REPRESENTING RUBIK'S CUBE IN APL

Howard A. Peelle
Professor
Instructional Applications of Computers
University of Massachusetts
Amherst, MA USA 01003
(413) 545 - 0496
(413) 545 - 0246

Abstract

Seven alternative representations of Rubik's Cube are presented and compared: a 3-by-3-by-3 array of 3-digit integers; a 6-by-3-by-3 array of literals; a 5-by-12 literal matrix; an 11-by-11 sparse literal matrix; a 54-element vector; a 4-dimension array; and a 3-by-3-by-3 nested array. APL functions are given for orientation moves and quarter-turns plus several useful tools for solving the cube.

Introduction

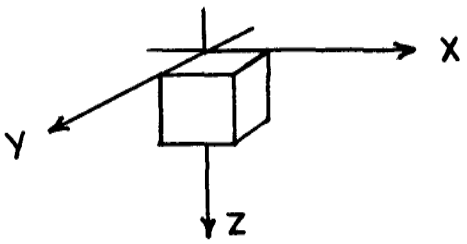
Rubik's Cube* is an intriguing puzzle which became extremely popular world-wide in an amazingly short time.(1 2) Some would say it is an 'addictive' pasttime, much like APL. It is not surprising, then, that the two would meet since APL is ideal for manipulating multi-dimensional objects.

This paper concerns the representation of Rubik's Cube -- a prerequisite for any solution effort. To begin with, we must consider questions such as: What data structure to use? How to define moves?

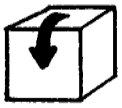
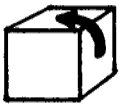
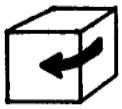
A number of different representations are presented and compared here -- each using APL notation. Arrays are used to represent Rubik's Cube, and defined functions are used to express movements on the cube. Direct definition form and `⎕IO←1` are used throughout.

Orientation Moves and Quarter-Turns

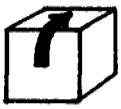
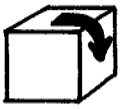
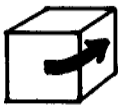
Assume Rubik's Cube is oriented by the coordinate axes shown below. The placement and direction of the axes correspond to conventional row-major order used in APL arrays,i.e. the origin is top, back, left:



The cube's orientation can be changed by the following moves:

X		90° rotation about X axis
Y		90° rotation about Y axis
Z		90° rotation about Z axis

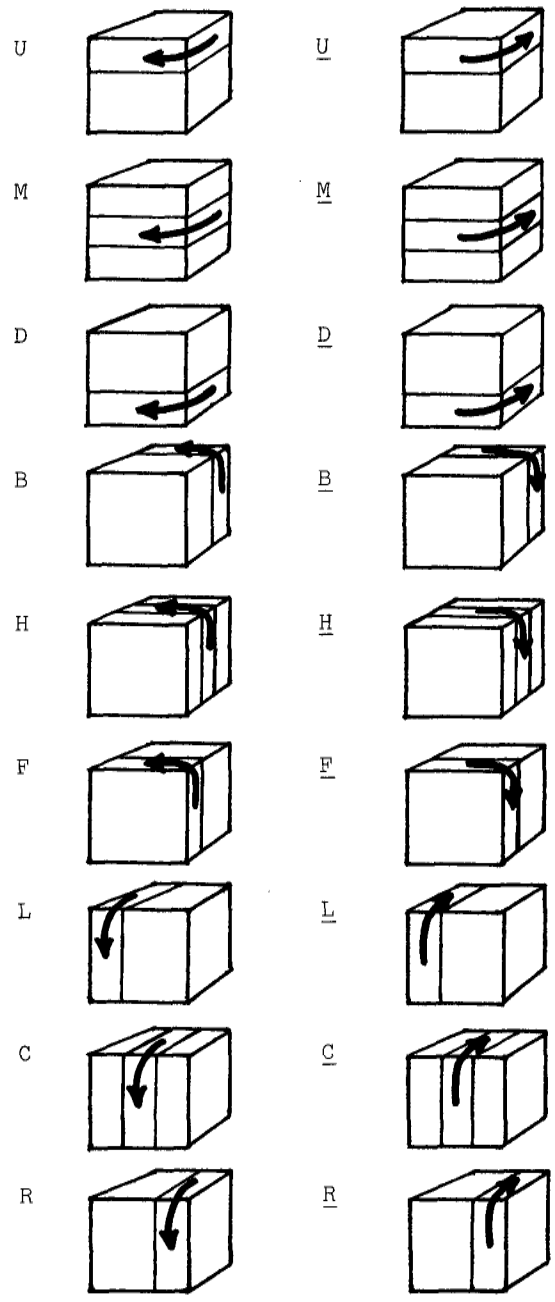
and their inverses:

$\underline{X}$		-90° rotation about X axis
$\underline{Y}$		-90° rotation about Y axis
$\underline{Z}$		-90° rotation about Z axis

---

* "Rubik's Cube" is a trademark of Ideal Toy Corporation.

There are nine quarter-turns plus their inverses, illustrated below along with their names:



Numeric Representation

Rubik's Cube can be represented as a 3-by-3-by-3 cubic array of integers. Each cubie is represented by a 3-digit integer, with each digit representing a color:

1 for White
2 for Blue
3 for Red
4 for Orange
5 for Green
6 for Yellow
(and 0 for Black)

The place-values of the digits correspond to the three axes of the cube in a certain (arbitrary) order: (up/down, back/front, left/right). So, the cube looks like this in its goal state:

```
      CUBE

135 130 136
105 100 106
145 140 146

 35  30  36
  5   0   6
 45  40  46

235 230 236
205 200 206
245 240 246
```

Note that a corner cubie has three nonzero digits, an edge cubie has two and a center, one.

```
      ⍴CUBE

3 3 3
```

Orientation moves can be represented by the APL functions below:

```
X: 10⊥((3⍴10)⊤2 1 3⌽⍋⍋⍵[1])[2 1 3;;;]

Y: 10⊥((3⍴10)⊤3 2 1⌽⍋⍋⍵[3])[3 2 1;;;]

Z: 10⊥((3⍴10)⊤1 3 2⌽⍋⍋⍵[2])[1 3 2;;;]
```

and their inverses:

```
X̲: 10⊥((3⍴10)⊤2 1 3⌽⍋⍋⍵[2])[2 1 3;;;]

Y̲: 10⊥((3⍴10)⊤3 2 1⌽⍋⍋⍵[1])[3 2 1;;;]

Z̲: 10⊥((3⍴10)⊤1 3 2⌽⍋⍋⍵[3])[1 3 2;;;]
```

Note that since three quarter-turns are equivalent to one reverse turn, these (and all inverses to follow) could be defined more simply (but less efficiently):

```
X̲: X X X ⍵

Y̲: Y Y Y ⍵

Z̲: Z Z Z ⍵
```

The quarter-turns are then defined in terms of X, Y, and Z:*

```
L: (X⍵[;;;1]),[3] ⍵[;;;2] ,[3] ⍵[;;;3]

C:   ⍵[;;;1] ,[3](X⍵[;;;2]),[3] ⍵[;;;3]

R:   ⍵[;;;1] ,[3] ⍵[;;;2] ,[3](X⍵[;;;3])

B: (Y⍵[;;1;]),[2] ⍵[;;2;] ,[2] ⍵[;;3;]

H:   ⍵[;;1;] ,[2](Y⍵[;;2;]),[2] ⍵[;;3;]

F:   ⍵[;;1;] ,[2] ⍵[;;2;] ,[2](Y⍵[;;3;])

U: (Z⍵[;1;;]),[1] ⍵[;2;;] ,[1] ⍵[;3;;]

M:   ⍵[;1;;] ,[1](Z⍵[;2;;]),[1] ⍵[;3;;]

D:   ⍵[;1;;] ,[1] ⍵[;2;;] ,[1](Z⍵[;3;;])
```

Their inverses are identical, except with X̲ in place of X, Y̲ for Y, and Z̲ for Z.

---

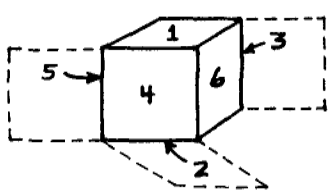
* These functions are written out this way in order to reveal patterns. Some can, of course, be expressed more concisely, e.g.:
R: ⍵[;;1 2],X⍵[;;;3]

Local Representation

An alternative representation of Rubik's Cube is a 6-by-3-by-3 cubic array. Each face of the cube is a 3 by 3 matrix, where each facelet's color is represented by a literal character:

'W' for White
'B' for Blue
'R' for Red
'O' for Orange
'G' for Green
'Y' for Yellow

The positions of the characters in each matrix correspond to the positions of colors on each respective face of the cube viewed locally -- that is, as they would be seen looking directly at a face (with the origin at the top, left). The ordering of faces in the array is indicated below:



1 Up
2 Down
3 Back
4 Front
5 Left
6 Right

```
      CUBE

WWW
WWW
WWW

BBB
BBB
BBB

RRR
RRR
RRR

OOO
OOO
OOO

GGG
GGG
GGG

YYY
YYY
YYY


      ρCUBE

6 3 3
```

Orientation moves are defined as follows:*

```
X: (⊖⍵[3;;])⍪⍵[4;;]⍪(⊖⍵[2;;])⍪
   ⍵[1;;]⍪(⌽⍉⍵[5;;]),[.5](⌽⍉⍵[6;;])

Y: (⍉⌽⍵[6;;])⍪(⌽⍉⍵[5;;])⍪(⍉⌽⍵[3;;])⍪
   (⍉⌽⍵[4;;])⍪(⌽⍉⍵[1;;]),[.5](⍉⌽⍵[2;;])

Z: (⍉⌽⍵[1;;])⍪(⌽⍉⍵[2;;])⍪⍵[5 6 4 3;;]
```

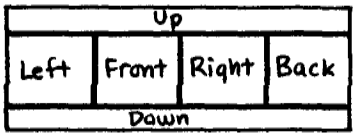
and their inverses are similar.

Quarter-turns are defined independently, for example:

```
L: ((⍉⍵[3;;3]),⍵[1;;2 3])⍪
   (⍵[4;;1],⍵[2;;2 3])⍪
   (⍵[3;;1 2],⍉⍵[2;;1])⍪
   (⍵[1;;1],⍵[4;;2 3])⍪
   (⍉⌽⍵[5;;]),[.5]⍵[6;;]
```

Composite Representation

The cube can also be represented as a 5-by-12 matrix. Four 3-by-3 sub-matrices represent the left, front, right, and back faces; and two 12-element rows represent the up and down faces, as shown below:



Note that the left and back faces appear flapped out. Note also that the up and down faces appear without their centers and with redundant corner colors; that is, the 12-element vector in the 1st (and 5th) row is composed of the 8 colors surrounding the center, with each of the four corner colors duplicated.

```
      CUBE

WWWWWWWWWWWW
GGGOOOYYYRRR
GGGOOOYYYRRR
GGGOOOYYYRRR
BBBBBBBBBBBB


      ρCUBE

5 12
```

Since the up and down centers are not explicitly represented, a fixed axis must be assumed and only one orientation move is defined:

```
Z: 3⍉⍵        and its inverse:    Z: ¯3⍉⍵
```

Examples of quarter-turns are:*

```
L: ⍵[;12 1 2 3 4]←⍉⌽⍵[;12 1 2 3 4] ⋇
   ⍵[1 5;12 4]←⍵[1 5;1 3] ⋇ ⍵

F: ⍵[;3 4 5 6 7]←⌽⍉⍵[;3 4 5 6 7] ⋇
   ⍵[1 5;3 7]←⍵[1 5;4 6] ⋇ ⍵

D: 0 0 0 3 3 ⍉ ⍵
```

Inverses use ⍉⌽ (clockwise face rotation) instead of ⌽⍉ (counterclockwise) or vice versa and ¯3 instead of 3 (for U, M, D).

---

* Note the use of ⍪ instead of ,[1] .

---

* Note use of ⋇ as a statement separator.

Visual Representation

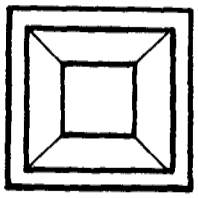
Here the cube is represented in an 11-by-11 matrix, containing literal characters representing all facelet colors -- except the back center. These 53 literals are placed along horizontal, vertical, and diagonal lines against a background of blanks, as shown below:

```
      (22ρ1 0)\CUBE
B           B           B
    R       R       R
  G   R     R   R     Y
    G   R R R     Y
      G W W W Y
B G G G W W W Y Y Y B
      G W W W Y
    G   O O O   Y
  G   O   O   O   Y
    O       O       O
B           B           B
```

Note that the display is expanded in order to appear more like a square.

```
      ρCUBE
11 11
```

The display is intended to give the visual appearance of a square pyramid, viewed from atop, as if the cube were stretched out from the bottom and flattened onto a 2-dimensional surface:



Only one orientation move is defined here:

```
Z: ⌽⍉ω
```

and its inverse:

```
Z: ⍉⌽ω
```

X and Y and their inverses are not defined because they affect the position of the down center, which is not represented.

The Z-axis quarter-turns can be defined like this:

```
U: ω[I;I]←Zω[I;I←3+ι5] ⍝ ω

D: ω[I;I]←Zω[I;I←2+ι7] ⍝ Zω

M: U D Z ω
```

The others are tedious piece-by-piece transformations.

Vector Representation

Of course, Rubik's Cube could be represented as a 54-element vector (literal or numeric). Each element represents a particular facelet's color. Taking the facelets in row-major order on each face locally (as in the Local Representation) in up, down, back, front, left, right order of faces, the cube in its goal state is:

```
      CUBE
WWWWWWWWWBBBBBBBBBRRRRRRRRRGGGGGGGGGOOOOOOOOO
      YYYYYYYYY
      ρCUBE
54
```

Quarter-turns are explicit transformations such as:*

```
U: ω[ι9]←ω[7 4 1 8 5 2 9 6 3] ⍝
   ω[19 20 21 28 29 30 37 38 39 46 47 48]←
   ω[37 38 39 46 47 48 19 20 21 28 29 30]⍝
   ω

F: ω[27+ι9]←ω[27+3 6 9 2 5 8 1 4 7] ⍝
   ω[7 8 9 10 11 12 39 42 45 46 49 52]←
   ω[46 49 52 39 42 45 9 8 7 12 11 10]⍝
   ω

R: ω[45+ι9]←ω[45+3 6 9 2 5 8 1 4 7] ⍝
   ω[3 6 9 12 15 18 19 22 25 30 33 36]←
   ω[25 22 19 30 33 36 18 15 12 3 6 9]⍝
   ω
```

And orientation moves are combinations of them:

```
X: L C R ω

Y: B H F ω

Z: U M D ω
```

4-D Representation

This representation employs a 4-dimensional literal array for Rubik's Cube. Each cubie is a planar vector (in the 2nd dimension) with facelet colors in up/down, back/front, left/right order going down each of three 3-by-3 matrices. Note that the 1st matrix in the 2nd cubic sub-array is all blank since it has no up or down colors; all 2nd rows of 2nd matrices have no back/front colors; and 2nd columns of all 3rd matrices similarly have no left/right colors. Note also that the bottom colors are shown in place, looking from above, as if the cube were transparent. Each 3-by-3-by-3 array, then, is a layer of Rubik's Cube, illustrated below side-by-side (to conserve paper):

```
      CUBE
WWW     RRR     OOO     G Y     BBB     RRR     OOO     G Y
WWW             OOO     G Y     BBB             OOO     G Y
WWW             OOO     G Y     BBB             OOO     G Y
      ρCUBE
3 3 3 3
```

---

* Each of these functions could be expressed instead in a single line (but less succinctly) using a 54-element permutation vector.

Orientation moves are:

```
X: 3 2 1 4⌽⍉[1]⍵[;2 1 3;;]

Y: 4 2 3 1⌽⍉[4]⍵[;3 2 1;;]

Z: 1 2 4 3⌽⍉[3]⍵[;1 3 2;;]
```

and inverses, similarly.

Quarter-turns apply an orientation move to a layer (as in the Numeric Representation). For example:

```
L: (X⍵[;;;,1]),⍵[;;;2 3]
```

Nested Representation

Lastly, we consider representing Rubik's Cube as a nested array: a 3-by-3-by-3 array of 'regular' 3-element vectors (padded with jots):

```
      CUBE

WRG WR∘ WRY          Note: each vector
W∘G W∘∘ W∘Y          lists colors in
WOG WO∘ WOY          up/down,
                     front/back,
∘RG ∘R∘ ∘RY          left/right
∘∘G ∘∘∘ ∘∘Y          order.
∘OG ∘O∘ ∘OY

BRG BR∘ BRY
B∘G B∘∘ B∘Y
BOG BO∘ BOY

      ⍴CUBE

3 3 3
```

This allows orientation moves to be defined like this:*

```
X: 2 1 3⌽⍉[1]2 1 3 PERMUTE EACHRIGHT ⍵

Y: 3 2 1⌽⍉[3]3 2 1 PERMUTE EACHRIGHT ⍵

Z: 1 3 2⌽⍉[2]1 3 2 PERMUTE EACHRIGHT ⍵
```

The quarter-turns are the same as for the Numeric Representation, e.g.:

```
L: (X⍵[;;,1]),⍵[;;2 3]
```

A variant of this representation could conserve space by using 'ragged' literal vectors in a 3-by-3-by-3 nested array to represent corners, edges, and centers most succinctly, e.g. ('WRG'), ('WR'), ('R'). Unfortunately, the function defintions become more complicated.

Other Representations

Copies of all defined functions discussed here are available in fixed form from the author by request. Suggestions for improvements as well as other proposed representations are also welcomed.

---

* Note the use of cover function *PERMUTE* and defined operator *EACHRIGHT* (from T. More's paper in APL79 Conference Proceedings) to permute the elements of nested vectors here instead of ⍵[¨2 1 3] .

Comparison of Representations

To begin with, these seven representations can be classified into three groups, roughly as follows: 1) The Numeric, 4-D, and Nested representations are very similar -- all essentially involve encoding four dimensions (x,y,z position and color). 2) The Local, Composite, and Visual representations involve unwrapping the cube in some way. 3) The Vector representation underlies them all, since there is a fundamental need to represent colors in memory locations and to swap their positions.

Further, representations can be compared in general terms of comprehensiveness, comprehensibility, ease of use, and efficiency. More specifically, "comprehensive" means:

- complete -- the extent to which the representation captures the properties of Rubik's Cube
- general -- how well it serves as a basis for generalization to other similar puzzles, such as the Pyraminx or Rubik's Revenge*

"comprehensible" means:

- simple -- how easy it is to understand the function definitions
- concise -- brevity of functions
- display -- how suitable the data structure is for viewing

"ease of use" means:

- manipulable -- how easy it is to use the functions to manipulate the data structure
- insight -- the potential for leading to insights about the mathematical structures inherent in Rubik's Cube

"efficiency" means:

- space -- storage requirements (for the data structure)
- speed -- time to execute moves (quarter-turns)

These will be the specific criteria for comparing representations here. Note that measures of space do not include storage requirements for the defined functions since some representations are not complete. Also note that measures of speed are for quarter-turns only since some representations do not include all orientation moves.

First, consider the Numeric Representation. It offers a complete 3-dimensional data structure which corresponds to the actual structure of the cube. The orientation moves and quarter-turns are consistent in their definitions and reveal isomorphisms. However, they use sophisticated primitive functions (⊥ ⊤ ⍉), are challenging to understand, and are very slow in execution. Storage requirements are competitive (and might be best if space for function definitions were considered). Display of the cube is difficult to interpret as is and would

---

* Pyraminx™ is a three-tier tetrahedron. Rubik's Revenge™ is a 4-by-4-by-4 cube.

probably call for an additional function to translate it into literals. The overall representation is poor in its generality, since it is limited to nine distinct colors and cubical shapes.

The Local Representation might seem natural to those who prefer treating each face of the cube independently*, i.e. matrices of colors, as opposed to cubies (which excludes the geometric center). The orientation functions are not simple since they must be done face-by-face and then pieced together. The quarter-turns are even less comprehensible because some (six of nine) cause two different kinds of effects -- rotation of positions on a face and translations of positions along a row or column of four contiguous faces -- which are treated separately. Further, they can not make good use of the defined orientation functions X, Y, and Z. The display, however, is nice (and would be more convenient if reshaped to print horizontally). Its space requirements are minimal, but execution speed is rather slow. The representation does generalize easily to puzzles with any number of (rectangular) faces, but lends little insight into how they interact.

The Composite Representation provides a reasonable compromise, offering a single matrix data structure and straightforward function definitions -- but with some inherent shortcomings. To begin with, it is not complete. The up and down centers of the cube are not represented (although they can be readily determined from the positions of the other centers); hence, X and Y orientation moves and their inverses are not defined, nor are quarter-turns H and C and their inverses. Further, the first and fifth rows of the 5-by-12 matrix include redundant data -- the colors of the up and down corners are duplicated -- so that the eight colors around their centers are represented as 12-element vectors; accordingly, the functions which affect these corners, namely, L, R, F, B, and their inverses, must respecify the redundant characters post hoc. Overall, the function definitions are relatively simple, if not very elegant -- with the exception of Z, U, M, and D which capture their movements on the cube quite well. The compact data representation is adequate for display as is, but still begs for interpretation and consistency. Specifically, quarter-turns U, M, D, and their inverses cause horizontal movements only, whereas the others cause circular rotations of portions of the matrix. This can be difficult to perceive. Further, it might be better if these rotations were in a consistent direction (L and B are clockwise, but F and R are counter-clockwise) which would require changes in the x,y,x frame of reference. The speed of this representation is surprisingly fast (close to the best!), but it is a little demanding in its storage.

The Visual Representation is, as its name suggests, designed for viewing. One can see Rubik's Cube "in one look". The choice of data structure and resulting function definitions have little other redeeming qualities. The storage required is excessive, including more than 50% blanks. Also, since the bottom center is not represented, only one orientation move (Z) and its inverse are defined -- but very neatly, using two APL primitives. Most of the quarter-turns, however, are tedious piece-by-piece transformations -- rather inelegant use of APL. But the display does afford a good view of the cube, and the speed of the functions (especially Z) is slightly better than average.

The Vector Representation is simple, efficient, and general, but obscure in display and disconcertingly inelegant. It is the fastest of the representations and also requires the minimum storage for the cube (but space for the functions is considerable). The defined functions are simple, but not short, and certainly not APLish. The literal vector display is hard to parse (and a numeric vector wouldn't be much better), lacking in structure, but economical in screen space. And, its virtue of generality -- virtually any similar puzzle can be represented as a vector -- is offset by its failure to lend insights.

The 4-D Representation is as complex as its rank indicates but may appeal to APL experts. Its data structure is complete but one-third fat with blanks and awkward to display -- it is too vertical to fit on a screen -- and difficult to interpret. The functions are neat and fairly fast. Prospects for generalizing the representation, however, are discouraging, and insights unlikely.

The Nested Representation is perhaps the most natural. It models the inherent 3-by-3-by-3 structure of Rubik's Cube while using mnemonic literals for colors enclosed in each cubie's spatial position. It uses the same additional space as the 4-D Representation in order to represent the orientations of the cubies, which allows very concise function definitions. One particularly elegant aspect is found in the definitions of the orientation moves(also in the Numeric Representation): the permutation indices of dyadic transposition and indexing are identical -- lending insight into the relationship between translating and rotating colors. Its display is very good (for a 2-D view of a 3-D object). The potential for generalizing this representation (or its variant, with enclosed ragged vectors) to other similar puzzles is strong since any structure can be represented iconicly.

---

* This might suggest using a local frame of reference for each quarter-turn. Instead of the global axes (shown earlier) each quarter-turn could be clockwise with respect to its face. See (4). This does not, however, lead to better function definitions.

In summary, the table below offers the author's (admittedly somewhat subjective) rankings of the seven representations presented here according to the criteria outlined earlier (with the exception of manipulability, which is essentially the same for each representation):

	Complete	General	Simple	Concise	Display	Insight	Space	Speed
Numeric	1	7	7	3	4	2	3	6
Local	4	3	3	6	3	5	2	5
Composite	6	6	2	4	5	6	4	2
Visual	5	4	4	7	1	5	6	3
Vector	4	1	1	5	6	7	1	1
4-D	1	5	6	1	7	4	5	4
Nested	1	2	5	2	2	1	7	7

Toward Solving Rubik's Cube

For any of these representations, quarter-turns are performed simply by executing APL expressions such as:

```
CUBE←R CUBE
```
This performs a Right quarter-turn

and:

```
CUBE←R̲ CUBE
```
This performs an inverse Right turn (which restores the cube).

Sequences of quarter-turns are executed as follows:

```
CUBE←B̲ F L R CUBE
```
This performs R, then L, then F, then B̲.

and:

```
CUBE←R̲ L F̲ B CUBE
```
This performs the inverse sequence (which restores the cube).

Additional APL functions can be readily developed for manipulating Rubik's Cube. For instance, one can combine quarter-turns to create useful "tools" such as:

```
SWITCH: R̲ D R D F D̲ F̲ ⍵
```

Or, to express quarter-turn sequences left-to-right, define the following function:

```
DO: ⍎(⍵,' ',[1.5]⍺),'⍵'
```

Then define tools like this:

```
SWITCH: ⍵ DO 'F̲D̲FDRDR̲'

TWIST: ⍵ DO 'RD̲D̲RD̲RD̲RD̲D̲'

SNAP: ⍵ DO 'F̲MFM̲'

LIFT: ⍵ DO 'CDC̲D̲'

CYCLE: ⍵ DO 'MLLM̲L̲L̲'

FLIP: ⍵ DO 'MR̲MR̲MR̲R̲MR̲MR̲MR̲R̲'
```

Function SWITCH exchanges positions of the two bottom-front corners while preserving the positions of all other corners and the orientation of the top four corners as well. Function TWIST rotates three bottom corners 120 degrees clockwise while preserving the positions and orientations of all other corners. Function SNAP moves the back-right edge to the up-front position without affecting any corners. LIFT moves the down-right edge to up-front similarly. CYCLE exchanges positions of three edges in the middle layer. FLIP reverses the orientation of both edges on the right side without affecting the rest of the cube. All functions keep the centers in place.

These are useful tools for solving Rubik's Cube. See (4).

Some utility functions are helpful too:

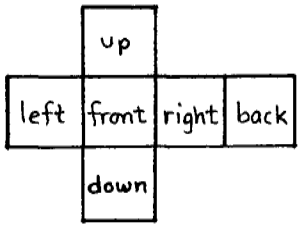
```
SCRAMBLE: ⍵ DO 17 RANDOM 'UU̲D̲D̲BB̲F̲FL̲L̲RR̲'

RANDOM: ⍵[?⍺⍴⍴⍵]

INVERSE: ⍵⍺[⍵+¯1+2×2|⍵←⍺⍳⍵]
```

SCRAMBLE mixes up the cube by doing 17 random quarter-turns (the minimum needed to guarantee reaching any possible state). INVERSE returns the names of inverses of ⍵ where ⍺ is the vector of all move names paired with their inverses.

It might also be desirable (for some representations) to have a function to display the cube perspicuously -- perhaps like this:



Further, it might be even more attractive to translate the internal data structure into actual colors (on an appropriate display device) and perhaps perspective drawing with controls for alternative viewing angles or selective display of portions of the cube.

Conclusion

Regardless of the representation one might choose -- it is as much a matter of taste as system resources -- beware the addiction of using APL to solve Rubik's Cube!

References

(1) Hofstadter, D. "The Magic Cube", in Metamagical Themas Dept., Scientific American, March 1981

(2) ---------- "Rubikmania", TIME, December 7, 1981

(3) Singmaster, D. "Computer Cubists" (list of names, addresses, and descriptions of computer programs) c/o Polytechnic of the South Bank, London, SE1 OAA, U.K.

(4) Peelle, H.A. "LEARN TO SOLVE RUBIK'S CUBE", December 1981

Acknowledgements

A number of people reviewed and offered constructive comments on this paper as it was being developed, including APL84 referees. A special mention is warranted for Jim Weigang, who, in response, himself wrote a paper extolling the virtues of a 4-dimensional representation. For a copy, interested readers may write him c/o Department of Mathematics and Statistics, University of Massachusetts, Amherst, MA 01003 USA.