Package 'cubing'

May 8, 2026

Version 1.0-5

Date 2018-04-24

Title Rubik's Cube Solving

Author Alec Stephenson.

Maintainer Alec Stephenson <alec_stephenson@hotmail.com>

Depends R (>= 3.0.0)

Imports grDevices, graphics, utils, stats, rgl

Description Functions for visualizing, animating, solving and analyzing the Rubik's cube. Includes data structures for solvable and unsolvable cubes, random moves and random state scrambles and cubes, 3D displays and animations using 'OpenGL', patterned cube generation, and lightweight solvers. See Rokicki, T. (2008) <doi:10.48550/arXiv.0803.3435> for the Kociemba solver.

License GPL-3

NeedsCompilation yes

Repository CRAN

Date/Publication 2018-04-23 12:29:32 UTC

Contents

`animate` *Create Cube Animations*

Description

Create cubing animation and record png frames using OpenGL.

Usage

```
animate(aCube, moves, fpt = 8, colvec = getOption("cubing.colors"), recolor = FALSE,
   bg = grey(0.8), rand.col = FALSE, size = 0.98, col.interior = grey(0.5),
  al.interior = 0.4, al.exterior = 1, start.delay = 2, move.delay = 0, rinit = 30,
  bbox = TRUE, bbcolor = "#333377", bbemission = "#333377", bbspecular = "#3333FF",
   bbshininess = 5, bbalpha = 0.5, movie = NULL, dir = file.path(getwd(), movie),
    verbose = TRUE, start.fdelay = fpt, end.fdelay = fpt, move.fdelay = 1, ...)
```

Arguments

`aCube`	Any cube object.
`moves`	A move sequence; either a single string or a character vector with one element per move. Can include URFDLBEMS face turns, wide turns and xyz rotations.
`fpt`	Number of frames per quarter turn. Must be a non-negative even integer. Whole cube rotations and wide moves use half the number of frames.
`colvec`	Vector of sticker colors. The default is the `cubing.colors` option.
`recolor`	If `TRUE`, previous rotations are ignored and therefore the cube is recolored when initially displayed.
`bg`	Background color.
`rand.col`	If `TRUE` then sticker colors are chosen at random and `colvec` is ignored.
`size`	Size of the individual cubies. Must be less than one. Values closer to one give cubes that look stickerless because the gap between cubies decreases. Smaller sizes give exploded cubes.
`col.interior`	Color of the cube interior.
`al.interior`	Alpha value of cube interior.
`al.exterior`	Alpha value of cube exterior.

start.delay	The delay in seconds added to the start.
move.delay	The delay in seconds between moves (turns or rotations).
rinit	The initial plot is rotated rinit degrees about the z-axis.
bbox	Use a bounding box?
bbcolor	Bounding box parameter.
bbemission	Bounding box parameter.
bbspecular	Bounding box parameter.
bbshininess	Bounding box parameter.
bbalpha	Bounding box parameter.
movie	If movie is a character string, then a png file is saved for every frame, using movie as the base file name. The following arguments are only relevant when movie is a character string.
dir	The directory where the png frames are stored. If the directory does not exist then it is created. By default the name of the directory is the same as the base filename and is located within the working directory.
verbose	Print progress on the saving of the png frames?
start.fdelay	The number of additional repeated frames added to the start.
end.fdelay	The number of additional repeated frames added to the end.
move.fdelay	The number of additional repeated frames between moves (turns or rotations).
...	Other parameters to be passed through to plotting functions.

Details

The move U3 represents three quarter turns in a clockwise direction, and so the animation is different to the quarter turn anti-clockwise move U', even though the resulting cube is the same. This similarly applies to the U3' and U moves, and to the half turn moves U2 and U2'. Wide turns can be denoted by lower case or w notation, so u2 and Uw2 are equivalent.

This function uses the R package **rgl** which is an interface to OpenGL. During the animation, the cube can be rotated using a mouse, and the rotations will be captured in the png frames if movie is not NULL. See the documentation for the **rgl** package to explore the large number of options available.

Following the production of the png frames, you can create movies or gifs using external utilities. One powerful command line utility is ffmpeg. ImageMagick is a software suite which performs similar conversions.

The plot3D function also uses the **rgl** package to produce interactive plots for individual cubes.

For a 2D version of the animate function, see plot.seqCubes.

See Also

plot3D.cube, plot.cube, plot.rotCubes, plot.seqCubes

Examples

```
scramb <- "D2F2UF2DR2DBL'BRULRUL2FL'U'"
aCube <- getMovesCube(scramb)
mvs <- "x2D'R'L2'U'FU'F'D'U'U'R'y'R'U'Ry'RU'R'U'RUR'U'R'U'F'UFRU'"
## Not run: animate(aCube, mvs, movie = "ChoWRSolve")
```

`comparison` *Logical Comparison for Cube Objects*

Description

Determine if two cubes are equal, accounting for recoloring and rotation.

Usage

```
## S3 method for class 'cube'
aCube == bCube
## S3 method for class 'cube'
all.equal(target, current, ...)
```

Arguments

`aCube`, `target`	Any cube object.
`bCube`, `current`	Any cube object.
`...`	Not used.

Details

Two cubes are defined to be equal via == if they are the same except perhaps for a recoloring. This means that the permutation and orientation components are the same but the spatial orientation component may be different.

Two cubes are defined to be the same via `all.equal` if and only if one cube is equal to the other following any of the 24 rotations of the whole cube (including the no rotation case).

For testing if two cubes are exactly identical, you can use the R function `identical`, however the cubes must be of the same type; either both cubieCubes or both stickerCubes.

Value

A logical value.

See Also

`is.solvable`, `is.solved`

Examples

```
aCube <- randCube()
bCube <- rotate(aCube, "y'")
aCube == bCube
```

`composition` *Composition Operators For Cube Objects*

Description

Composition operators for cubieCube objects.

Usage

```
aCube %v% bCube
aCube %e% bCube
aCube %c% bCube
```

Arguments

`aCube`	A cubieCube object.
`bCube`	A cubieCube object.

Details

Both arguments must be cubieCube objects, not stickerCube objects.

If A and B are cubes then A `%v%` B is the composition (or multiplication) of A and B. This means that if `a` and `b` are move sequences that produce A and B respectively from the solved cube `I`, then A `%v%` B is produced using the combined move sequence `ab` applied to `I`. Typically we just write AB for A `%v%` B.

Similarly to matrix multiplication, the operator `%v%` is associative but in general not commutative, with `AI` and `IA` both equal to A. Every 3x3x3 cube A has a unique inverse cube A' where AA' and A'A are both equal to `I`. The inverse cube can be calculated using the `invCube` function.

The `%e%` and `%c%` operators are similar to `%v%` but they compose only the edges and corners respectively. Use of these operators may create an unsolvable cube from two solvable cubes. They largely exist for internal reasons. See the help page on `cycleEdges` for details of their impact on solvability.

Value

A cubieCube object

See Also

`cycleEdges`, `getMovesCube`, `invCube`, `is.solvable`, `move`

Examples

```
aCube <- getCubieCube("Superflip")
bCube <- getCubieCube("EasyCheckerboard")
aCube %v% bCube
```

---

cubieCube *Create and Convert CubieCubes*

Description

Creates, converts and tests for cubieCube objects.

Usage

```
getCubieCube(pattern = c("Solved","Superflip","EasyCheckerboard","Wire","PlusMinus",
  "Tablecloth","Spiral","SpeedsolvingLogo","VerticalStripes","OppositeCorners",
  "Cross","UnionJack","CubeInTheCube","CubeInACubeInACube","Anaconda","Python",
  "BlackMamba","GreenMamba","FourSpots","SixSpots","Twister","Kilt","Tetris",
 "DontCrossLine","Hi","HiAllAround","AreYouHigh","CUAround","OrderInChaos","Quote",
  "MatchingPictures","3T","LooseStrap","ZZLine","Doubler","CheckerZigzag",
  "ExchangedDuckFeet","StripeDotSolved","Picnic","PercentSign","Mirror",
  "PlusMinusCheck","FacingCheckerboards","OppositeCheckerboards","4Plus2Dots",
  "Rockets","Slash","Pillars","TwistedDuckFeet","RonsCubeInACube","Headlights",
  "CrossingSnake","Cage","4Crosses","Pyraminx","EdgeTriangle","TwistedRings",
  "ExchangedRings","TwistedChickenFeet","ExchangedChickenFeet","CornerPyramid",
 "TwistedPeaks","ExchangedPeaks","SixTwoOne","YinYang","YanYing","HenrysSnake",
  "TwistedCorners","QuickMaths"))
cubieCube(string)
as.cubieCube(aCube)
is.cubieCube(aCube)
```

Arguments

pattern	A character string giving a pattern for the returned cube. Approximately seventy different patterns are available. The default pattern is the solved cube. The patterns and names are derived from the ruwix.com website.
string	A character string representing the color on each cube sticker. The string must contain only the letters URFLBD, representing the color on each face, and may contain any amount of white space. There must be 9 occurrences of each letter, or 8 occurrences if the centre stickers are omitted. A character vector can also be given instead of a character string, with one element for each letter. The sticker template can be displayed using the code at the end of the Examples section below.
aCube	Any object.

Details

The `is.cubieCube` function returns `TRUE` for cubieCube objects and `FALSE` otherwise. The `as.cubieCube` function converts a cube object to a cubieCube object and returns an error for other arguments.

The `getCubieCube` function creates cubieCube objects using known patterns. The `cubieCube` function creates cubieCube objects using colors entered by the user. For alternative ways of creating cubieCube objects, see `randCube` and `getMovesCube`.

A cubieCube is a list with five vector elements. The first four are cp ep co eo for the corner permuation, edge permuation, corner orientation, and edge orientation. The fifth, spor, tracks the fixed centres and therefore represents the spatial orientation. It exists to avoid recoloring the cube when plotting it after a rotation, middle slice move or wide move.

A stickerCube object does not hold information on spatial orientation, therefore if you convert a cubieCube to a stickerCube, and then convert back to a cubieCube, the spor vector will be reset to `1:6`.

The `cubieCube` function contains a large amount of bulletproofing to ensure the cube has valid cubies that are stickered correctly, but the cube may or may not be solvable. Both stickerCube and cubieCube objects are designed to hold both solvable and unsolvable cubes. You can test solvability with the `is.solvable` function.

Value

A logical value for `is.cubieCube`. A cubieCube object for all other functions.

See Also

`getMovesCube`, `is.solvable`, `randCube`, `stickerCube`

Examples

```
aCube <- getCubieCube("Wire")
bCube <- cubieCube("UUUUUUUUU RLLRRRLLR BBFFFFFBB DDDDDDDDD LRRLLLRRL FFBBBBBFF")
cCube <- cubieCube("FBBBUFRRB DUUFRUFFB DBRBFDUFL FRDDDLDDL UUFULLLLL RDRRBLURB")
identical(aCube, bCube)
is.cubieCube(aCube)

## Not run: plot(aCube)
## Not run: plot3D(aCube)
## Not run: plot(cCube)
## Not run: plot3D(cCube)

## Not run: plot(getCubieCube(), numbers = TRUE)
## Not run: plot(getCubieCube(), numbers = TRUE, blank = TRUE)
```

`cycle` *Cycle and Twist Cubies*

Description

Functions for cycling permuatations and altering orientations.

Usage

```
flipEdges(aCube, flip = 1:12)
twistCorners(aCube, clock = numeric(0), anti = numeric(0))
cycleEdges(aCube, cycle, right = TRUE, orient = TRUE)
cycleCorners(aCube, cycle, right = TRUE, orient = TRUE)
```

Arguments

`aCube`	A cubieCube object.
`flip`	An integer vector subset of `1:12` giving the set of edges to flip. By default, all edges are flipped.
`clock`	An integer vector subset of `1:8` giving the set of corners to twist clockwise. By default, none are twisted.
`anti`	An integer vector subset of `1:8` giving the set of corners to twist anti-clockwise. By default, none are twisted.
`cycle`	An integer vector representing the permutation cycle. See Details.
`right`	If `FALSE`, cycles to the left. See Details.
`orient`	Controls the orientation change for the permuation cycle. If `TRUE` each cubie is re-oriented according to the cubie it replaces, so the orientation vector does not change. If `FALSE` each cubie orientation is fixed so that the orientation vector also cycles.

Details

The `cycle` vector should be given according to mathematical cycle notation. For example, the vector `c(5,3,7)` means that the edge at position 5 moves to 3, 3 moves to 7 and 7 moves to 5. The vectors `c(3,7,5)` and `c(7,5,3)` are equivalent. The length of the vector is the length of the cycle, and so this example is a 3-cycle. If `right` is `FALSE` it cycles in the opposite direction, so 7 moves to 3, 3 moves to 5 and 5 moves to 7.

All of these functions can change the solvability of a cube. Solvability of a cube can be tested using the `is.solvable` function. For orientation solvability, the sum of the edge orientation vector must be even, and the sum of the corner orientation vector must be divisible by three. For the edge orientation to remain solvable, you must flip an even number of edges. For the corner orientation to remain solvable, the difference between the number of clockwise and anti-clockwise twists must be divisible by three. For example, the number of twists in each direction could be the same (a difference of zero), or you could have three clockwise twists and no anti-clockwise twists.

The sign of a permuation (even or odd) changes under a 2-cycle, which is just a swapping of two elements. In mathematical terminology this is called a transposition. A k-cycle can be written as k-1 transpositions, and therefore a k-cycle will change the sign of a permuatation if and only if k is even. So for a solvable cube to remain solvable, the length of `cycle` should be odd.

Two binary operators within the package that also impact solvability are `%e%` and `%c%`, which are composition operators for only edges and only corners respectively. If two cubes A and B are solvable, then A `%e%` B may be unsolvable because the edge and corner permuations may be of different sign; the orientations will always remain solvable. It is also possibe for A `%e%` B to be solvable but B `%e%` A to be unsolvable. The same reasoning applies to `%c%`.

In detail: if A and B have odd permutations, then both A `%e%` B and B `%e%` A become unsolvable. If A and B have even permutations, then both A `%e%` B and B `%e%` A remain solvable. If A has even and B has odd, then A `%e%` B is unsolvable but B `%e%` A is solvable.

Value

A cubieCube object.

See Also

`%v%`, `is.solvable`, `is.solved`, `invCube`

Examples

```
aCube <- getCubieCube("Superflip")
aCube <- flipEdges(aCube, flip = 1:12)
is.solved(aCube)
aCube <- twistCorners(aCube, clock = 3:6, anti = 2)
is.solvable(aCube)
aCube <- cycleEdges(aCube, c(2,10,5,6))
is.solvable(aCube)
```

`getMovesCube` *Create a Cube for a Move Sequence*

Description

Creates a cube that corresponds to a move sequence via post-multiplication.

Usage

```
getMovesCube(moves = character(0), cubie = TRUE)
```

Arguments

`moves`	A move sequence; either a single string or a character vector with one element per move. Can include URFDLB face turns. Cannot include rotations, middle slice moves or wide moves.
`cubie`	If `FALSE`, produce a stickerCube rather than a cubieCube.

Details

The cube object created by this function represents the application of the move sequence by means of post-multiplication with the composition operator `%v%`. If A is a cube and `m` is the move sequence, then A `%v%` `getMovesCube(m)` is the cube that results from applying the move sequence to A. In particular, if A is the solved state then this is just `getMovesCube(m)`. If `m` represents a scramble sequence, then `getMovesCube(m)` is the scrambled cube state.

The move sequence cannot include rotations, middle slice moves or wide moves because these cannot be expressed via post-multiplication. To implement these moves, see the `move` and `rotate` functions.

Value

A cube object.

See Also

`%v%`, `invMoves`, `move`, `rotate`, `slice`

Examples

```
scramb <- "D2F2UF2DR2DBL'BRULRUL2FL'U'"
aCube <- getMovesCube(scramb)
mvs <- "x2D'R'L2'U'FU'F'D'U'U'R'y'R'U'Ry'RU'R'U'RUR'U'R'U'F'UFRU'"
is.solved(move(aCube, mvs))
```

`getMovesPattern` *Get Moves for Patterned Cubes*

Description

Extracts the move sequence for a patterned cube.

Usage

```
getMovesPattern(pattern = c("Solved","Superflip","EasyCheckerboard","Wire","PlusMinus",
  "Tablecloth","Spiral","SpeedsolvingLogo","VerticalStripes","OppositeCorners",
  "Cross","UnionJack","CubeInTheCube","CubeInACubeInACube","Anaconda","Python",
  "BlackMamba","GreenMamba","FourSpots","SixSpots","Twister","Kilt","Tetris",
 "DontCrossLine","Hi","HiAllAround","AreYouHigh","CUAround","OrderInChaos","Quote",
  "MatchingPictures","3T","LooseStrap","ZZLine","Doubler","CheckerZigzag",
  "ExchangedDuckFeet","StripeDotSolved","Picnic","PercentSign","Mirror",
  "PlusMinusCheck","FacingCheckerboards","OppositeCheckerboards","4Plus2Dots",
  "Rockets","Slash","Pillars","TwistedDuckFeet","RonsCubeInACube","Headlights",
  "CrossingSnake","Cage","4Crosses","Pyraminx","EdgeTriangle","TwistedRings",
  "ExchangedRings","TwistedChickenFeet","ExchangedChickenFeet","CornerPyramid",
 "TwistedPeaks","ExchangedPeaks","SixTwoOne","YinYang","YanYing","HenrysSnake",
  "TwistedCorners","QuickMaths"))
```

Arguments

pattern	A character string giving a pattern for the corresponding cube. Approximately seventy different patterns are available. The default pattern is the solved cube, returning an empty character vector. The patterns and names are derived from the ruwix.com website.

Details

This function is mainly for internal use. It is used by the functions getCubieCube and getStickerCube to produce patterned cubes via getMovesCube. The returned move sequence contains only URFDLB face turns.

Value

A character vector of moves.

See Also

getCubieCube, getMovesCube, getStickerCube

Examples

```
getMovesPattern()
getMovesPattern("Solved")
getMovesPattern("Wire")
getMovesPattern("UnionJack")
```

---

invCube *Calculate Inverse Cube*

Description

Calculates the inverse of a cube.

Usage

```
invCube(aCube, edges = TRUE, corners = TRUE)
```

Arguments

aCube	A cubieCube object.
edges	If FALSE, the inverse is not taken for the edges.
corners	If FALSE, the inverse is not taken for the corners.

Details

Every 3x3x3 cube A has a unique inverse cube A' where AA' and A'A are both equal to the solved state. The cube A does not need to be solvable. Larger cubes do not have unique inverses because larger cubes have indistinct pieces in the centres.

One use of the inverse is to enable the `solver` function to generate moves towards a target state that is not the solved state. For an initial cube A and a target state B the solver is applied to B'A. The moves of the solver then represent post-multiplication by A'B, which when applied to A gives AA'B which is equal to B, the target state. Only B'A needs to be solvable; both A and B could be unsolvable.

A solvable cube will always remain solvable after the function `invCube` is applied, even if `edges` or `corners` is `FALSE`. This is because the sign (even or odd) of a permutation is the same as the sign of the inverse permutation.

Value

A cubieCube object.

See Also

`%v%`, `invMoves`, `is.solvable`, `solver`

Examples

```
aCube <- getCubieCube("Tetris")
is.solved(aCube %v% invCube(aCube))
is.solved(invCube(aCube) %v% aCube)

## Not run: plot(aCube)
## Not run: plot(invCube(aCube))
## Not run: plot3D(aCube)
## Not run: plot3D(invCube(aCube))
```

---

invMoves *Manipulate Move Sequences*

Description

Invert, mirror and rotate move sequences, and calculate the order of a move sequence.

Usage

```
invMoves(moves, revseq = TRUE, collapse = NULL)
mirMoves(moves, mirror = c("0","UD","DU","RL","LR","FB","BF"), collapse = NULL)
rotMoves(moves, rotation = c("0","x","x1","x3'","y","y1","y3'","z","z1","z3'",
"x2","x2'","y2","y2'","z2","z2'","x'","x3","x1'","y'","y3","y1'","z'","z3",
"z1'"), invrot = FALSE, collapse = NULL)
moveOrder(moves)
```

Arguments

moves	A move sequence. Either a character sting, which may include white space, or a character vector where each element is a single move. For moveOrder, can include only URFDLB face turns. For other functions, can include URFDLBEMS face turns, URFDLB wide turns and xyz rotations.
revseq	If FALSE, the move sequence is not reversed so that only the direction of the turns is altered.
mirror	The mirror to be used. The U/D mirror can be specified using the UD or DU character string. Similarly for R/L and F/B.
rotation	The rotation to be used.
invrot	Inverts the direction of the rotation.
collapse	If not NULL then the returned moves are output as a single string with collapse as the separator, rather than a character vector of moves. If collapse is the empty string then a single string with no separator is returned.

Details

For moveOrder an integer value is returned giving the order of the move sequence, which is the number of times it needs to be applied for the solved cube to return to its solved state. The largest order for any sequence is known to be 1260; for example, the order of "R U2 D' B D'" is 1260.

For other functions, a move sequence is returned. The returned move sequence will always use the canonical form for the turn notation: U not U1, U' not U1', and Uw not u for wide turns. However any form may be used for the input.

The Examples section below demonstrates the relationship between rotated move sequences. If the rotation is r and the rotated move sequence is m, then the move sequence rmr' is equivalent to the original. If invrot is TRUE, then this becomes r'mr.

Value

A character vector of moves, or a character string if collapse is not NULL.

See Also

move, invCube, rotate, scramble

Examples

```
mv <- "RB'y'F2MD'"
invMoves(mv)
mirMoves(mv, mirror = "RL")

iCube <- getCubieCube("TwistedChickenFeet")
rmv <- rotMoves(mv, rotation = "x")
aCube <- move(iCube, c("x", rmv, "x'"))
bCube <- move(iCube, mv)
identical(aCube, bCube)

moveOrder("RU2D'BD'")
```

move *Moving a Cube and Creating a Move Sequence*

Description

Applies moves to a cube and creates a move sequence.

Usage

```
move(aCube, moves, history = FALSE)
## S3 method for class 'seqCubes'
plot(x, initial = TRUE, which = 1:length(moves), ask = FALSE,
    colvec = getOption("cubing.colors"), recolor = FALSE, show.rot = TRUE,
    title = NULL, cex.title = 1, font.title = 2, ...)
```

Arguments

aCube	A cubieCube object.
moves	A move sequence; either a single string or a character vector with one element per move. Can include URFDLBEMS face turns, URFDLB wide turns and xyz rotations.
history	If TRUE the output is a list containing the initial cube state and every subsequent cube state created during the move sequence. If FALSE (the default) only the final cube state is returned.
x	An object produced by the move function when history is set to TRUE.
initial	Plot the initial cube state?
which	If only a subset of subsequent cube states are to be plotted, specify a subset of the numbers 1:length(moves).
ask	If TRUE, the user is asked before each plot.
colvec	Vector of sticker colors. The default is the cubing.colors option.
recolor	If TRUE, the spatial orientation is ignored and therefore the cube is recolored.
show.rot	Should rotation moves be plotted?
title	If specified as a character string, the first plot represents a title page with title in the centre. This is useful for putting a title page on multiple page graphing devices such as pdf.
cex.title	Size of title on title page.
font.title	Font of title on title page.
...	Other parameters to be passed through to plotting functions.

Details

By default move gives the result of applying the move sequence moves to the cube aCube. All rotations xyz, face turns URFDLBEMS, and wide moves URFDLB are allowed. Moves specifications such as U U1 U' U1' U2 U2' U3 U3' and rotation specifications such as x x1 x' x1' x2 x2' x3 x3' are all allowed. For wide moves, lower case lettering and w notation are both allowed, so u2 and Uw2 are equivalent. If moves is a single string, it may contain any amount of white space.

The definition of the E M S middle slice moves is respectively given by D'Uy' L'Rx' B'Fz'. In particular, the S slice direction is different to what you may find elsewhere; the definition used for S in this package is consistent with the rotation directions.

When history is TRUE a list is created of class seqCubes. The list contains cubieCube objects. The length of the list is the number of moves plus one, where the first element of the list is the original cube.

The plot.seqCubes function plots a list of class seqCubes. It can be regarded and a 2D version of the animate function. For permanent recording of the 2D plots for the move sequence, it is helpful to open a multiple page graphing device such as pdf. A pdf file can then be created and used as a flick book.

Value

For move, either a cubieCube object, or (if history is TRUE) an object of class seqCubes, which is a list where each element is a cubieCube. An attribute of the list stores the move sequence.

See Also

animate, getMovesCube, invMoves, plot.cube, rotate, slice

Examples

```
scramb <- "D2F2UF2DR2DBL'BRULRUL2FL'U'"
aCube <- getMovesCube(scramb)
mvs <- "x2D'R'L2'U'FU'F'D'U'U'R'y'R'U'Ry'RU'R'U'RUR'U'R'U'F'UFRU'"
is.solved(move(aCube, mvs))
sCubes <- move(aCube, mvs, history = TRUE)
## Not run: plot(sCubes, title = "SeungBeom Cho\nWorld Record Solve\n4.59")

## Not run: pdf("SeungBeomCho.pdf")
## Not run: plot(sCubes, title = "SeungBeom Cho\nWorld Record Solve\n4.59")
## Not run: dev.off()
```

---

plot.cube *Cube Object 2D Plot*

Description

Plots a 2D representation of a cube object.

Usage

```
## S3 method for class 'cube'
plot(x, colvec = getOption("cubing.colors"), recolor = FALSE,
   xlab = "", ylab = "", main = "", centres = TRUE, numbers = FALSE,
   text.size = 1, text.col = "black", rand.col = FALSE, blank = FALSE,
   ...)
```

Arguments

x	Any cube object.
colvec	Vector of sticker colors. The default is the cubing.colors option.
recolor	If TRUE, previous rotations are ignored and therefore the cube is recolored.
xlab, ylab	Plot labels.
main	Plot title.
centres	Add identifier text to the centre stickers.
numbers	Add identifier text to all stickers.
text.size	Size of text.
text.col	Color of text. The default is black but if you are using black stickers then purple is a good choice. White does not read well on yellow stickers.
rand.col	If TRUE then sticker colors are chosen at random and colvec is ignored.
blank	If TRUE then all colors are set to ghost white. This is designed to create a template plot for the sticker naming schemes.
...	Other parameters to be passed through to plotting functions.

See Also

animate, plot3D.cube, plot.rotCubes, plot.seqCubes

Examples

```
aCube <- getCubieCube("Superflip")
## Not run: plot(aCube)
```

---

plot3D.cube *Cube Object Interactive 3D Plot*

Description

Plots an interactive 3D representation of a cube object using OpenGL.

Usage

```
## S3 method for class 'cube'
plot3D(x, colvec = getOption("cubing.colors"), recolor = FALSE,
   bg = grey(0.8), rand.col = FALSE, size = 0.98, col.interior = grey(0.5),
   al.interior = 0.4, al.exterior = 1, rinit = 30, bbox = TRUE, bbcolor =
   "#333377", bbemission = "#333377", bbspecular = "#3333FF", bbshininess =
   5, bbalpha = 0.5, ...)
```

Arguments

x	Any cube object.
colvec	Vector of sticker colors. The default is the cubing.colors option.
recolor	If TRUE, previous rotations are ignored and therefore the cube is recolored.
bg	Background color.
rand.col	If TRUE then sticker colors are chosen at random and colvec is ignored.
size	Size of the individual cubies. Must be less than one. Values closer to one give cubes that look stickerless because the gap between cubies decreases. Smaller sizes give exploded cubes.
col.interior	Color of the cube interior.
al.interior	Alpha value of cube interior.
al.exterior	Alpha value of cube exterior.
rinit	The initial plot is rotated rinit degrees about the z-axis.
bbox	Use a bounding box?
bbcolor	Bounding box parameter.
bbemission	Bounding box parameter.
bbspecular	Bounding box parameter.
bbshininess	Bounding box parameter.
bbalpha	Bounding box parameter.
...	Other parameters to be passed through to plotting functions.

Details

This function uses the R package **rgl** which is an interface to OpenGL. The cube can be rotated using a mouse. See the documentation for the **rgl** package to explore the large number of options available.

The animate function also uses the **rgl** package to produce cubing animations.

See Also

animate, plot.cube, plot.rotCubes, plot.seqCubes

Examples

```
aCube <- getCubieCube("Superflip")
## Not run: plot3D(aCube)
```

`read.cubesolve` *Read Cube Solving Reconstructions*

Description

Reads cube solving scrambles and solutions for the cube solves website.

Usage

```
read.cubesolve(n, warn = FALSE)
```

Arguments

`n`	A single integer `n` giving the id for the solve.
`warn`	If `FALSE`, an error is returned when the id does not exist. If `TRUE`, a warning is given.

Details

The function reads from the html source of the www.cubesolv.es website, and may fail if the website subsequently changes.

Round brackets immediately followed by any single digit are expanded upon reading, for example (R U)4 becomes R U R U R U R U. All other round brackets are removed. Commutator and conjugate notation is not implemented, so square brackets cannot be used and a warning is given if they are found.

The website does not have automated checking and therefore a typographic error may lead to non-valid moves (or cubes that do not solve). This most commonly occurs when the space between two valid moves is accidentally omitted.

Value

A list containing the scramble, solution and description. The scramble and solution items are character vectors of moves. The description is a character string. If the scramble or solution (or both) is missing, then length zero vectors are returned for these items.

See Also

`getMovesCube`, `invMoves`, `is.solved`, `move`

Examples

```
## Not run: cho <- read.cubesolve(4995)
## Not run: aCube <- getMovesCube(cho$scramble)
## Not run: mv <- cho$solution
## Not run: is.solved(move(aCube, mv))
```

rotate *Perform Rotations, Wide Moves and Middle Slice Moves*

Description

Functions for rotating the whole cube and performing middle slice (E M S) and wide (Uw Rw Fw Dw Lw Bw) moves.

Usage

```
wide(aCube, wmv)
slice(aCube, smv)
rotate(aCube, rot)
```

Arguments

aCube	A cubieCube object.
smv	A single string giving a middle slice move. Must be one of E E1 E' E1' E2 E2' E3 E3' M M1 M' M1' M2 M2' M3 M3' S S1 S' S1' S2 S2' S3 S3'.
wmv	A single string giving a wide move. Lower case letters and w notation are both allowed. For turns on the U face must be one of Uw u Uw1 u1 Uw' u' Uw1' u1' Uw2 u2 Uw2' u2' Uw3 u3 Uw3' u3', and similarly for the R F D L B faces.
rot	A rotation. This can be specified using an integer or a character string. If an integer, must be a number between 1 and 24 representing one of the 24 possible rotations in space. The number 1 is the no rotation case. If a character string, must be one of 0 x x1 x' x1' x2 x2' x3 x3' y y1 y' y1' y2 y2' y3 y3' z z1 z' z1' z2 z2' z3 z3' where the 0 character string represents no rotation. Character strings correspond only to the 10 distinct rotations around the the x, y and z axes (including the no rotation case).

Details

These functions are used internally by the move function for performing rotations, wide moves and middle slice moves, but can also be called directly if you only need to perform a single move. In most circumstances it is better to use the move function.

The definition of the E M S middle slice moves is respectively given by D'Uy' L'Rx' B'Fz'. In particular, the S slice direction is different to what you may find elsewhere; the definition used for S in this package is consistent with the rotation directions.

Value

A cubieCube object.

See Also

move

Examples

```
aCube <- getCubieCube("HenrysSnake")
bCube <- slice(rotate(aCube, "z'"), "M2")
cCube <- move(aCube, "z'M2")
identical(bCube, cCube)

## Not run: plot(cCube)
## Not run: plot3D(cCube)
```

`rotations` *Create and Plot All Rotations of a Cube*

Description

Creates and plots all 24 possible whole cube rotations.

Usage

```
rotations(aCube)
## S3 method for class 'rotCubes'
plot(x, which = 1:24, ask = FALSE,  colvec =
  getOption("cubing.colors"), recolor = FALSE, ...)
```

Arguments

`aCube`	A cubieCube object.
`x`	An object produced by the `rotations` function.
`which`	If only a subset of the rotations is to be plotted, specify a subset of the numbers `1:24`.
`ask`	If `TRUE`, the user is asked before each plot.
`colvec`	Vector of sticker colors. The default is the `cubing.colors` option.
`recolor`	If `TRUE`, the spatial orientation is ignored and therefore the cube is recolored.
`...`	Other parameters to be passed through to plotting functions.

Details

There are 24 possible spatial orientations, including the original no rotation case. The `rotations` function produces all of these cubes as a list of class `rotCubes`, which can be subsequently plotted by `plot.rotCubes`.

The 24 cubes produced by `rotations` are all related to each other via some rotation and therefore they are all the same as defined by the `all.equal.cube` function. If the original cube has symmetric properties then the set of 24 may contain equivalent cubes as defined by the `==.cube` operator. In the case of the solved cube all 24 will be equivalent.

Value

For `rotations`, an object of class `rotCubes`, which is a list of length 24 where each element is a cubieCube.

See Also

`==.cube`, `all.equal.cube`, `rotate`

Examples

```
rCubes <- rotations(randCube())
all.equal(rCubes[[5]], rCubes[[10]])
rCubes[[5]] == rCubes[[10]]
## Not run: plot(rCubes)
```

---

scramble *Generate Random Cubes, Moves and Scrambles*

Description

Generate random cubes, random move sequences, and scrambles.

Usage

```
randCube(n = 1, cubie = TRUE, solvable = TRUE, drop = TRUE, spor = 1:6)
randMoves(n = 1, nm = 20, drop = TRUE)
scramble(n = 1, state = FALSE, nm = 20, drop = TRUE, type = c("KB", "ZT",
  "TF", "ZZ", "CFOP"), maxMoves = 24, bound = TRUE)
```

Arguments

`n`	Number of cubes, move sequences or scrambles to generate.
`cubie`	If `FALSE`, simulate stickerCubes rather than cubieCubes.
`solvable`	If `FALSE`, then solvable and unsolvable cubes can be simulated. See Details.
`drop`	If `FALSE`, then a list of one element is returned when `n` is equal to one.
`spor`	The spatial orientation vector that is added to each simulated cubieCube.
`nm`	The number of moves in the move sequence or the scramble. Ignored for `scramble` if `state` is `TRUE`.
`state`	If `FALSE` (the default), use a random moves scramble. If `TRUE`, use a random state scramble.
`type`	The type of solver used for the random state scramble. This and all following arguments are ignored unless `state` is `TRUE`.
`maxMoves`	Argument passed to the solver.
`bound`	Argument passed to the solver.

Details

If `solvable` is `TRUE`, the `randCube` function generates a solvable cube where every solvable state is equally likely. If `solvable` is `FALSE`, it generates a random cube where every obtainable physical construction is equally likely. The resulting cube may or may not be solvable: the chance of it being solvable is only 1 in 12.

The `randMoves` function generates a random move sequence using URFDLB face turns, restricted so that you cannot get two moves in a row on the same face, or three moves in a row on opposing faces. An alternative way of constructing a random cube is `getMovesCube(randMoves())`. If `nm` is 20 or more then every solvable state has a non-zero chance of occuring, but the states will not be equally likely to occur.

The `scramble` function generates random move scrambles by default (which just uses the `randMoves` function), but will generate random state scrambles when `state` is `TRUE`.

Value

A cube object or list of cube objects for `randCube`. A move sequence or list of move sequences for `randMoves` and `scramble`.

See Also

`getMovesCube`, `invCube`, `invMoves`, `is.solvable`, `solver`

Examples

```
randCube()
getMovesCube(randMoves())
sapply(randCube(20, solvable = FALSE), is.solvable)
randMoves(5, nm = 25)
scramble(nm = 17)
scramble(state = TRUE, type = "ZT", maxMoves = 24)
```

---

solvable *Solved and Solvability Tests for Cube Objects*

Description

Determine if a cube is solved or solvable, and calculate the sign of the corner and edge permutations.

Usage

```
is.solved(aCube, split = FALSE, co = TRUE, eo = TRUE)
is.solvable(aCube, split = FALSE)
parity(aCube)
```

Arguments

aCube	Any cube object.
split	Split output into logical vector? See Details.
co	If FALSE, ignore corner orientation.
eo	If FALSE, ignore edge orientation.

Details

The cubieCube and stickerCube objects contain Rubik's cubes that can be physically constructed from properly stickered cubies, but they are not necessarily solvable. These functions test for solved and solvable cubes. The parity function gives the permuation sign (+1 for even and -1 for odd) for the corner and edge permuations. For a cube to be solvable, the two signs must be the same.

For is.solved, a logical value for each separate permutation and orientation component will be given if split is TRUE.

For is.solvable, logical values corresponding to permuatation parity, edge orientation and corner orientation will be given if split is TRUE. The cube is only solvable if all three values are TRUE. The edge and corner orientation values correspond to the fact that if all but one edge (or corner) orientation is known, then the orientation of the final edge (or corner) must be fixed for the cube to be solvable. More precisely, the sum of the edge orientation vector must be even, and the sum of the corner orientation vector must be divisible by three.

Value

A logical value or vector for is.solved and is.solvable. A named integer vector of length two for parity.

See Also

==.cube, randCube, solver

Examples

```
aCube <- randCube()
is.solvable(aCube)
aCube <- randCube(solvable = FALSE)
is.solvable(aCube)
```

---

solver *Rubik's Cube Solver*

Description

Cube solvers to generate moves to a target cube state.

Usage

```
solver(aCube, tCube, type = c("KB", "ZT", "TF"), inv = FALSE,
    maxMoves = switch(type, KB = 24, ZT = 20, TF = 16), bound =
    TRUE, collapse = NULL, divide = FALSE, history = FALSE,
    verbose = FALSE)
```

Arguments

aCube	A cubieCube object giving the cube to be solved. If tCube is not specified then the object must be solvable.
tCube	A cubieCube object giving the target state. If not specified, then the target state is the solved state. See Details.
type	The type of solver used. KB is Kociemba. ZT is Zemdegs-Twist. TF is Twist-Flip.
inv	If TRUE the moves are inverted. For producing random state scrambles.
maxMoves	The maximum number of moves allowed for the search phases of the algorithm. The search algorithm may take a long time for smaller move requirements. The default value depends on the solver.
bound	By default the maxMoves value cannot be too small to avoid the search algorithm taking an excessively long time or never returning. If bound is set to FALSE this safety measure is removed, allowing any value of maxMoves to be specified. This is not recommended unless you know the cube can be solved within a small number of moves.
collapse	If not NULL then the returned moves are output as a single string with collapse as the separator, rather than a character vector of moves. If collapse is the empty string then a single string with no separator is returned.
divide	If TRUE, a period symbol is placed between the phases of the search algorithm.
history	If TRUE the solver returns a list where the second element gives a matrix object that provides information on the history of the search algorithm. Mainly used for debugging.
verbose	If TRUE print details of the status of the search. Mainly used for debugging.

Details

The solver produces a move sequence that brings aCube to either a solved state or to the target state tCube. If the target state is specified, then invCube(tCube) %v% aCube must be solvable, but the two cubes aCube and tCube could be unsolvable. See the help file on invCube for more details.

The KB algorithm is a 2-phase search. The ZT algorithm is similar but allows for twisting corners at the end to solve the corner orientation. The TF algorithm allows for twisting corners and flipping edges at the end to solve both corner and edge orientation. The twisting and flipping procedures are given as attributes in the returned object. If inv is TRUE, then they need to be performed at the start from the solved (or target) state.

The ZT and TF solvers may not produce a smaller move count than KB because the aim of the solver is to return any solution that consists of maxMoves moves or less. If smaller move counts are required then maxMoves should be specified.

These solvers are lightweight in the sense that they use small look-up tables (move tables and prune tables). If maxMoves is small then it can take a few seconds to find a solution.

The look-up tables for a solver are silently loaded into memory the first time the solver is used. The tables are hidden objects that are not visible to the user. If you wish to ensure that all tables are already loaded into memory (for example, if you want to do timing comparisons), then you can run each type of solver once, using any cube other than the solved (or target) state.

The solvers will never produce two moves in a row on the same face, but may produce three (or even four) moves in a row on opposite faces if this coincides with the break between the two search phases. They cannot produce three moves in a row on opposite faces within the same search phase. This behaviour is a design choice in order to minimize second phase solutions that are rejected due to move sequences across the phase break.

Value

A character vector of moves, or a character string if collapse is not NULL. For ZT the vector (or string) has a twist attribute. For TF the vector (or string) has twist and flip attributes.

If history is TRUE, then a list of length two is returned where the second element is a matrix that provides information on the history of the search algorithm.

See Also

getMovesCube, invCube, invMoves, is.solvable, scramble

Examples

```
aCube <- getCubieCube("EasyCheckerboard")
## Not run: plot(aCube)
## Not run: plot3D(aCube)
mvs <- solver(aCube, type = "KB")
is.solved(aCube %v% getMovesCube(mvs))
```

---

stickerCube *Create and Convert StickerCubes*

Description

Creates, converts and tests for stickerCube objects.

Usage

```
getStickerCube(pattern = c("Solved","Superflip","EasyCheckerboard","Wire","PlusMinus",
  "Tablecloth","Spiral","SpeedsolvingLogo","VerticalStripes","OppositeCorners",
  "Cross","UnionJack","CubeInTheCube","CubeInACubeInACube","Anaconda","Python",
  "BlackMamba","GreenMamba","FourSpots","SixSpots","Twister","Kilt","Tetris",
 "DontCrossLine","Hi","HiAllAround","AreYouHigh","CUAround","OrderInChaos","Quote",
  "MatchingPictures","3T","LooseStrap","ZZLine","Doubler","CheckerZigzag",
  "ExchangedDuckFeet","StripeDotSolved","Picnic","PercentSign","Mirror",
```

```
  "PlusMinusCheck","FacingCheckerboards","OppositeCheckerboards","4Plus2Dots",
  "Rockets","Slash","Pillars","TwistedDuckFeet","RonsCubeInACube","Headlights",
  "CrossingSnake","Cage","4Crosses","Pyraminx","EdgeTriangle","TwistedRings",
  "ExchangedRings","TwistedChickenFeet","ExchangedChickenFeet","CornerPyramid",
 "TwistedPeaks","ExchangedPeaks","SixTwoOne","YinYang","YanYing","HenrysSnake",
  "TwistedCorners","QuickMaths"))
stickerCube(string)
as.stickerCube(aCube)
is.stickerCube(aCube)
```

Arguments

pattern	A character string giving a pattern for the returned cube. Approximately seventy different patterns are available. The default pattern is the solved cube. The patterns and names are derived from the ruwix.com website.
string	A character string representing the color on each cube sticker. The string must contain only the letters URFLBD, representing the color on each face, and may contain any amount of white space. There must be 9 occurrences of each letter, or 8 occurrences if the centre stickers are omitted. A character vector can also be given instead of a character string, with one element for each letter. The sticker template can be displayed using the code at the end of the Examples section below.
aCube	Any object.

Details

The `is.stickerCube` function returns `TRUE` for stickerCube objects and `FALSE` otherwise. The `as.stickerCube` function converts a cube object to a stickerCube object and returns an error for other arguments.

The `getStickerCube` function creates stickerCube objects using known patterns. The `stickerCube` function creates stickerCube objects using colors entered by the user. For alternative ways of creating stickerCube objects, see `randCube` and `getMovesCube`.

A `stickerCube` is a named character vector of length 54 where each element is one of the six letters URFLBD. The element named U5 is always equal to the character string U as this represents a centre sticker. The elements named R5 F5 L5 B5 D5 are similarly fixed.

The `stickerCube` function contains a large amount of bulletproofing to ensure the cube has valid cubies that are stickered correctly, but the cube may or may not be solvable. Both stickerCube and cubieCube objects are designed to hold both solvable and unsolvable cubes. You can test solvability with the `is.solvable` function.

Value

A logical value for `is.stickerCube`. A stickerCube object for all other functions.

See Also

`getMovesCube`, `is.solvable`, `randCube`, `cubieCube`

Examples

```
aCube <- getStickerCube("Wire")
bCube <- stickerCube("UUUUUUUUU RLLRRRLLR BBFFFFFBBD DDDDDDDDL RRLLLRRLF FBBBBBFF")
cCube <- stickerCube("FBBBUFRRB DUUFRUFFB DBRBFDUFLF RDDDLDDLU UFULLLLLR DRRBLURB")
identical(aCube, bCube)
is.stickerCube(aCube)

## Not run: plot(aCube)
## Not run: plot3D(aCube)
## Not run: plot(cCube)
## Not run: plot3D(cCube)

## Not run: plot(getStickerCube(), numbers = TRUE)
## Not run: plot(getStickerCube(), numbers = TRUE, blank = TRUE)
```

Index