or 0-cube) is a point, with one vertex. The 1D hypercube (or 1-cube) is a segment, with two vertices and one edge. The 2D hypercube (or 2-cube) is a square, with four vertices, four edges and one face (the square including its interior). The 3D hypercube is a cube (or 3-cube), with eight vertices, 12 edges, six square faces and one volume. Going up a dimension doubles the number of vertices. More generally, the number of  $d$ -cubes (points, segments, squares, ...) in an  $n$ -cube,  $0 \leq d \leq n$ , is  $2^{n-2} \binom{n}{d}$ .

```
Table[2^(n - d) Binomial[n, d], {n, 0, 3}, {d, 0, n}]
```

```
{{1}, {2, 1}, {4, 4, 1}, {8, 12, 6, 1}}
```

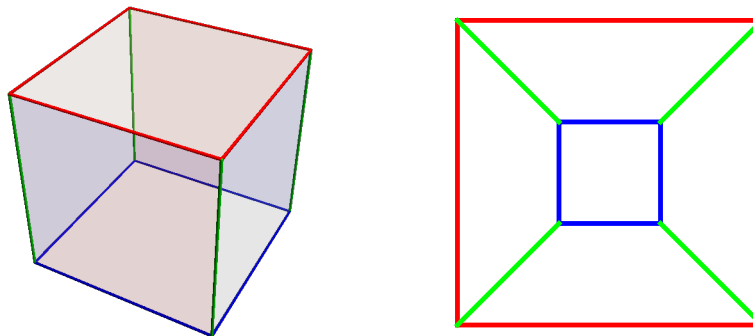
The 3D cube can be represented in a 2D plane using central projection, defined by taking the intersection of the plane  $z = 0$  with the line joining the two points  $(0, 0, a)$  and  $(x, y, z)$ . This projection maps the point  $(x, y, z)$  to  $\frac{a-z}{a}(x, y)$ . Choose  $a = -2$  to obtain the projection shown on the right in Figure 1. Five of the faces overlap with a sixth face, the price to pay for the loss of one dimension.

```

To2DFrom3D[{x_, y_, z_}] := (2 + z) / 2 {x, y}

Module[{vertices3D, faces3D},
  vertices3D = Tuples[{-1, 1}, 3];
  faces3D = {{1, 2, 4, 3}, {1, 2, 6, 5}, {1, 3, 7, 5},
    {2, 4, 8, 6}, {4, 3, 7, 8}, {5, 7, 8, 6}};
  Grid[
    {{Graphics3D[
      {GraphicsComplex[vertices3D,
        {Thick, Blue, Tube[{1, 3, 7, 5, 1}], Red,
          Tube[{2, 4, 8, 6, 2}], Green, Tube[{1, 2}],
          Tube[{3, 4}], Tube[{7, 8}], Tube[{5, 6}],
          White, Opacity[0.2], Polygon /@ faces3D}}],
      Boxed → False],
    Spacer[40],
    Graphics[GraphicsComplex[To2DFrom3D /@ vertices3D,
      {Thick, Blue, Line[{1, 3, 7, 5, 1}], Red,
        Line[{2, 4, 8, 6, 2}], Green, Line[{1, 2}],
        Line[{3, 4}], Line[{7, 8}], Line[{5, 6}]}}}]]}
]
]

```



▲ **Figure 1.** A cube and its image under a central projection.

## □ 4-Cube

Overall, the 4D Rubik puzzle is a 4-cube [1] (or 4D hypercube or tesseract), with 16 vertices, 32 edges, 24 squares, eight cubes and one 4-cube. The eight cubes are called *cells*, which are like the six square faces of a 3D cube. The proper faces of the 4-cube are its vertices, edges, squares and cells.

```
With[{n = 4}, Table[2^(n - d) Binomial[n, d], {d, 0, n - 1}]]

{16, 32, 24, 8}
```

Each point of a proper face is on the 3D hypersurface of the 4-cube. No point of a proper face is strictly in the interior of the 4-cube; that is, a hypersphere at such a point contains points inside and points outside the 4-cube. In particular, no interior point of a cell as a 3D object is in the interior of the hypercube; all the points of a cell are on the boundary of the 4-cube.

The 16 vertices of a 4-cube can be defined as lists of length four of all possible combinations of  $-1$  and  $1$ .

```
vertices = Tuples[{-1, 1}, 4]

{{-1, -1, -1, -1}, {-1, -1, -1, 1}, {-1, -1, 1, -1},
 {-1, -1, 1, 1}, {-1, 1, -1, -1}, {-1, 1, -1, 1},
 {-1, 1, 1, -1}, {-1, 1, 1, 1}, {1, -1, -1, -1},
 {1, -1, -1, 1}, {1, -1, 1, -1}, {1, -1, 1, 1},
 {1, 1, -1, -1}, {1, 1, -1, 1}, {1, 1, 1, -1}, {1, 1, 1, 1}}
```

The 24 squares of the 4-cube are described in terms of their vertex indices.

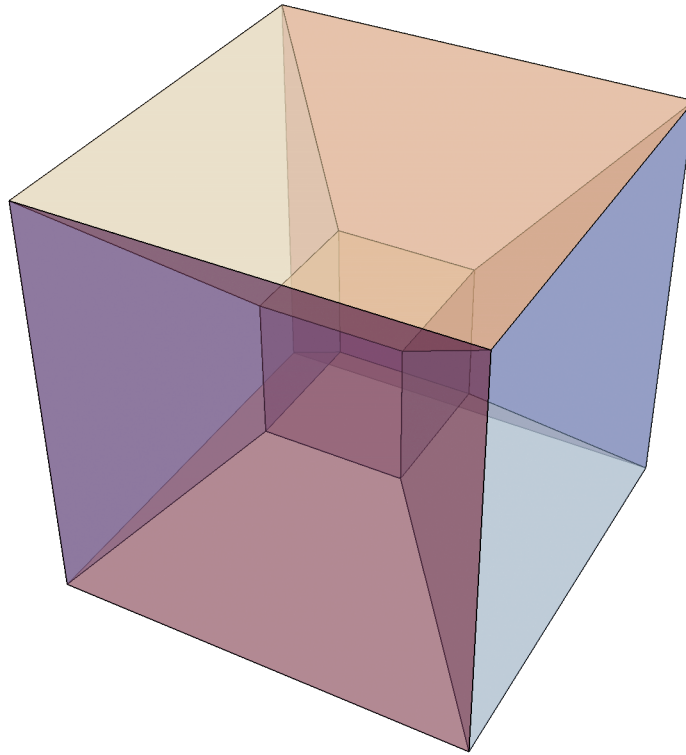
```
faces = {{1, 2, 4, 3}, {1, 2, 6, 5}, {1, 2, 10, 9},
 {1, 3, 7, 5}, {1, 3, 11, 9}, {1, 5, 13, 9}, {2, 4, 8, 6},
 {2, 4, 12, 10}, {2, 6, 14, 10}, {3, 4, 8, 7},
 {3, 4, 12, 11}, {3, 7, 15, 11}, {4, 8, 16, 12},
 {5, 6, 8, 7}, {5, 6, 14, 13}, {5, 7, 15, 13},
 {6, 8, 16, 14}, {7, 8, 16, 15}, {9, 10, 12, 11},
 {9, 10, 14, 13}, {9, 11, 15, 13}, {10, 12, 16, 14},
 {11, 12, 16, 15}, {13, 14, 16, 15}};
```

Besides the 4-cube, there are five other regular polytopes in four dimensions. The .csv and .m files containing information for these polytopes are provided at [2]: the positions of the vertices, vertex indices for the proper faces and which faces are neighbors.

To display the 4-cube in 3D, central projection from 4D to 3D is analogous to central projection from 3D to 2D; the function `To3DFrom4D` is the natural extension of `To2DFrom3D`; see Figure 2.

```
To3DFrom4D[{x_, y_, z_, w_}] := {x, y, z} ((w + 2)) / 2
```

```
Graphics3D[
  {Opacity[0.5], GraphicsComplex[To3DFrom4D /@ vertices,
    Polygon /@ faces]}, Boxed → False]
```



▲ **Figure 2.** Projected image of a 4-cube by means of center projection. The larger outer cube is one of the cells of the 4-cube.

## □ Rotation

An axis of rotation in 3D is a fixed line. In 4D, an axis of rotation in four dimensions is a fixed plane [3]. For example, the rotation matrix about the  $x$ - $y$  plane  $R_{xy}(\theta)$  is defined by:

$$R_{xy}(\theta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \cos \theta & -\sin \theta \\ 0 & 0 & \sin \theta & \cos \theta \end{pmatrix}. \quad (1)$$

There are six planes of rotation spanned by pairs of coordinate axes, namely  $x$ - $y$ ,  $x$ - $z$ ,  $x$ - $w$ ,  $y$ - $z$ ,  $y$ - $w$ ,  $z$ - $w$ .

```
rotateXY[θ_] := {{1, 0, 0, 0}, {0, 1, 0, 0},
  {0, 0, Cos[θ], -Sin[θ]}, {0, 0, Sin[θ], Cos[θ]}};
rotateXZ[θ_] := {{1, 0, 0, 0}, {0, Cos[θ], 0, -Sin[θ]},
  {0, 0, 1, 0}, {0, Sin[θ], 0, Cos[θ]}};
rotateXW[θ_] := {{1, 0, 0, 0}, {0, Cos[θ], -Sin[θ], 0},
  {0, Sin[θ], Cos[θ], 0}, {0, 0, 0, 1}};
rotateYZ[θ_] := {{Cos[θ], 0, 0, -Sin[θ]}, {0, 1, 0, 0},
  {0, 0, 1, 0}, {Sin[θ], 0, 0, Cos[θ]}};
rotateYW[θ_] := {{Cos[θ], 0, -Sin[θ], 0}, {0, 1, 0, 0},
  {Sin[θ], 0, Cos[θ], 0}, {0, 0, 0, 1}};
rotateZW[θ_] := {{Cos[θ], -Sin[θ], 0, 0},
  {Sin[θ], Cos[θ], 0, 0}, {0, 0, 1, 0}, {0, 0, 0, 1}};
```

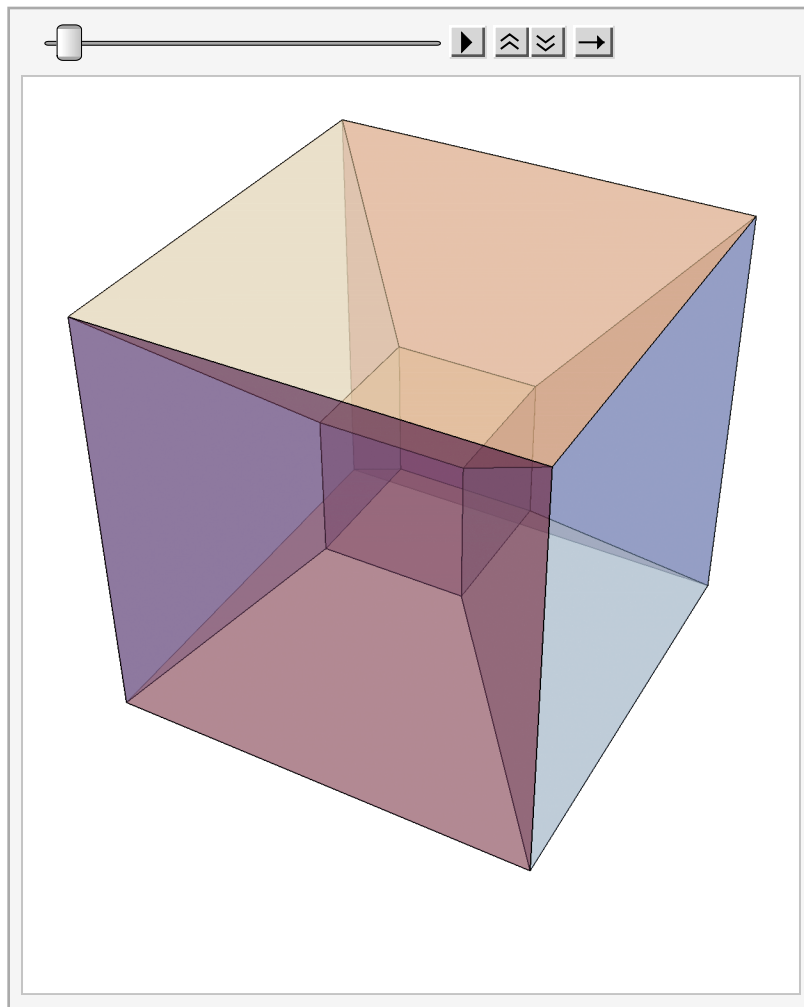
Here is the first one, for example, which leaves points in the  $x$ - $y$  plane fixed.

```
rotateXY[θ] // MatrixForm
```

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \cos[\theta] & -\sin[\theta] \\ 0 & 0 & \sin[\theta] & \cos[\theta] \end{pmatrix}$$

This animation shows two successive rotations of the 4-cube projected to 3D.

```
ListAnimate[
  With[{n = 50},
    Join[
      Table[
        Graphics3D[{Opacity[0.5],
          GraphicsComplex[
            To3DFrom4D /@ (rotateXY[θ].# & /@vertices),
            Polygon /@ faces]], Boxed → False],
        {θ, 0, Pi - 2 Pi / n, 2 Pi / n}],
      Table[
        Graphics3D[{Opacity[0.5],
          GraphicsComplex[
            To3DFrom4D /@ (rotateYZ[θ].# & /@vertices),
            Polygon /@ faces]], Boxed → False],
        {θ, 0, Pi - 2 Pi / n, 2 Pi / n}]]
  ], AnimationRunning → False
]
```



## ■ Implementing Rubik's 4-Cube

### □ Dividing the 4-Cube

Consider a  $3 \times 3 \times 3 \times 3$  4-cube with center at the origin  $(0, 0, 0, 0)$ , side length 3, and with all proper faces of positive dimension parallel to the coordinate axes. Then its 16 vertices are:

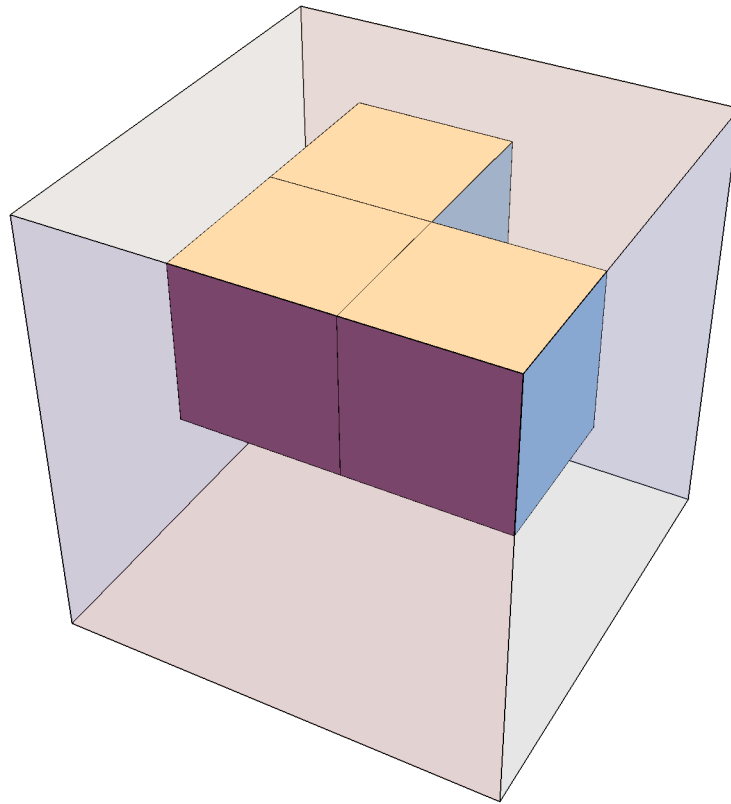
$$\left( \pm \frac{3}{2}, \pm \frac{3}{2}, \pm \frac{3}{2}, \pm \frac{3}{2} \right). \quad (2)$$

The eight cells of the initial 4-cube are colored differently. The word “initial” means that no rotations have been applied. The coloring touches every point of a cell, including its 3D interior points.

Just as the faces of Rubik's cube  $R_3$  are divided into nine squares by dividing each edge into three, the edges of Rubik's 4-cube  $R_4$  are also divided into three. Then the initial 4-cube is divided into  $81 = 3^4$  small 4-cubes, each with edge length 1. The boundary (a hypersurface) of a small 4-cube contains eight small cubes, its cells.

The Rubik 3-cube has 27 subcubes in  $R_3$ ; no square of the center cube is colored and some squares of other cubes are colored. These 26 subcubes are classified into three types according to whether they are at a corner, at an edge or at the center of a face of the larger cube. Figure 3 shows one of each type.

```
Module[{cube = Cuboid[{-1, -1, -1}, {1, 1, 1}]},
Graphics3D[{{Opacity[.2], Scale[cube, 1.001]},
  Translate[Scale[cube, 1/3], {0, 0, 2/3}],
  Translate[Scale[cube, 1/3], {0, -2/3, 2/3}],
  Translate[Scale[cube, 1/3], {2/3, -2/3, 2/3}]},
Boxed → False]
```



▲ **Figure 3.** Three types of small cubes: in the center of a square face, at an edge and at a vertex, with one, two or three colored squares.

Analogously, the 81 small 4-cubes of  $R_4$  include the uncolored one at the center and 80 partially colored small 4-cubes. These are classified into four types according to the dimension of their intersection with  $R_4$ . The type of a small 4-cube does not change after rotation. Table 1 summarizes the numbers for each type for  $R_3$  and  $R_4$ .

type	Rubik's Cube		Rubik's 4-Cube	
	colored small squares per small cube	small cubes	colored small cells per small 4-cube	small 4-cubes
cell	–	–	1	8
face	1	6	2	24
edge	2	12	3	32
vertex	3	8	4	16
total	–	26	–	80

▲ **Table 1.** Numbers of colored small squares for  $R_3$  and small cubes for  $R_4$  for each type of small cube or cell.

The number of colored small squares for Rubik's cube is calculated using the data in Table 1:

$$6 \times 1 + 12 \times 2 + 8 \times 3 = 54.$$

Another way is to count the number of faces times the number of squares per face:  $6 \times 9 = 54$ .

The small 4-cubes with nonzero coordinates form the hypersurface of  $R_4$ . In particular, a small 4-cube with center given by four nonzero coordinates contains a vertex of  $R_4$ . Again from Table 1, the number of colored small cells is:

$$8 \times 1 + 24 \times 2 + 32 \times 3 + 16 \times 4 = 216.$$

This number can also be obtained as the number of cells of  $R_4$  times the number of small cells per cell of  $R_4$ :  $8 \times 27 = 216$ .

We define several global variables to be used here and later. Figure 4 shows the divided 4-cube with 216 colored small cells.

```

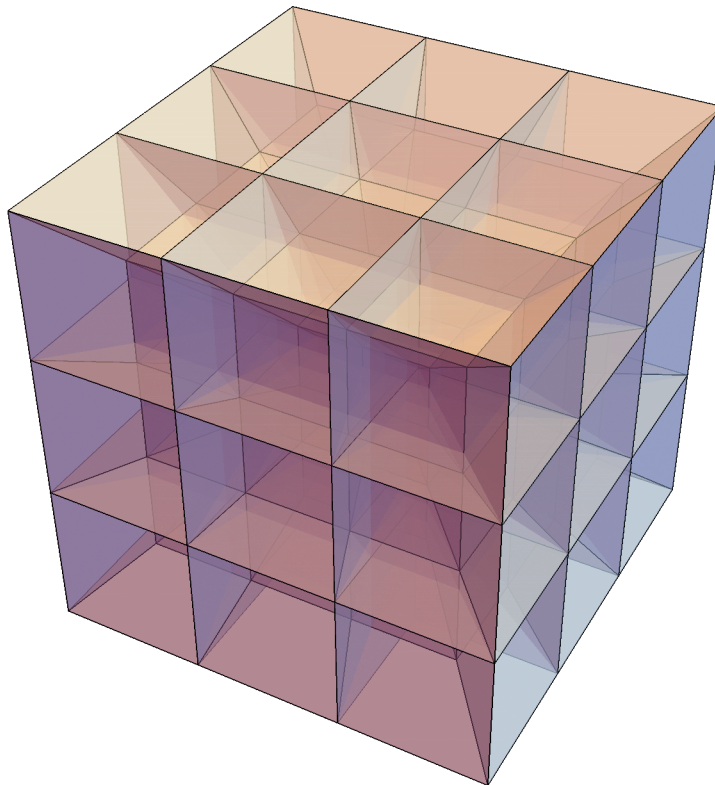
centersOfAllSmall4Cubes =
  Flatten[Table[{i, j, k, 1}, {i, -1, 1}, {j, -1, 1},
    {k, -1, 1}, {1, -1, 1}], 3];

verticesOfAllSmall4Cubes =
  Table[(centersOfAllSmall4Cubes[[i]] + #) & /@ (vertices / 2),
    {i, 81}];

```



```
Graphics3D[
  {Opacity[0.5],
   Table[GraphicsComplex[
     To3DFrom4D /@ verticesOfAllSmall4Cubes[[i]],
     Polygon /@ faces],
    {i, Length[verticesOfAllSmall4Cubes]}]], Boxed -> False]
```



▲ **Figure 4.** Center projection of a hypercube consisting of 216 colored small cells.

Each edge is divided into three parts, so that the length of the 4-subcubes is 1. Consider a 4-subcube with center at  $(p, q, r, s)$ . Then, the vertices of the 4-subcube are  $(p \pm 1/2, q \pm 1/2, r \pm 1/2, s \pm 1/2)$ . The coordinates of the center of each 4-subcube are a combination of one of  $-1, 0$  and  $+1$ . When the value is nonzero ( $-1$  or  $+1$ ), the 4-subcubes face outward in the corresponding directions. In other words, the nonzero values in coordinates denote the outward-facing 4-subcubes.

## □ The 216 Colored Small Cells and Initial State

For the initial state, the colors of the cells are set according to the coordinates of their centers:

center coordinate →	-1	1
$x$	pink	purple
$y$	orange	red
$z$	green	blue
$w$	white	yellow

For example, in the small 4-cube with center  $(1, 0, -1, 0)$ , the two small cells with vertices  $(3/2, \pm 1/2, -1 \pm 1/2, \pm 1/2)$  and  $(1 \pm 1/2, \pm 1/2, -3/2, \pm 1/2)$  are colored because both the  $x$  and  $z$  coordinate values are nonzero.

```
colors = {Pink, Purple, Orange, Red, Green, Blue, White,  
Yellow};
```

The geometry of the 216 small colored cells is used to manage the puzzle. Each element of the datasets consists of four elements: (1) the vertices of six squares; (2) the location of the center of the small 4-cube to which the small cell belongs; (3) color; and (4) the location of the center of the small cell. The vertices of the six squares are used for drawing the subcubes, and the locations of the centers of the subcubes are used to judge the completeness of the puzzle. The dataset of the initial state is obtained by the following procedures. First, the vertex numbers of the squares making up each small cell are defined.

```
squaresOf4CubeCells = {  
  {1, 2, 4, 3}, {1, 2, 6, 5}, {1, 3, 7, 5}, {2, 4, 8, 6},  
  {3, 4, 8, 7}, {5, 6, 8, 7}},  
  {9, 10, 12, 11}, {9, 10, 14, 13}, {9, 11, 15, 13},  
  {10, 12, 16, 14}, {11, 12, 16, 15}, {13, 14, 16, 15}},  
  {1, 2, 4, 3}, {1, 2, 10, 9}, {1, 3, 11, 9}, {2, 4, 12, 10},  
  {3, 4, 12, 11}, {9, 10, 12, 11}},  
  {5, 6, 8, 7}, {5, 6, 14, 13}, {5, 7, 15, 13},  
  {6, 8, 16, 14}, {7, 8, 16, 15}, {13, 14, 16, 15}},  
  {1, 2, 6, 5}, {1, 2, 10, 9}, {1, 5, 13, 9}, {2, 6, 14, 10},  
  {5, 6, 14, 13}, {9, 10, 14, 13}},  
  {3, 4, 8, 7}, {3, 4, 12, 11}, {3, 7, 15, 11},  
  {4, 8, 16, 12}, {7, 8, 16, 15}, {11, 12, 16, 15}},  
  {1, 3, 7, 5}, {1, 3, 11, 9}, {1, 5, 13, 9}, {3, 7, 15, 11},  
  {5, 7, 15, 13}, {9, 11, 15, 13}},  
  {2, 4, 8, 6}, {2, 4, 12, 10}, {2, 6, 14, 10},  
  {4, 8, 16, 12}, {6, 8, 16, 14}, {10, 12, 16, 14}}  
};
```

Next, the 216 small cells are selected by checking all possible  $81 \times 8 = 648$  small cells.

**$81 \times 8$**

648

```

ColoredSmallCells =
Module[{surCubes},
surCubes = {};
Do[
If[centersOfAllSmall4Cubes[[j, i]] == 2 k + 1,
surCubes = Append[surCubes ,
{Map[verticesOfAllSmall4Cubes[[j]][[#]] &,
squaresOf4CubeCells[[2 i + k]], {2}],
centersOfAllSmall4Cubes[[j]], colors[[2 i + k]],
Mean[Flatten[Map[verticesOfAllSmall4Cubes[[j]][[#]] &,
squaresOf4CubeCells[[2 i + k]], {2}], 1]]]],
{k, -1, 0}, {i, 4}, {j, 81}];
surCubes
];

```

This list contains 216 entries and each entry contains four components corresponding to a small cell.


```
Dimensions[ColoredSmallCells]
```

```
{216, 4}
```

For example, here is entry 123 of ColoredSmallCells. The components for this small cell are its six square faces, center, color and current position.

```
ColoredSmallCells[[123]]
```

```

{{{
{{3/2, -1/2, -1/2, 1/2}, {3/2, -1/2, -1/2, 3/2},
{3/2, -1/2, 1/2, 3/2}, {3/2, -1/2, 1/2, 1/2}},
{{3/2, -1/2, -1/2, 3/2}, {3/2, 1/2, -1/2, 3/2}, {3/2, 1/2, -1/2, 1/2}},
{{3/2, -1/2, -1/2, 1/2}, {3/2, -1/2, 1/2, 1/2}, {3/2, 1/2, 1/2, 1/2},
{3/2, 1/2, -1/2, 1/2}},
{{3/2, -1/2, -1/2, 3/2}, {3/2, -1/2, -1/2, 1/2}, {3/2, -1/2, 1/2, 3/2},
{3/2, 1/2, 1/2, 3/2}},
{{3/2, 1/2, -1/2, 3/2}, {3/2, 1/2, -1/2, 1/2}, {3/2, 1/2, 1/2, 1/2},
{3/2, -1/2, 1/2, 3/2}},
{{3/2, 1/2, 1/2, 3/2}, {3/2, 1/2, 1/2, 1/2}, {3/2, 1/2, -1/2, 1/2},
{3/2, -1/2, -1/2, 3/2}},
{{3/2, 1/2, -1/2, 1/2}, {3/2, 1/2, -1/2, 3/2}, {3/2, 1/2, 1/2, 3/2},
{3/2, -1/2, 1/2, 1/2}},
{{3/2, 1/2, 1/2, 1/2}}, {1, 0, 0, 1}, , {3/2, 0, 0, 1}}

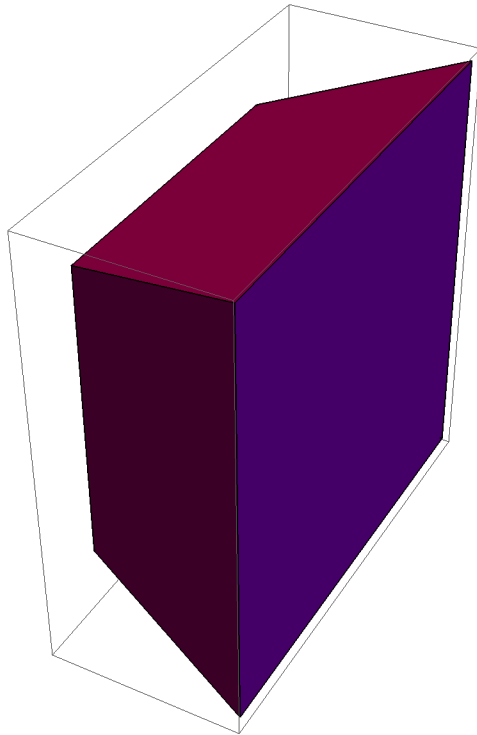
```

draw4Cube sets up a 4-cube for drawing.

```
draw4Cube[{vertices_, _, color_, _}] :=  
Prepend[Map[Polygon, Map[To3DFrom4D, vertices, {2}]], color]
```

Here is an example.

```
Graphics3D[draw4Cube@ColoredSmallCells[[123]]]
```



drawCell sets up a cell (with 27 4-cubes) for drawing.

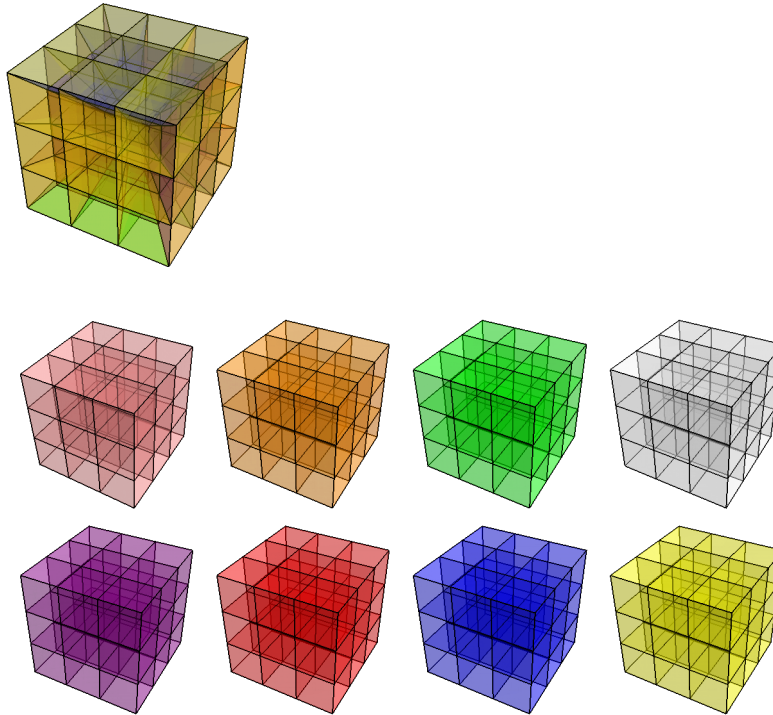
```
drawCell[sc_, xwzw_, pm_] :=  
Prepend[  
  Map[Polygon, Map[Delete[#, xwzw] &, #[[1]], {2}]],  
  #[[3]] ] & /@ Select[sc, #[[4, xwzw]] == pm 3 / 2 &  
]  
  
Length@drawCell[ColoredSmallCells, 1, -1]
```

27

Figure 5 shows the initial state of Rubik's 4-cube.

```
Figure5Aux[graphics3D_] :=
Graphics3D[{Opacity[0.3], graphics3D}, Boxed → False,
  Lighting → "Neutral"]

Row[{
  Graphics3D[{Opacity[0.3], draw4Cube /@ ColoredSmallCells},
    Boxed → False, ImageSize → 160, Lighting → "Neutral"],
  Spacer[40],
  Grid[{
    {
      Figure5Aux@drawCell[ColoredSmallCells, 1, -1],
      Figure5Aux@drawCell[ColoredSmallCells, 2, -1],
      Figure5Aux@drawCell[ColoredSmallCells, 3, -1],
      Figure5Aux@drawCell[ColoredSmallCells, 4, -1]
    },
    {
      Figure5Aux@drawCell[ColoredSmallCells, 1, 1],
      Figure5Aux@drawCell[ColoredSmallCells, 2, 1],
      Figure5Aux@drawCell[ColoredSmallCells, 3, 1],
      Figure5Aux@drawCell[ColoredSmallCells, 4, 1]
    }
  }]
}]
```



▲ **Figure 5.** Center projection of initial state of the Rubik 4-cube with its eight cells.

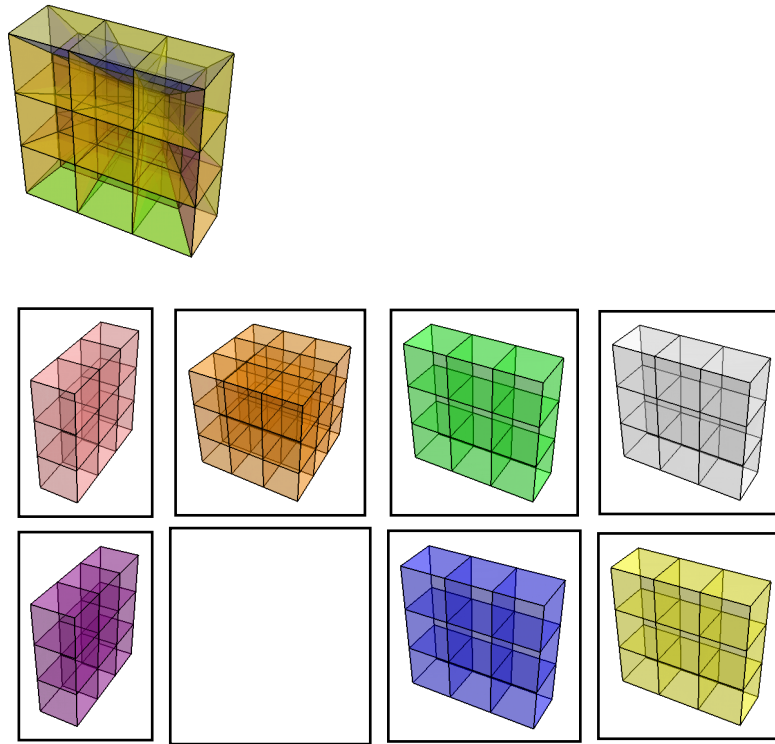
In the 3D case of Rubik's cube  $R_3$ , a *block* is a set of nine small cubes whose centers have one coordinate that is constant,  $-1$ ,  $0$  or  $1$ . There are nine blocks, three per coordinate axis. A natural technique to rotate a middle block, for example, the one cut by the  $z = 0$  plane by  $\pi/2$ , is to rotate the block above by  $-\pi/2$ , the block below by  $-\pi/2$ , and then the whole cube by  $\pi/2$ .

In the 4D case, a *block* is a set of 27 small 4-cubes whose centers have one coordinate that is constant. There are 12 blocks, four per axis and three per choice of constant coordinate  $-1$ ,  $0$  or  $1$ . Under a rotation, the small 4-cubes in a block change position simultaneously. Each block is a four-dimensional *hyperprism* with height 1.

Figure 6 shows an example of the block  $y = -1$ ; the cell opposite the orange cell is not colored.

```
Figure6Aux[graphics3D_] :=
  Framed@Graphics3D[{Opacity[0.3], graphics3D},
    Boxed → False, Lighting → "Neutral"]

Module[{c2},
  c2 = Select[ColoredSmallCells, #[[2, 2]] == -1 &];
  Row[{
    Graphics3D[{Opacity[0.3], draw4Cube /@ c2}, Boxed → False,
      ImageSize → 160, Lighting → "Neutral"],
    Spacer[40],
    Grid[{
      {
        Figure6Aux@drawCell[c2, 1, -1],
        Figure6Aux@drawCell[c2, 2, -1],
        Figure6Aux@drawCell[c2, 3, -1],
        Figure6Aux@drawCell[c2, 4, -1]
      },
      {
        Figure6Aux@drawCell[c2, 1, 1],
        Figure6Aux@drawCell[c2, 2, 1],
        Figure6Aux@drawCell[c2, 3, 1],
        Figure6Aux@drawCell[c2, 4, 1]
      }
    }]
  }]
]
```



▲ **Figure 6.** Example of a block of 27 small 4-cubes (orange,  $y = -1$ ).

A block is rotated by  $\pi/2$ ,  $\pi$  or  $-\pi/2$  around an axis, which is a fixed plane. Therefore, the information needed for an action on the Rubik 4-cube is (1) the block to be rotated; (2) the axis of rotation; and (3) the angle. For Rubik's cube  $R_3$ , the axis of rotation is automatically determined by selecting a block. But for  $R_4$ , two coordinate axes must be chosen to determine the fixed plane. One is the constant coordinate axis used to select the block, and the other must be chosen from the remaining three coordinate axes. There are 108 possible actions on  $R_4$ : 12 choices of block, three choices for the second coordinate axis and three choices of angle:  $12 \times 3 \times 3 = 108$ . Therefore, 108 buttons are required for the rotations in the Rubik 4-cube computer program. Table 2 lists the properties of Rubik's cube and Rubik's 4-cube.

	Rubik's cube		Rubik's 4-cube	
2-faces (number of colors)	6	3-faces (number of colors)	8	
small cubes	$27 = 3^3$	small 4-cubes	$81 = 3^4$	
divided 2-faces	$54 = 6 \times 3^2$	divided 3-faces	$216 = 8 \times 3^3$	
squares to color	54	squares to color	$1296 = 216 \times 6$	

▲ **Table 2.** Properties of Rubik's cube and Rubik's 4-cube.

## □ Final Form

The program to realize Rubik's 4-cube in 3D relies on central projection of a hypercube and rotation matrices in 4D. The program is shown in the next section.

Implementing an interface consists of three parts: constructing the buttons for the rotations, displaying the current state and judging whether the puzzle is complete.

The buttons for the rotations are placed in grids. The player can rotate a block by clicking one of the buttons. The rows correspond to the selection of the axis of the coordinates of the block and the columns correspond to the coordinate values for that axis. The player can select a block by choosing one of the rows and one of the columns. For example, clicking the button where row  $z$  crosses column  $-1$  chooses the block on  $z = -1$ . For each block, the other three axes are listed. Then, the selection of the second axis is required to verify the rotational plane. Finally, one of the three buttons (up, diagonal and down) must be chosen to determine the rotation angle of  $\pi/2$  ( $\blacktriangle$ ),  $\pi$  ( $\blacksquare$ ) and  $-\pi/2$  ( $\blacktriangledown$ ). (The 0 rows can be ignored—the player can perform an equivalent pair of actions instead in the parallel blocks.)

When the colors of the 27 subcubes on a cell are all the same, that cell is complete. The puzzle is solved when all the cells are complete.

## ■ Program

```
GridAux[graphics3D_] :=
  Graphics3D[{Opacity[1], graphics3D}, Background → Black,
    SphericalRegion → True, Boxed → False, ImageSize → 150,
    Lighting → "Neutral"];

Module[{reset, surfaceCubeRotate, randomize, rXYm,
  rXYo, rXYp, rXZm, rXZo, rXZp, rXWm, rXWo, rXWp,
  rYZm, rYZo, rYZp, rYWm, rYWo, rYWp, rZWm, rZWo, rZWp,
  rmtx},
  Panel[
    DynamicModule[
      {initialSurfaceCubes},

      {rXYm, rXYo, rXYp, rXZm, rXZo, rXZp, rXWm, rXWo, rXWp,
        rYZm, rYZo, rYZp, rYWm, rYWo, rYWp, rZWm, rZWo, rZWp} = {
        {{1, 0, 0, 0}, {0, 1, 0, 0}, {0, 0, 0, 1}, {0, 0, -1, 0}},
        {{1, 0, 0, 0}, {0, 1, 0, 0}, {0, 0, -1, 0},
          {0, 0, 0, -1}},
        {{1, 0, 0, 0}, {0, 1, 0, 0}, {0, 0, 0, -1}, {0, 0, 1, 0}},
        {{1, 0, 0, 0}, {0, 0, 0, 1}, {0, 0, 1, 0}, {0, -1, 0, 0}},
        {{1, 0, 0, 0}, {0, -1, 0, 0}, {0, 0, 1, 0},
          {0, 0, 0, -1}},
        {{1, 0, 0, 0}, {0, 0, 0, -1}, {0, 0, 1, 0}, {0, 1, 0, 0}},
        {{1, 0, 0, 0}, {0, 0, 1, 0}, {0, -1, 0, 0}, {0, 0, 0, 1}},
        {{1, 0, 0, 0}, {0, -1, 0, 0}, {0, 0, -1, 0},
```



```

    {0, 0, 0, 1}},
    {{1, 0, 0, 0}, {0, 0, -1, 0}, {0, 1, 0, 0}, {0, 0, 0, 1}},
    {{0, 0, 0, 1}, {0, 1, 0, 0}, {0, 0, 1, 0}, {-1, 0, 0, 0}},
    {{-1, 0, 0, 0}, {0, 1, 0, 0}, {0, 0, 1, 0},
    {0, 0, 0, -1}},
    {{0, 0, 0, -1}, {0, 1, 0, 0}, {0, 0, 1, 0}, {1, 0, 0, 0}},
    {{0, 0, 1, 0}, {0, 1, 0, 0}, {-1, 0, 0, 0}, {0, 0, 0, 1}},
    {{-1, 0, 0, 0}, {0, 1, 0, 0}, {0, 0, -1, 0},
    {0, 0, 0, 1}},
    {{0, 0, -1, 0}, {0, 1, 0, 0}, {1, 0, 0, 0}, {0, 0, 0, 1}},
    {{0, 1, 0, 0}, {-1, 0, 0, 0}, {0, 0, 1, 0}, {0, 0, 0, 1}},
    {{-1, 0, 0, 0}, {0, -1, 0, 0}, {0, 0, 1, 0},
    {0, 0, 0, 1}},
    {{0, -1, 0, 0}, {1, 0, 0, 0}, {0, 0, 1, 0}, {0, 0, 0, 1}}
  };

rmtx[1, 2, 1] := rXYp;
rmtx[1, 2, 0] := rXYo;
rmtx[1, 2, -1] := rXYm;
rmtx[1, 3, 1] := rXZp;
rmtx[1, 3, 0] := rXZo;
rmtx[1, 3, -1] := rXZm;
rmtx[1, 4, 1] := rXWp;
rmtx[1, 4, 0] := rXWo;
rmtx[1, 4, -1] := rXWm;

rmtx[2, 1, 1] := rXYm;
rmtx[2, 1, 0] := rXYo;
rmtx[2, 1, -1] := rXYp;
rmtx[2, 3, 1] := rYZp;
rmtx[2, 3, 0] := rYZo;
rmtx[2, 3, -1] := rYZm;
rmtx[2, 4, 1] := rYWp;
rmtx[2, 4, 0] := rYWo;
rmtx[2, 4, -1] := rYWm;

rmtx[3, 1, 1] := rXZm;
rmtx[3, 1, 0] := rXZo;
rmtx[3, 1, -1] := rXZp;
rmtx[3, 2, 1] := rYZm;
rmtx[3, 2, 0] := rYZo;
rmtx[3, 2, -1] := rYZp;
rmtx[3, 4, 1] := rZWp;
rmtx[3, 4, 0] := rZWo;
rmtx[3, 4, -1] := rZWm;

rmtx[4, 1, 1] := rXWm;
rmtx[4, 1, 0] := rXWo;
rmtx[4, 1, -1] := rXWp;
rmtx[4, 2, 1] := rYWm;
rmtx[4, 2, 0] := rYWo;
rmtx[4, 2, -1] := rYWp;
rmtx[4, 3, 1] := rZWm;
rmtx[4, 3, 0] := rZWo;
rmtx[4, 3, -1] := rZWp;

```

```

initialSurfaceCubes = ColoredSmallCells;
reset[] := surfaceCubes = initialSurfaceCubes;
reset[];

surfaceCubeRotate[sc_, xyzw_, loc_, ax_, pm_] :=
Module[{tmp},
  tmp =
    (If[#[[2, xyzw]] == loc,
      tmp = {Map[(rmtx[xyzw, ax, pm].#) &, #[[1]], {2}],
            rmtx[xyzw, ax, pm].#[[2]], #[[3]],
            rmtx[xyzw, ax, pm].#[[4]]}, #] & /@ sc);
  tmp
];

randomize[sc_] := Module[{xyzw, pos, planeNo, direc},
  xyzw = RandomInteger[{1, 4}];
  pos = RandomInteger[{-1, 1}];
  planeNo = xyzw;
  While[planeNo == xyzw,
    planeNo = RandomInteger[{1, 4}]];
  direc = 2 RandomInteger[{0, 1}] - 1;
  surfaceCubeRotate[sc, xyzw, pos, planeNo, direc]
];

Buttons[q_] := {ToString[q],
  Grid[{
    {"y",
      Button[▲, surfaceCubes =
        surfaceCubeRotate[surfaceCubes, 1, q, 2, 1]],
      Button[■, surfaceCubes =
        surfaceCubeRotate[surfaceCubes, 1, q, 2, 0]],
      Button[▼, surfaceCubes =
        surfaceCubeRotate[surfaceCubes, 1, q, 2, -1]]},
    {"z",
      Button[▲, surfaceCubes =
        surfaceCubeRotate[surfaceCubes, 1, q, 3, 1]],
      Button[■, surfaceCubes =
        surfaceCubeRotate[surfaceCubes, 1, q, 3, 0]],
      Button[▼, surfaceCubes =
        surfaceCubeRotate[surfaceCubes, 1, q, 3, -1]]},
    {"w",
      Button[▲, surfaceCubes =
        surfaceCubeRotate[surfaceCubes, 1, q, 4, 1]],
      Button[■, surfaceCubes =
        surfaceCubeRotate[surfaceCubes, 1, q, 4, 0]],
      Button[▼, surfaceCubes =
        surfaceCubeRotate[surfaceCubes, 1, q, 4, -1]]}],
  Grid[{
    {"x",
      Button[▲, surfaceCubes =
        surfaceCubeRotate[surfaceCubes, 2, q, 1, 1]],
      Button[■, surfaceCubes =
        surfaceCubeRotate[surfaceCubes, 2, q, 1, 0]],
      Button[▼, surfaceCubes =

```

```

        surfaceCubeRotate[surfaceCubes, 2, q, 1, -1]]},
{"z",
  Button[▲, surfaceCubes =
    surfaceCubeRotate[surfaceCubes, 2, q, 3, 1]],
  Button[■, surfaceCubes =
    surfaceCubeRotate[surfaceCubes, 2, q, 3, 0]],
  Button[▼, surfaceCubes =
    surfaceCubeRotate[surfaceCubes, 2, q, 3, -1]]},
{"w",
  Button[▲, surfaceCubes =
    surfaceCubeRotate[surfaceCubes, 2, q, 4, 1]],
  Button[■, surfaceCubes =
    surfaceCubeRotate[surfaceCubes, 2, q, 4, 0]],
  Button[▼, surfaceCubes =
    surfaceCubeRotate[surfaceCubes, 2, q, 4, -1]]}],
Grid[{
  {"x",
    Button[▲, surfaceCubes =
      surfaceCubeRotate[surfaceCubes, 3, q, 1, 1]],
    Button[■, surfaceCubes =
      surfaceCubeRotate[surfaceCubes, 3, q, 1, 0]],
    Button[▼, surfaceCubes =
      surfaceCubeRotate[surfaceCubes, 3, q, 1, -1]]},
  {"y",
    Button[▲, surfaceCubes =
      surfaceCubeRotate[surfaceCubes, 3, q, 2, 1]],
    Button[■, surfaceCubes =
      surfaceCubeRotate[surfaceCubes, 3, q, 2, 0]],
    Button[▼, surfaceCubes =
      surfaceCubeRotate[surfaceCubes, 3, q, 2, -1]]},
  {"w",
    Button[▲, surfaceCubes =
      surfaceCubeRotate[surfaceCubes, 3, q, 4, 1]],
    Button[■, surfaceCubes =
      surfaceCubeRotate[surfaceCubes, 3, q, 4, 0]],
    Button[▼, surfaceCubes =
      surfaceCubeRotate[surfaceCubes, 3, q, 4, -1]]}],
Grid[{
  {"x",
    Button[▲, surfaceCubes =
      surfaceCubeRotate[surfaceCubes, 4, q, 1, 1]],
    Button[■, surfaceCubes =
      surfaceCubeRotate[surfaceCubes, 4, q, 1, 0]],
    Button[▼, surfaceCubes =
      surfaceCubeRotate[surfaceCubes, 4, q, 1, -1]]},
  {"y",
    Button[▲, surfaceCubes =
      surfaceCubeRotate[surfaceCubes, 4, q, 2, 1]],
    Button[■, surfaceCubes =
      surfaceCubeRotate[surfaceCubes, 4, q, 2, 0]],
    Button[▼, surfaceCubes =
      surfaceCubeRotate[surfaceCubes, 4, q, 2, -1]]},
  {"z",
    Button[▲, surfaceCubes =
      surfaceCubeRotate[surfaceCubes, 4, q, 3, 1]],

```

```

        Button[■, surfaceCubes =
            surfaceCubeRotate[surfaceCubes, 4, q, 3, 0]],
        Button[▼, surfaceCubes =
            surfaceCubeRotate[surfaceCubes, 4, q, 3, -1]]}}}
    };

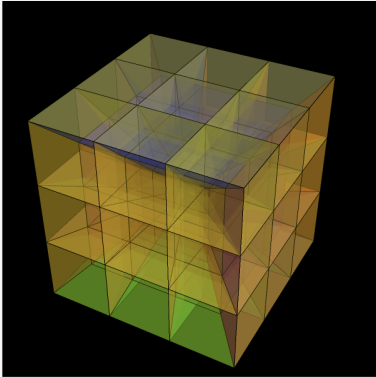
Column[{
    Dynamic[Graphics3D[
        {Opacity[0.3], draw4Cube /@ surfaceCubes},
        Background → Black, SphericalRegion → True,
        Boxed → False, ImageSize → 360,
        Lighting → "Neutral"]],
    "",

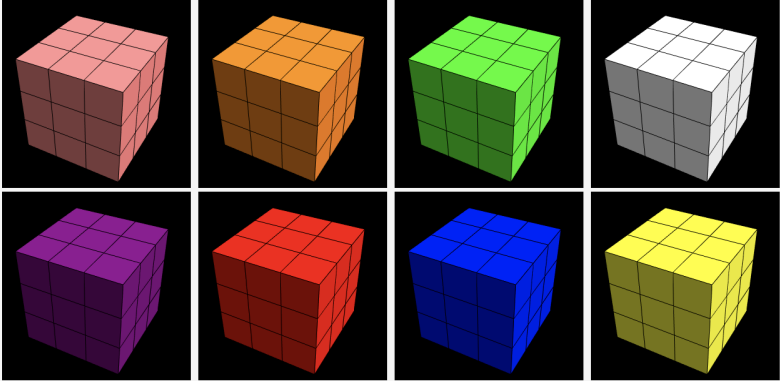
    Grid[{
        {
            Dynamic[GridAux[drawCell[surfaceCubes, 1, -1]]],
            Dynamic[GridAux[drawCell[surfaceCubes, 2, -1]]],
            Dynamic[GridAux[drawCell[surfaceCubes, 3, -1]]],
            Dynamic[GridAux[drawCell[surfaceCubes, 4, -1]]]
        },
        {
            Dynamic[GridAux[drawCell[surfaceCubes, 1, 1]]],
            Dynamic[GridAux[drawCell[surfaceCubes, 2, 1]]],
            Dynamic[GridAux[drawCell[surfaceCubes, 3, 1]]],
            Dynamic[GridAux[drawCell[surfaceCubes, 4, 1]]]
        }
    ]],

    Grid[{{
        Button["Reset", reset[]],
        Button["Randomize",
            surfaceCubes = Nest[randomize, surfaceCubes,
                20]]}}],

    Grid[
        Transpose@{
            {"", "x", "y", "z", "w"},
            Buttons[-1],
            Buttons[0],
            Buttons[1]
        }
        , Frame → All]
    }, Alignment → Center],
    SaveDefinitions → True
]
]
]

```





Reset Randomize

		-1			0			1				
x	y	▲	■	▼	y	▲	■	▼	y	▲	■	▼
	z	▲	■	▼	z	▲	■	▼	z	▲	■	▼
	w	▲	■	▼	w	▲	■	▼	w	▲	■	▼
y	x	▲	■	▼	x	▲	■	▼	x	▲	■	▼
	z	▲	■	▼	z	▲	■	▼	z	▲	■	▼
	w	▲	■	▼	w	▲	■	▼	w	▲	■	▼
z	x	▲	■	▼	x	▲	■	▼	x	▲	■	▼
	y	▲	■	▼	y	▲	■	▼	y	▲	■	▼
	w	▲	■	▼	w	▲	■	▼	w	▲	■	▼
w	x	▲	■	▼	x	▲	■	▼	x	▲	■	▼
	y	▲	■	▼	y	▲	■	▼	y	▲	■	▼
	z	▲	■	▼	z	▲	■	▼	z	▲	■	▼

## ■ Discussion

Although we succeeded in implementing Rubik's 4-cube, some problems remain to be addressed. We aimed for ease of implementation rather than efficiency. Therefore, in the future, we should consider enhancing the application to get a more effective visualization method and an intuitive interface.

The program redraws 1,296 squares after each rotation, so that efficient coding is important. There is a great deal of redundancy in calculating the vertices of the 4-subcubes for each rotation. The most effective method for handling the vertices using subcubes or 4-subcubes remains to be clarified. Note that we must transfer the vertices of the 4-subcubes to those of the subcubes when we handle the vertices of 4-subcubes as a dataset rather than handling them as subcubes.

Effective visualization is a common problem for four-dimensional geometry. In this article, we used central projection to represent 4-cubes. However, the proposed projection does not completely represent the features of the puzzle. Although there are other projections for representing a 4-cube, the most suitable method is not yet clear.

Another possible improvement would be to animate the rotation. The animation of the rotation of the colored small 4-cubes would help the player intuitively understand their rearrangement.

An intuitive interface is important for playing this puzzle. The interface and visualization issues are related, and their development may provide a new method for understanding four-dimensional space.

## ■ References

- [1] H. S. M. Coxeter, *Introduction to Geometry*, 2nd ed., Hoboken: Wiley, 1989.
- [2] T. Yoshino. "Activities of Dr. Takashi Yoshino." (Dec 11, 2017)  
[takashiyoshino.random-walk.org/basic-data-of-4d-regular-polytopes](http://takashiyoshino.random-walk.org/basic-data-of-4d-regular-polytopes).
- [3] K. Miyazaki, M. Ishii and S. Yamaguchi, *Science of Higher-Dimensional Shape and Symmetry*, Kyoto: Kyoto University Press, 2005 (In Japanese).

T. Yoshino, "Rubik's 4-Cube," *The Mathematica Journal*, 2017. [dx.doi.org/doi:10.3888/tmj.19-8](https://doi.org/10.3888/tmj.19-8).

## About the Author

Profession: Science of Form. Fields of interest: skeletal structure of plankton, non-Euclidean geometry, hyperspace, pattern formation.

**Takashi Yoshino**

Toyo University,  
 Kujirai 2100, Kawagoe, 350-8585, JAPAN  
[tyoshino@toyo.jp](mailto:tyoshino@toyo.jp)