Rubik examples

www.ctan.org/tex-archives/macros/latex/contrib/rubik/rubikexamples.pdf *

RWD Nickalls † & A Syropoulos ‡

25 February, 2018 (Rubik bundle v5.0)

Contents

*This file is part of the Rubik bundle v5.0. To generate this file, use the following command:

```
$ pdflatex --shell-escape rubikexamples.tex
```

†email: dick@nickalls.org
‡email: asyropoulos@yahoo.com

1 Preliminaries

These examples were generated using the TEX Rubik bundle [1] v5.0. They assume some familiarity with the four packages RUBIKCUBE, RUBIKROTATION, RUBIKPATTERNS and RUBIKTWOCUBE. For documentation see the following files:

```
rubikcube.pdf           3x3x3 documentation
rubikrotation.pdf       commands for implementing rotation sequences
rubikrotationPL.pdf     documentation of rubikrotation.pl (Perl program)
rubikpatterns.pdf       small database of well-known 3x3x3 configurations
rubikpatternsLIST.pdf   shows all configurations in the rubikpatterns database
rubiktwocube.pdf        2x2x2 documentation
```

This file requires the following packages: `tikz, rubikcube, rubikrotation, rubikpatterns, rubiktwocube`; note that the `tikz` package must be loaded *before* the `rubikcube` package.

This file needs to be run using the `--shell-escape` command-line option; for example:

```
pdflatex  --shell-escape  rubikexample.tex
```

This is because nearly all the examples make use of the `\RubikRotation` command, which calls the Perl script `rubikrotation.pl`. If you do forget to use the `--shell-escape` command-line switch, the file will still run, but all the cubes will remain in the initial unprocessed configuration [2].

1.1 3x3x3 vs 2x2x2 command names

The introduction of the RUBIKTWOCUBE package (Rubik bundle v5) has necessitated a few command name changes in order to avoid confusion (see the RUBIKCUBE package documentation § 3.1 for details). Consequently, command names containing the word '`Rubik`' denote a 3x3x3 cube action; similarly those containing the word '`Two`' denote a 2x2x2 cube action.

1.2 Environments

When using the Rubik bundle one sometimes needs to be mindful of the various LATEX environments in which Rubik commands are placed (e.g., the figure, minipage and TikZ picture environments), since these environments restrict the actions of commands they contain to the particular environment. The `\ShowCube` command is also relevant here, since it is a minipage-wrapper for the TikZ picture environment. Only Rubik `\Draw..` commands and TikZ commands (e.g., `\draw` and `\node`) actually need to be inside a TikZ picture environment, and hence inside a `\ShowCube` command.

This issue arises because the Rubik bundle allows you to create figures showing different stages during a sequence of rotations. Consequently the effects of commands executed inside an environment (especially commands which determine the colour-state or rotations), may not be apparent to subsequent commands outside that particular environment. See Example 3 for an illustration of how to handle environments.

[1] `http://www.ctan.org/pkg/rubik`

[2] As a quick check, run this file (`rubikexamples.tex`) in a separate directory and look at Figure 1; if this appears as a 'solved' cube then your system has failed to call the Perl script `rubikrotation.pl` correctly for some reason. The most likely causes of this are (a) failure to invoke the `--shell-escape` command-line option correctly, or (b) failure to use a configuration file correctly (ie your system can't find the Perl program)—search the `.log` file to see whether the file `rubikrotation.pl` has been used.

2 RubikCube examples (3x3x3)

2.1 Sixspot

In Figure 1 we show the so-called "sixspot" configuration, generated from a solved cube using the rotation sequence **U**, **D'**, **R**, **L'**, **F**, **B'**, **U**, **D'**.

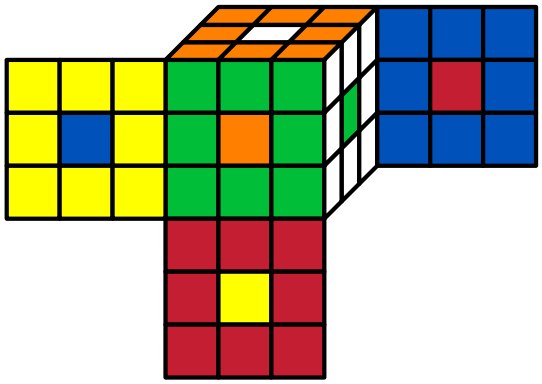


Figure 1: The 'sixspot' configuration.

Creating a macro to hold a rotation sequence greatly facilitates their use, as follows:

```
\newcommand{\sixspot}{[sixspot],U,Dp,R,Lp,F,Bp,U,Dp}
```

We can now process this sequence using its macro name as an argument for the `\RubikRotation` command. The code used for the above Figure uses the `\ShowCube{}{}{}` command for which `#1` is the minipage width, `#2` is the tikzpicture scale factor (0–1), and `#3` can include RUBIKCUBE package `\Draw..` commands, and any commands which are valid for use in a TikZ `tikzpicture` environment. The code for the above figure is as follows:

```
\begin{figure}[hbt]
   \centering
   \RubikCubeSolvedWY
   \RubikRotation{\sixspot}
   \ShowCube{7cm}{0.7}{\DrawRubikCubeSF}
\caption{...}
\end{figure}
```

Note that the sixspot sequence is a so-called 'order 3' sequence, which means that running the 'sixspot' sequence 3 times returns the cube to its original 'solved' state. The command for processing it three times is `\RubikRotation[3]{\sixspot}`.

Note that the semi-flat form of the cube here is generated by the `\DrawRubikCubeSF` command, where the terminal SF denotes the Semi-Flat form. To draw it completely flat use a terminal F, e.g., `\DrawRubikCubeF` (see Figure 2).

2.1.1 Log-file extract

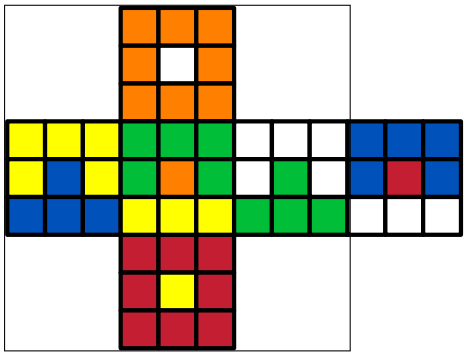
Users may find it instructive to inspect the the log-file and follow the dynamic interaction between LaTeX and the Perl script. This is easy to follow, since output by `rubikrotation.sty` is prefixed by 3 dashes (`---`), while output by the Perl script is prefixed by 3 dots (`...`). Search for the keyword 'Example', as this is written to the log-file at the start of each example.

The following is part of the log-file extract associated with Example 1.

```
 ---Example (sixspot)

---TeX process----------------------------
---script = rubikrotation.sty v5.0 (2018/02/25)
---NEW rotation command
---command = RubikRotation[1]{[SixSpot],U,Dp,R,Lp,F,Bp,U,Dp,<(8q*, 8f*)>}
---writing current cube state to file rubikstate.dat
\openout7 = `rubikstate.dat'.

\ourRRcounter=\count134
---CALLing Perl script (rubikrotation.pl)
 runsystem(perl $HOME/RubikTeX/PERLstuff/dtx-rubikrotation/rubikrotation.pl
  -i rubikstate.dat -o rubikstateNEW.dat)...executed.

---inputting NEW datafile (data written by Perl script)
(./rubikstateNEW.dat

 ...PERL process.................................
 ...script = rubikrotation.pl v5.0 (25 February 2018)
 ...reading the current cube state (from File: rubikstate.dat)
 ...
 ...command = cubesize,three
 ...cube = THREEcube
 ...
 ......up,W,W,W,W,W,W,W,W,W
 ....down,Y,Y,Y,Y,Y,Y,Y,Y,Y
 ....left,B,B,B,B,B,B,B,B,B
 ...right,G,G,G,G,G,G,G,G,G
 ...front,O,O,O,O,O,O,O,O,O
 ....back,R,R,R,R,R,R,R,R,R
 ...
 ...rotation keyword
 ...checking state of cube
 ...cubiesum = 54 (Red=9, Or=9, Ye=9, Gr=9, Bl=9, Wh=9, X=0)
 ...
 ...command = rotation,[SixSpot],U,Dp,R,Lp,F,Bp,U,Dp,<(8q*, 8f*)>
 ...dataline = rotation,[SixSpot],U,Dp,R,Lp,F,Bp,U,Dp,<(8q*; 8f*)>
 ...[SixSpot] is a label OK
 ...rotation U, OK
 ...rotation Dp, OK
 ...rotation R, OK
 ...rotation Lp, OK
 ...rotation F, OK
 ...rotation Bp, OK
 ...rotation U, OK
 ...rotation Dp, OK
 ...writing new cube state to file rubikstateNEW.dat
 ...SequenceName = SixSpot
 ...SequenceInfo = (8q*; 8f*)
 ...SequenceShort = [SixSpot],U,Dp,R,Lp,F,Bp,U,Dp
 ...SequenceLong = U,Dp,R,Lp,F,Bp,U,Dp
 )
```

2.2 ShowErrors

In this example we demonstrate the use of the `\ShowErrors` command[3], which places a copy of the Perl output file `rubikstateERRORS.dat` underneath the graphic so you can see a list of all the errors, if any. Note that this example is similar to the previous one except that we have introduced several errors—e.g., bad minipage width, typos, as well as some animals—into the rotation sequence). It is important to note that the `\ShowErrors` command must be placed *after* the TikZ picture environment (i.e., in this case after the `\ShowCube` command), or even at the end of the document. Note that full details of all errors are also included in the .log file (search for `*ERR`).



```
%% ShowErrors (rubikstateERRORS.dat)
%% --------------------------------
*ERR cmd= rotation,[sixspot],U,Dp,R,Lp,F,Bp,U,Dpppp,cat,dog
*ERR      Dpppp  -- code not known ? typo or missing comma
*ERR      cat  -- code not known ? typo or missing comma
*ERR      dog  -- code not known ? typo or missing comma
```

Figure 2: The same 'sixspot' sequence of rotations as shown in Example 1, but now with some errors (wrong minipage width, typos and some animals!) in the rotation sequence (it *should* be just `U,Dp,R,Lp,F,Bp,U,Dp`).

In this example we have used the F version of the `\ShowCube` command (`\ShowCubeF`) which places an fbox around the image so you can see the extent of any white space etc. This reveals that the set minipage-width (4.5cm) in the `\ShowCubeF` command—see code below—is too narrow: it should be 5cm (10×0.5) to just include the whole image (i.e., $10\times$ the TikZ scale-factor in this case). Once fixed, we can remove the F from the `\ShowCubeF` command. Note also that only '`\Draw...`' commands really need to be inside the TikZ picture environment (i.e., inside the `\ShowCube` command). The above figure was generated by the following code.

```
\RubikCubeSolvedWY
\RubikRotation{[sixspot],U,Dp,R,Lp,F,Bp,U,Dpppp,cat,dog}
\begin{figure}[hbt]
  \centering
  \ShowCubeF{4.5cm}{0.5}{\DrawRubikCubeF}
  \ShowErrors
\caption{....}
\end{figure}
```

Even if the `\ShowErrors` command is not used, it is always a good idea to check the file `rubikstateERRORS.dat` after a LaTeX run, since this file will also reveal any errors.

[3] The `\ShowErrors` command replaces the earlier `\ShowRubikErrors` command which is now deprecated. However, the earlier command is still supported (for now) for backward compatibility.

2.3 Environments

In this example we highlight the fact that Rubik commands used inside a LaTeX environment remain local to that environment, and how this can sometimes be problematic. Commands whose reach is meant to be more global need to be executed outside such environments, where they can implement global colour settings, which will then be accessible to subsequent `\Draw..` commands.

Since we are drawing images, this is primarily an issue with the `minipage`, `figure`, and TikZ picture environments. Consequently, it is generally best when drawing a sequence of cubes to reserve the TikZ picture environment only for `\Draw..` commands and TikZ commands. Importantly, this also applies to our main display tool the `\ShowCube` command, since this is a minipage-wrapper for the TikZ picture environment (see the RUBIKCUBE package documentation).

In this example the first cube (9a) uses a `\RubikCubeSolvedWY` command *inside* a `\ShowCube` environment. However, if we now perform the rotation **R** (using the command `\RubikRotation{R}`) this results in a quite unexpected effect on cube (9b) (and obviously not correct). This is because the action of the initial `\RubikCubeSolved` command (setting a new colour-state) is not visible outside its `\ShowCube` environment, and hence the subsequent `\RubikRotation{R}` command is unaware of this earlier attempt to update the global colour-state information. It turns out that the internal colour-state was actually last updated following the action of the `\RubikRotation{[sixspot],....}` command used in Example 2, being the last colour-state command executed *outside* an environment (a `figure` environment in that example). Consequently, the strange cube 9b is not what we expected.

9a 9b

R

```
\usepackage{tikz,rubikcube,rubikrotation}
\newcommand{\cubenumber}[1]{\strut\raisebox{1cm}{#1}}
...
\cubenumber{9a}%
\ShowCube{2cm}{0.5}{%
    \RubikCubeSolvedWY%
    \DrawRubikCubeRU%
     }%
\quad\Rubik{R}%
\RubikRotation{R}%
\cubenumber{9b}%
\ShowCube{2cm}{0.5}{\DrawRubikCubeRU}%
```

If we now bring the `\RubikCubeSolvedWY` command out of the `\ShowCube` environment and place it *before* the `\ShowCube` command (see cubes 9c, 9d), then its colour 'state' information becomes globally accessible (i.e., colour-state gets updated) and therefore gets used by the subsequent `\RubikRotation{R}` command, and hence the cube (9d) is now rendered correctly.

9c 9d

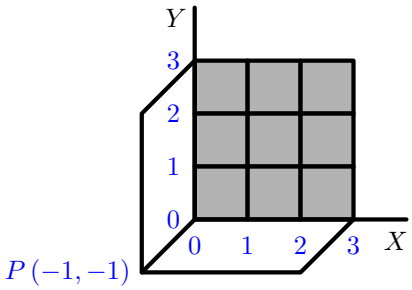
R

```
\cubenumber{9c}%
\RubikCubeSolvedWY%
\ShowCube{2cm}{0.5}{\DrawRubikCubeRU}%
\quad\Rubik{R}%
\RubikRotation{R}%
\cubenumber{9d}%
\ShowCube{2cm}{0.5}{\DrawRubikCubeRU}%
\end{minipage}
```

Note that the horizontal space occupied by `9d` is almost exactly the same as `\quad`, and so we don't need to include any additional space before the `\cubenumber{9d}` command.

2.4 Coordinates

For all cubes the origin of coordinates is defined as the bottom left corner of the FRONT face. Consequently, it is easy to determine the coordinates of points and hence draw lines, circles, and place lettering or other objects using the standard TikZ `\draw..` and `\node..` commands. Note that for convenience point P (bottom back left corner) is designed to be $(-1,-1)$ on the 2D view. (The following diagram is Fig 1 from the RUBIKCUBE package documentation).



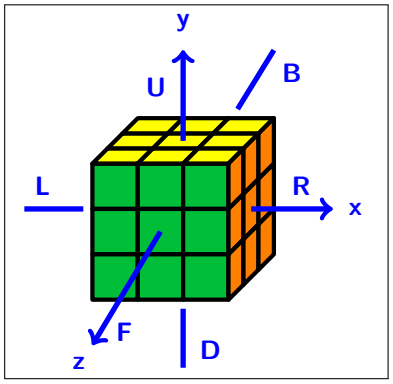
The code for the figure is given below.

We draw everything in the `\ShowCube` environment; the FRONT face in grey (colour code = `X`) using the Rubik command `\DrawRubikFlatFront`, and then draw all the lines and text using standard TikZ commands. The correct minipage-width argument (5.6cm) for the `\ShowCube` command is determined by trial-and-error, using the 'fbox' form of the command (`\ShowCubeF`), and then the 'F' is removed (→ `\ShowCube`). In order to avoid confusion, all Rubik commands start with a capital letter (e.g., `\Draw..`), while all TikZ commands start with a lower-case letter (e.g., `\draw..`).

```
\begin{figure}[hbt]
\centering
\RubikFaceFrontAll{X}% X = default non-colour (grey)
\ShowCube{5.6cm}{0.7}{%
   \DrawRubikFaceFront
   \draw[line join=round,line cap=round,ultra thick] (0,0) -- (0,4);% Yaxis
   \draw[line join=round,line cap=round,ultra thick] (0,0) -- (4,0);% Xaxis
   \node (Ylabel) at (-0.35,  3.8)  {$Y$};
   \node (Xlabel) at ( 3.8,  -0.4)  {$X$};
   %% outline Left and Down faces
   \draw[line join=round,line cap=round,ultra thick]%
        (0,3) -- (-1,2) -- (-1,-1) -- (2,-1) -- (3,0);
   \draw[line join=round,line cap=round,ultra thick]%
        (-1,-1) -- (0, 0);
   \node (Y0) at (-0.4, 0)  [blue]{$0$};
   \node (Y1) at (-0.4, 1)  [blue]{$1$};
   \node (Y2) at (-0.4, 2)  [blue]{$2$};
   \node (Y3) at (-0.4, 3)  [blue]{$3$};
   \node (X0) at (0, -0.5)  [blue]{$0$};
   \node (X1) at (1, -0.5)  [blue]{$1$};
   \node (X2) at (2, -0.5)  [blue]{$2$};
   \node (X3) at (3, -0.5)  [blue]{$3$};
   \node (P) at (-2.4, -1)  [blue]{$P\,(-1,-1)$};
   }
\end{figure}
```

2.5 Face notation

The following diagram is Fig 2 from the RUBIKCUBE package documentation.



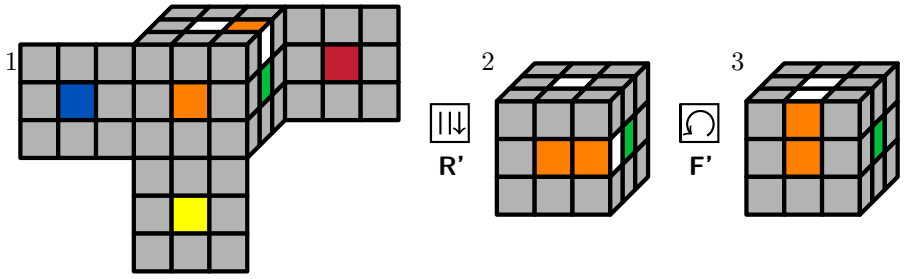
The code for the figure is as follows (the origin $(0, 0)$ is at the bottom left corner of the FRONT (green) face).

```
\begin{figure}[htb]
\centering%
\RubikFaceUpAll{Y}
\RubikFaceFrontAll{G}
\RubikFaceRightAll{O}
\ShowCubeF{5cm}{0.6}{%
  \DrawRubikCubeRU%
  %% Right
  \draw[line width=2pt,color=blue,->] (3.5,2) -- (5.3, 2);
  \node (R) at (4.6, 2.5)  [blue]{\textbf{\textsf{R}}};
  \node (x) at (5.8, 2)  [blue]{\textbf{\textsf{x}}};
  %%Left
  \draw[line width=2pt,color=blue] (-0.2,2) -- (-1.5, 2);
  \node (L) at (-1.1, 2.5)  [blue]{\textbf{\textsf{L}}};
  %%Up
  \draw[line width=2pt,color=blue,->] (2, 3.5) -- (2, 5.5);
  \node (U) at (1.4, 4.7)  [blue]{\textbf{\textsf{U}}};
  \node (y) at (2, 6.1)  [blue]{\textbf{\textsf{y}}};
  %%Down
  \draw[line width=2pt,color=blue] (2, -0.2) -- (2, -1.5);
  \node (D) at (2.6, -1.1)  [blue]{\textbf{\textsf{D}}};
  %%Front
  \draw[line width=2pt,color=blue,->] (1.5, 1.5) -- (0, -1);
  \node (F) at (0.7, -0.7)  [blue]{\textbf{\textsf{F}}};
  \node (z) at (-0.3, -1.4)  [blue]{\textbf{\textsf{z}}};
  %%Back
  \draw[line width=2pt,color=blue] (3.2, 4.2) -- (4, 5.5);
  \node (B) at (4.4, 5)  [blue]{\textbf{\textsf{B}}};
  }
\end{figure}
```

2.6 Grey cube

When explaining elementary layer-1 moves, it can be useful to use the 'grey cube' (`\RubikCubeGreyWY`) —White opposite Yellow— as this sets up only the central cubie on each face. We have shown the first cube here in Semi-Flat (SF) mode simply to show how the grey cube is configured (note that a WB version (`\RubikCubeGreyWB`) and an 'all-grey' version (`\RubikCubeAllGrey`) are also available. Note that all these grey cube commands will also accept the word 'gray' (to be consistent with TikZ).

In this example, we show how to position a single 'flipped' white/orange edge cubie in the top layer.



The code for the figure is given below. After setting up the first cube, we then just use the `\RubikRotation` command to generate the remaining cubes. The colours are coded as follows: R (red), O (orange), Y (yellow), G (green), B (blue), W (white), and X (grey).

```
\usepackage{tikz,rubikcube,rubikrotation}
\newcommand{\cubenumber}[1]{\strut\raisebox{1cm}{#1}}
...
\begin{figure}[hbt]
\centering
% set up the first cube
\RubikCubeGreyWY%
\RubikFaceUp{X}{X}{X}%
            {X}{W}{O}%
            {X}{X}{X}%

\RubikFaceRight{X}{W}{X}
               {X}{G}{X}
               {X}{X}{X}
\cubenumber{1}%
\ShowCube{5cm}{0.5}{\DrawRubikCubeSF}%
%
\quad\Rubik{Rp}\RubikRotation{Rp}
\cubenumber{2}%
\ShowCube{2cm}{0.5}{\DrawRubikCube}%
%
\quad\Rubik{Fp}\RubikRotation{Fp}
\cubenumber{3}%
\ShowCube{2cm}{0.5}{\DrawRubikCube}%
\end{figure}
```

2.7 Scramble a cube

In this example we use the `\RubikRotation` command to scramble a 'solved' Rubik cube via a sequence of 120 random rotations, using the following command (the details of the process can be seen in the `.log` file):

```
\RubikRotation{random,120}
```

On this occasion we draw the cube Flat (F) using the command `\DrawRubikCubeF`. In this example, we also make use of the `\SaveRubikState{}` command to save the configuration (state) displayed here in Figure 3 to a file (`rubikexampfig3.tex`) using `\SaveRubikState{rubikexampfig3.tex}`, so we can display the same cube configuration later but in a different format (we show it again in the following example (Example 2.8). Note that since we are using a random sequence, it follows that each time this file is run not only will a visually different cube be generated, but the same state will be shown both here and in Example 2.8.

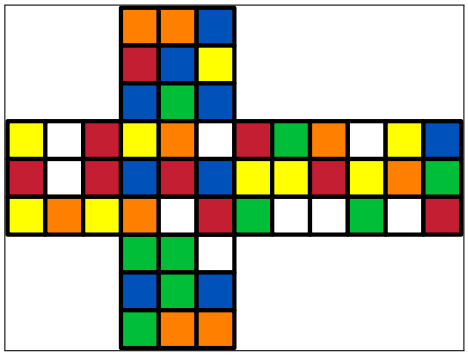


Figure 3: This shows a cube generated by 120 random rotations

```
\usepackage{tikz,rubikcube,rubikrotation}
...
\begin{figure}[hbt]
  \centering\RubikCubeSolvedWY
  \RubikRotation{random,120}
  \SaveRubikState{rubikexampfig3.tex}
  \ShowCubeF{6cm}{0.5}{\DrawRubikCubeF}
\caption{....}
\end{figure}
```

Q: How do we determine the minipage-width (7.2cm) in the `\ShowCube` command?

A: The object is 12 cubie squares wide. Since the TikZ scale-factor argument of the `\ShowCube` command (cms/unit length) in this case is set to 0.5, then the true width of the image will be $12 \times 0.5 = 6$ cm. Note that here we have used the `\ShowCubeF` command and so we can see that this is correct. Changing the scale-factor will change the image size and hence a new width argument will be required to just fit the image.

Note that in this particular case (where there is only a single image in the 'figure' environment), since the `\ShowCube` command places the image (in a TikZ picture environment) centrally inside a minipage, the image will in fact be centrally placed in the `\textwidth` provided the image is *smaller* than the fbox—i.e., if we used instead a minipage-width of, say, 12 cm the image would still appear centred in the `\textwidth` in this case. However, when there are several images in the 'figure', then the spacing may appear strange unless each image closely fits its own minipage-width etc. It is often useful, therefore, to check the size of the fbox (using the `\ShowCubeF` command) as we have done here.

2.8 SaveRubikState

In this example we display a cube having the same colour state as that shown in the previous example (Example 2.7) —see Figure 3. The cube's colour state was saved from the previous example using the command `\SaveRubikState{rubikexampfig3.tex}`, and then input here using the command `\input{rubikexampfig3.tex}`. These commands therefore allow the state of a previous cube to be saved to a file, and then displayed again later in a different format.

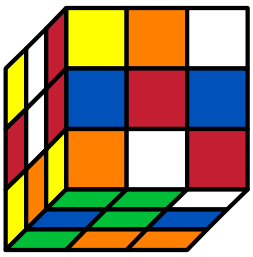


Figure 4: This shows a Rubik cube in exactly the same colour state as the one shown in Figure 3, but here displayed as a cube seen from the LD (Left-Down) viewpoint.

```
\usepackage{tikz,rubikcube,rubikrotation}
...
\begin{figure}[hbt]
  \centering
  \input{rubikexampfig3.tex}
  \ShowCube{4cm}{0.8}{\DrawRubikCubeLD}
  \parbox{0.7\textwidth}{%
      \caption{....}%
  }%
\end{figure}
```

NOTE: The command `\SaveRubikState{}` is part of the RUBIKROTATION package; see the documentation for further details.

2.9 Series of cubes

Here we show a convenient way of displaying a series of small cubes showing the action of a sequence of rotations (**U**, **R**, **F**) on a solved cube.

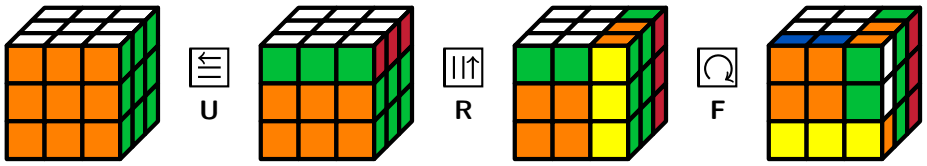
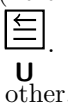


Figure 5: The rotations **U**, **R**, **F** on a solved cube.

The code for the above sequence is as follows:

```
\usepackage{tikz,rubikcube,rubikrotation}
...
\begin{figure}[hbt]
  \centering%
  \RubikCubeSolvedWY%
  \ShowCube{2cm}{0.5}{\DrawRubikCubeRU}%
  \quad\Rubik{U}\quad%
  \RubikRotation{U}%
  \ShowCube{2cm}{0.5}{\DrawRubikCubeRU}%
  \quad\Rubik{R}\quad%
  \RubikRotation{R}%
  \ShowCube{2cm}{0.5}{\DrawRubikCubeRU}%
  \quad\Rubik{F}\quad%
  \RubikRotation{F}%
  \ShowCube{2cm}{0.5}{\DrawRubikCubeRU}%
\caption{The rotations \rr{U}, \rr{R}, \rr{F}\ on a solved cube.}
\end{figure}
```

NOTE 1: We start with the White-opposite-Yellow (WY) solved cube, using the command `\RubikCubeSolvedWY`. A White-opposite-Blue (WB) solved cube is available as `\RubikCubeSolvedWB`.

NOTE 2: There are four different commands for typesetting 3x3x3 rotation codes in various formats (a slightly different set of four commands are used for the equivalent 2x2x2 rotation codes). In this (3x3x3) example, we have used two of them; for example:
`\rr{U}` → **U** (here the `\rr{}` stands for 'rubik rotation'), and also
`\Rubik{U}` →  . This command typesets both the hieroglyph and the rotation code vertically one above the other. The other two commands are as follows:
`\rrh{U}` →  (here the `\rrh{}` stands for 'rubik rotation hieroglyph'), and also
`\textRubik{U}` → **U**  . This command typesets both the rotation code and the hieroglyph on the same line one after the other.

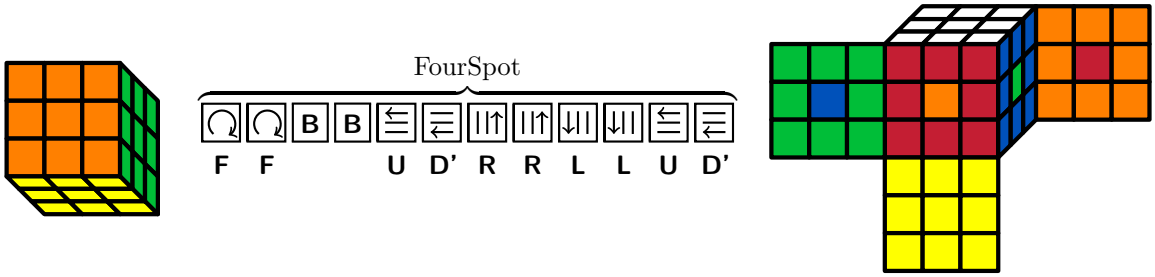
2.10 Rotation sequence

We now explore using the named Rubik cube rotation sequences and associated patterns available in the RUBIKPATTERNS package—a small macro database (see the RUBIKPATTERNS documentation, and also its companion file `rubikpatternsLIST.pdf`). Having the sequences available as macros is very convenient since (a) it avoids making errors when typing them out, and (b) allows the easy application of software tools.

A Rubik pattern is the configuration generated by a sequence of rotations (or 'moves') from some initial starting configuration (typically a 'solved' configuration). For example, FourSpot is a well known pattern which we can generate from a solved Rubik cube using the macro `\FourSpot`; it is defined in the RUBIKPATTERNS package as follows:

```
\newcommand{\FourSpot}{[FourSpot],F2,B2,U,Dp,R2,L2,U,Dp,<(12q*, 8f*)>}
\newcommand{\fourspot}{\FourSpot}
```

Note that for convenience the macros names in the RUBIKPATTERNS package are defined in both upper and lower-case (i.e., the macros `\FourSpot` and `\fourspot` are identical). The following figure shows the FourSpot sequence and pattern.



The code for the above figure is as follows:

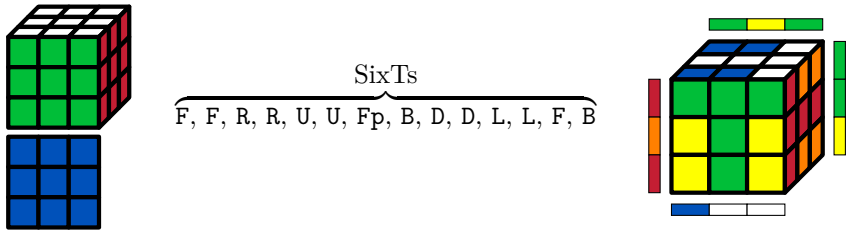
```
\usepackage{tikz,rubikcube,rubikrotation,rubikpatterns}
...
\noindent%
\RubikCubeSolvedWY%
\ShowCube{2cm}{0.5}{\DrawRubikCubeRD}
\RubikRotation{\FourSpot}
\quad%
\SequenceBraceA{FourSpot}{%
                          \ShowSequence{}{\Rubik}{\SequenceLong}%
                         }
\quad%
\ShowCube{5cm}{0.5}{\DrawRubikCubeSF}
```

Note that we have spread the code slightly here in order to emphasise that the `\ShowSequence` command is being used as an argument for the `\SequenceBraceA` command (the 'A' in the command `\SequenceBraceA` denotes that the annotation is placed Above the sequence.) We have used a solved cube with the WY (White opposite Yellow) configuration (`\RubikCubeSolvedWY`). The first cube is drawn from the RD (Right-Down) viewpoint (`\DrawRubikCubeRD`). The second cube is drawn from the SF (Semi-Flat) viewpoint (`\DrawRubikCubeSF`) so we can see all the faces.

Note that we execute the `\RubikRotation{}` command *before* showing the sequence itself. This is because the Perl program (CALLed by `\RubikRotation`) allocates the rotation sequence to the variable `\SequenceLong` which we then display using the `\ShowSequence` command (see the RUBIKROTATION documentation for details).

2.11 SixTs

A more interesting cube pattern is the SixTs configuration (from the RUBIKPATTERNS package), which we now show in a slightly different way (adding an extra face and also some cube sidebars), as follows:



This time we have started with a solved cube having the WB configuration (White opposite Blue), which is generated using the command `\RubikCubeSolvedWB` (we have added the DOWN face (blue) below to reveal the colour of this face—see note below).

The rotation sequence is in 'long-format' (expanded into separate rotations), comma-separated and space, typewriter font, using the command `\ShowSequence{,\ }{\texttt}{\SequenceLong}`.

The final image shows the sixT cube together with the main cube sidebars indicating the colours of the adjacent facelets, using the `\DrawRubikCubeSidebarXX` commands. The code for the above figure is as follows:

```
\RubikCubeSolvedWB
\ShowCube{1.6cm}{0.4}{%
            \DrawRubikCubeRU%
            \DrawRubikFlatDown{0}{-3.3}%
            }
\RubikRotation{\SixTs}
\quad\SequenceBraceA{SixTs}{%
            \ShowSequence{,\ }{\texttt}{\SequenceLong}%
            }
\quad\ShowCube{2cm}{0.5}{
            \DrawRubikCubeRU
            \DrawRubikCubeSidebarFL{RU}
            \DrawRubikCubeSidebarFD{RU}
            \DrawRubikCubeSidebarBR{RU}
            \DrawRubikCubeSidebarBU{RU}
            }
```

Notes

1. We have drawn the DOWN face of the first cube using the command `\DrawRubikFlatDown{0}{-3.3}` where the two arguments are the x and y coordinates of the *bottom left* corner of the DOWN face (blue); these (x, y) arguments allow us to position the blue 'flat' face in relation to the cube. Note that the grid origin of a cube image (on the page) coincides with the *bottom left* corner of the FRONT face (green in this case).

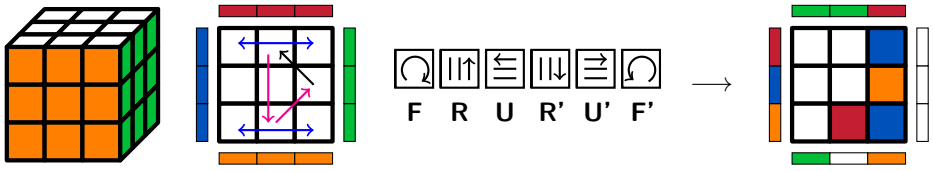
2. The first image is really just 4 units wide. This is because the 2D width of the *side* face (red) is designed to measure 1 unit wide in the oblique view (similarly, the 2D height of the *top* face also measures just 1 unit). Consequently, since the TikZ scale factor used is 0.4, then the (minimum) width argument for its `\ShowCube{}{}{}` command is $4 \times 0.4 = 1.6$cm., hence we have `\ShowCube{1.6cm}{0.4}{...}`.

2.12 Cycle three edges (flip two)

The following example illustrates a sequence used for cycling three edges, which is often used in generating the 'cross' formation while solving the final layer. However, it is a useful exercise to perform this sequence on a solved cube, as this reveals quite clearly its rather complex actions (it also swaps two pairs of corners).

The black arrow (no flip) and magenta arrows (flip) indicate the UP face edge cubies which are cycled anti-clockwise by the sequence **F**,**R**,**U**,**R'**,**U'**,**F'**. The blue arrows indicate the associated 'collateral damage' actions (two pairs of corner cubies swap positions); depending on the circumstances, these other actions may be an acceptable cost since they can be fixed at a later stage. (This diagram is from Section 13 in the RUBIKCUBE package documentation).

Note that we include the UP-face sidebars to reveal the remaining facelet colours of the top layer (using the command `\DrawRubikFaceUpSide`) so we can see how the sequence makes the cubies rotate and turn as they move.



The code for the figure is as follows

```
\newcommand{\CycleThreeEdgesFlipTwo}{F,R,U,Rp,Up,Fp}%
 ...
 ...
 \RubikCubeSolved%
 \ShowCube{2cm}{0.4}{\DrawRubikCubeRU}%
 \quad\ShowCube{2.3cm}{0.4}{%
    \DrawRubikFaceUpSide%
    \draw[thick,->,color=magenta] (1.5,0.5) -- (2.4, 1.4);
    \draw[thick,->] (2.5,1.5) -- (1.6, 2.4);
    \draw[thick,->,color=magenta] (1.3, 2.3) -- (1.3, 0.5);
    \draw[thick,<->,  color=blue] (0.5,2.6) -- (2.5, 2.6);
    \draw[thick,<->,  color=blue] (0.5,0.3) -- (2.5, 0.3);
    }%
 \RubikRotation{\CycleThreeEdgesFlipTwo}%
 \quad\ShowSequence{}{\Rubik}{\SequenceLong}\quad$\longrightarrow$\quad%
 \ShowCube{2.3cm}{0.4}{\DrawRubikFaceUpSide}%
```

Note that the TikZ `\draw` commands (for drawing the arrows on the cube face) start with a lower-case letter, and also require a terminal semicolon.

2.13 Superflip

Once you can solve Rubik's cube, then an interesting exercise is to generate the so-called 'superflip' configuration, in which all the corners are correctly solved, while all the edges are flipped [4].

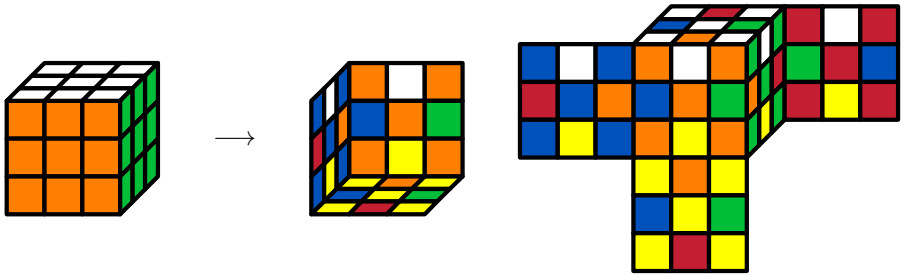


Figure 6: Two representations of the superflip configuration.

A superflip sequence converts the solved cube on the left into the form on the right, using the command `\RubikRotation{\superflip}`. The code for the above figure is shown below (the `\superflip` macro used here is from the RUBIKPATTERNS package).

```
\usepackage{tikz,rubikcube,rubikrotation,rubikpatterns}
...
\begin{figure}[hbt]
  \centering
  \RubikCubeSolvedWY%
  \ShowCube{2cm}{0.5}{\DrawRubikCubeRU}%
  \qquad$\longrightarrow$\qquad%
  \RubikRotation{\superflip}%
  \ShowCube{2cm}{0.5}{\DrawRubikCubeLD}
  \qquad%
  \ShowCube{5cm}{0.5}{\DrawRubikCubeSF}
\caption{...}
\end{figure}
```

The following superflip sequence [5] has just 20 HTM rotations (Half Turn Metric: counting 180 degree turns as just one 'rotation'). Note that the RUBIKPATTERNS package contains this particular superflip sequence as the macro `\superflip` (see `rubikpatterns.pdf`). Consequently the code `\ShowSequence{,}{\large\texttt}{\superflip}`, will typeset the sequence as follows:

`[Superflip],Dp,R2,Fp,D2,F2,U2,Lp,R,Dp,R2,B,F,Rp,U2,Lp,F2,Rp,U2,Rp,Up,<(20f*)>`

Note that for convenience, the RUBIKPATTERNS package includes the sequence name (in square brackets) as the first element of the associated macro. This is possible since the contents of a comma-separated square bracket are not actioned as a rotation by the `\RubikRotation` command.

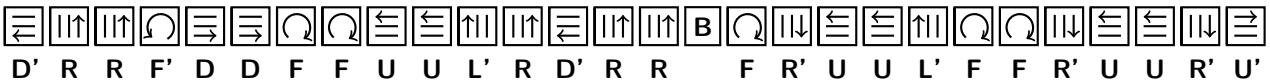
[4] See the 'superflip' entry in *Wikipedia*, and also the Kociemba website (`www.kociemba.org/cube.htm`); particularly the page `http://kociemba.org/math/oh.htm`

[5] This particular superflip sequence (in the RUBIKPATTERNS package) is due to Reid (1995); for details see the RUBIKPATTERNS package documentation, and also `http://kociemba.org/math/oh.htm`. Another 20-move superflip sequence (due to H Kociemba), is designated as K32466 in `www.nickalls.org/dick/papers/tex/RUBIK20moves.zip`.

Next we present the same superflip sequence but without commas and in the form of hieroglyphs, using the `\Rubik` font, for which we require the 'expanded' `\SequenceLong` form (since the 'short form' includes trailing digits—see Section 11 in the RUBIKCUBE documentation), using the following code:

```
\usepackage{tikz,rubikcube,rubikrotation,rubikpatterns}
...
\RubikCubeSolvedWY
\RubikRotation{\superflip}
\noindent\strut\hspace{-8mm}\ShowSequence{}{\Rubik}{\SequenceLong}
```

which gives

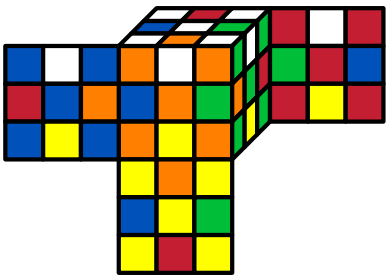


Equivalent sequence

Interestingly, the superflip sequence is actually equivalent to $\left\{\left(\boxed{\uparrow\uparrow}\,\boxed{\leftarrow\equiv}\right)4,\ [\mathbf{x}],\ [\mathbf{y'}]\right\}3$ (Harris (2008), p151). Furthermore we can readily demonstrate this, as we can process this novel form of the sequence using some useful features of the `\RubikRotation` command, as follows:

```
\RubikCubeSolvedWF
\RubikRotation[3]{[superflip],(Rm,U)4,x,yp,<Harris (2008), p151>}
\ShowCube{4cm}{0.4}{\DrawRubikCubeSF}
```

which generates the following



which is exactly the same configuration as before. Note that to do this we made use of the 'repeat' option `[3]` as well as the `(Rmp,Up)4` 'repeat-block' in the argument of the `\RubikRotation` command above.

2.14 Inverse sequence

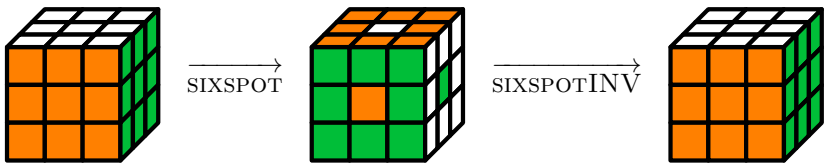
Generating the inverse of a Rubik sequence involves (a) reversing the order of the sequence, and (b) inverting each rotation in the sequence (see Sections 5.1 and 5.1.1 in the RUBIKROTATION package documentation).

From the grey-cube example (2.6) we saw that the sixspot sequence is: `U,Dp,R,Lp,F,Bp,U,Dp`; its inverse is therefore readily determined as `D,Up,B,Fp,L,Rp,D,Up`. Note that this is easy to check since the sequence generated by the `\RubikRotation` command is held by the macro `\SequenceLong`. For example, the output of the following commands

```
\fbox{\strut\rule{0pt}{10pt}%
      The inverse of the sixspot sequence is:
      \RubikRotation{\sixspot,<inverse>}
      \ShowSequence{,}{\texttt}{\SequenceLong}.
    }
```

is | The inverse of the sixspot sequence is: D,Up,B,Fp,L,Rp,D,Up. |

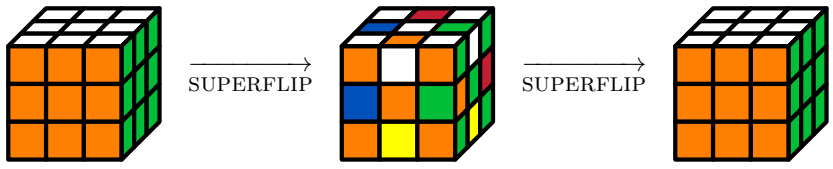
A sequence and its inverse will annihilate each other when applied consecutively. For example, in the following figure we start with a solved cube and apply the sixspot sequence. Applying the inverse of the sixspot sequence then results in the solved cube configuration again.



The code for the above figure is as follows:

```
\newcommand{\sixspotarrow}{$\quad\overrightarrow{\strut\textsc{sixspot}}\quad$}
\newcommand{\sixspotINVarrow}{$\quad\overrightarrow{\strut\textsc{sixspotINV}}\quad$}
...
\RubikCubeSolvedWY\ShowCube{2cm}{0.5}{\DrawRubikCubeRU}%
\RubikRotation{\sixspot}%
   \sixspotarrow%
   \ShowCube{2cm}{0.5}{\DrawRubikCubeRU}%
\RubikRotation{\sixspot,<inverse>}%
   \sixspotINVarrow%
   \ShowCube{2cm}{0.5}{\DrawRubikCubeRU}
```

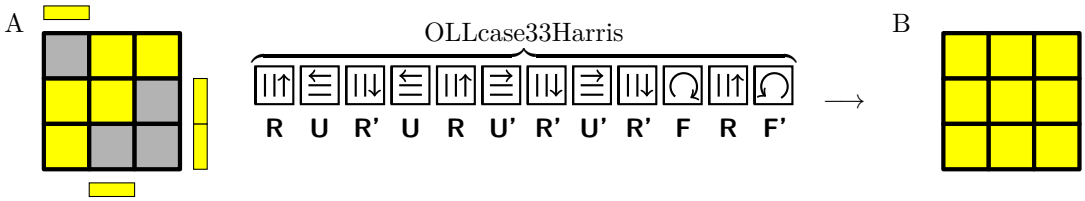
A significant property of the superflip configuration is that it is its own inverse. Consequently we can achieve a similar result simply by applying the superflip sequence *twice in succession*, as follows:



2.15 OLL example

Here we illustrate the use of the `\NoSidebar` command (see the RUBIKCUBE package documentation) to generate nice displays of so-called OLL (Orientate Lower Layer) configurations, which are typically rendered using the yellow face.

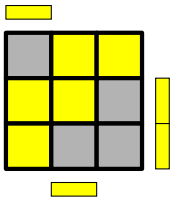
For example, the following sequence shows an efficient algorithm (sequence) for 'solving' the OLL configuration shown in cube A (listed as OLL case 33 in Harris (2008, page 61)[6] and also on the Harris OLL webpage `http://www.cubestation/3x3x3/cfop/oll/ollprintablepage.php`).



The code for the above figure is as follows (note that the `\NoSidebar{X}` command is placed inside the `\ShowCube` environment in order to limit its action).

```
\newcommand{\cubenumber}[1]{\strut\raisebox{1cm}{#1}}
\newcommand{\CaseXXXIII}{[caseHarris33],R,U,Rp,U,R,Up,Rp,Up,Rp,F,R,Fp}
%
\RubikCubeGreyAll
\RubikFaceUp{X}{Y}{Y}
            {Y}{Y}{X}
            {Y}{X}{X}
\RubikFaceBack  XXY XXXXXX
\RubikFaceLeft  XXX XXXXXX
\RubikFaceRight YYX XXXXXX
\RubikFaceFront XYX XXXXXX
\cubenumber{A}%
\ShowCube{2.6cm}{0.6}{\NoSidebar{X}\DrawRubikFaceUpSide}%
\RubikRotation{\CaseXXXIII}%
\quad\SequenceBraceA{OLLcase33Harris}{%
                \ShowSequence{}{\Rubik}{\CaseXXXIII}%
                }%
\quad$\longrightarrow$\quad%
\cubenumber{B}%
\ShowCube{2.6cm}{0.6}{\NoSidebar{X}\DrawRubikFaceUpSide}%
```

Note that since we originally defined the OLL case 33 algorithm (Harris 2008) as a macro, it follows that we could instead have recovered the OLL 33 configuration (cube A above) using the inverse algorithm, as follows:



```
\RubikCubeGreyAll
\RubikFaceUpAll{Y}
\RubikRotation{<inverse>,\CaseXXXIII}
\ShowCube{2.6cm}{0.6}{\NoSidebar{X}\DrawRubikFaceUS}
```

Note that in this last example we have used the more convenient short-hand US (UpSide) version of the `\DrawRubikFaceUpSide` command, namely `\DrawRubikFaceUS`.

[6] Harris D (2008). *Speedsolving the cube*, (Stirling Publishing Co, New York) (`www.stirlingpublishing.com`) [covers 2x2x2, 3x3x3, 4x4x4, 5x5x5 cubes].

2.16 Cube patterns

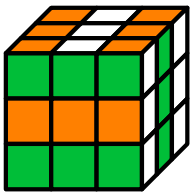
We now look at how the `\RubikRotation` command can be used to investigate cube patterns, which are typically shown only as a regular cube. For example, consider the following interesting sequence `L' F' R' L U D R' D' L' B' U2 L' U' R L U' F2 R'` (Harris 2008, page 150), which is presented as a cube with orthogonal stripes (see figure A). In order to show the pattern using the Rubik bundle, it is often useful to first express the sequence as a macro. In this particular case we need to adjust the Harris (2008) notation slightly to make it compatible (ie, insert commas, change trailing 'primes' into a trailing 'p'), give it a suitable name (say, `[StripesOrtho]`), and, for completion, include the citation in an 'infoblock' (<...>), as follows:

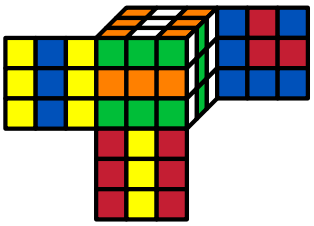
```
\newcommand{\StripesOrtho}{%
   [StripesOrtho],Lp,Fp,Rp,L,U,D,Rp,Dp,Lp,Bp,U2,Lp,Up,R,L,Up,F2,Rp,<Harris 2008, p150>}
```

We can now process it simply by using its macro-name as the argument for the `\RubikRotation` command; for example, the following code displays it as figure A.

```
\RubikCubeSolved
\RubikRotation{\StripesOrtho}
\ShowCube{2.6cm}{0.6}{\DrawRubikCubeRU}
```

At first sight this looks like a nice example of orthogonal stripes, but if we look at all the faces by rendering the cube in a Semi-Flat (SF) view (using the command `\DrawRubikCubeSF`) —see figure B— this reveals that unfortunately only five faces actually have a center stripe.

A 

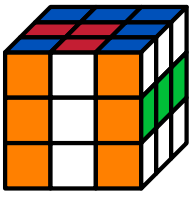
B 

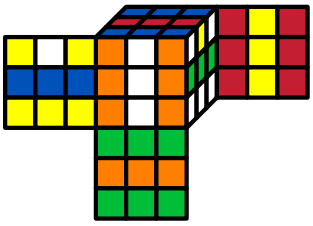
So the question is: can we improve on this and create a cube with a fully orthogonal pattern, by swapping the relevant red/yellow and blue/yellow edge cubies in the back layer? Since it is impossible to swap just two edge cubies within the same layer in a 3x3x3 cube without changing any other cubies, the answer is no. However, we are able to cycle cleanly three edge cubies within one face, and a single cycle could at least create the final stripe with only minimal collateral damage.

For example, if we rotate the whole cube using the rotation denoted as **Rc'** (= `\rr{Rcp}`)[7] to bring the BACK face to the UP position (as shown in figure C), we can then cycle the three UP edge cubies UL, UR, UF clockwise using the following 'CycleThreeUpEdges' sequence:

```
\newcommand{\CycleThreeUpEdges}{F2,U,Rm,Up,Up,Lm,U,Fp2}
```

We can combine these two actions using the command `\RubikRotation{Rcp,\CycleThreeUpEdges}`, which will then align the three red facelets in the UP face and form the final stripe (see figure D).

C 

D 

The downside, of course, is that a yellow and a white facelet are also swapped in the top layer, but at least we now have a cube with six mutually adjacent orthogonal central stripes!

[7] There are several equivalent notations for this: **Rc'** ≡ **x'** ≡ **CR'**; the c and C denote a whole cube rotation.

3 TwoCube examples (2x2x2)

3.1 Commands & notation

Regarding the 2x2x2 cube as a 'TwoCube' (instead of a RubikCube) makes the 2x2x2 commands very intuitive (assuming you are familiar with the 3x3x3 commands), since they are mostly obtained just by exchanging the word 'Rubik' in 3x3x3 commands for the word 'Two'. For example,

```
3x3x3          →          2x2x2
\RubikCubeSolvedWY → \TwoCubeSolvedWY
\RubikRotation     →   \TwoRotation
\SaveRubikState    →   \SaveTwoState
```

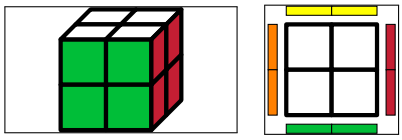
Note that although the 2x2x2 cube does not have any middle slices, one can still actually use the 3x3x3 commands `\RubikRotation` and `\SaveRubikState` in the usual way (because the corner cubies are still processed in the usual way). However, in the interests of consistency both of these commands have equivalent 'Two' versions: namely `\TwoRotation` and `\SaveTwoState`.

Of course there are considerably fewer rotation codes and hieroglyphs associated with the TwoCube (since there are no middle slices), and their commands also mirror the 3x3x3 equivalents in the RUBIKCUBE package; i.e., `Rubik` → `Two` and `r` → `t`. Thus the changes associated with the four key commands can be summarised as follows :

```
3x3x3   2x2x2
\rr{} → \tr{}
\rrh{} → \trh{}
\Rubik{} → \Two{}
\textRubik{} → \textTwo{}
```

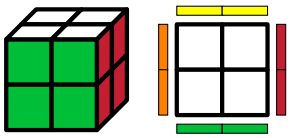
3.2 Visualising white-space

This example shows how the F version of the `\ShowCube{}{}{}` command can be useful for revealing excess white space.



```
\TwoCubeSolvedWB%
\ShowCubeF{3cm}{0.6}{\DrawTwoCubeRU}%
\quad%
\ShowCubeF{1.7cm}{0.5}{\DrawTwoFaceUS}%
```

Both images above have an F-box drawn around them (so we can see the extent of any white space) using the F version of the `\ShowCube`{*width*}{*scale-factor*}{} command, namely `\ShowCubeF{}{}{}`. Clearly the `3cm` width we have used for the first cube above is too much, and so in the second pair of images below we have reduced this width for the first image to its correct value of `1.6cm`, and have removed the F from the `\ShowCubeF` command accordingly.



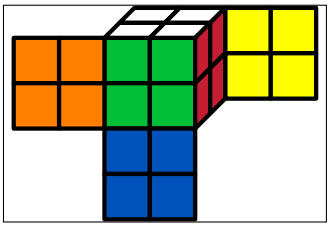
```
\TwoCubeSolvedWB%
\ShowCube{1.6cm}{0.6}{\DrawTwoCubeRU}%
\quad%
\ShowCube{1.7cm}{0.5}{\DrawTwoFaceUS}%
```

Thus use of the F version of the the `ShowCube` command allows the user to easily determine—by trial & error—the correct minipage width for a particular cube.

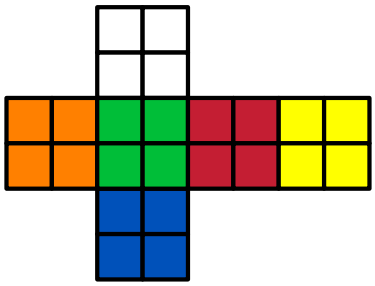
3.3 Flat and semi-flat formats

We can view the hidden faces of a cube using the semi-flat (SF) and flat (F) versions of the `\DrawTwoCube..` command as follows.

In the first figure we have also used the F version of the `\ShowCube{}{}{}` command to reveal the size of the minipage which contains the image. Notice that since the 2D width of the side face (red) is approximately 1 unit, we can estimate that the total width of the image will be $6 + 1 = 7$ units, and hence as we have specified a TikZ scale factor of 0.6, then the required width of the minipage is approximately 4.2 ($= 7 \times 0.6$) —see Section 4.1 in the RUBIKCUBE package documentation for details).



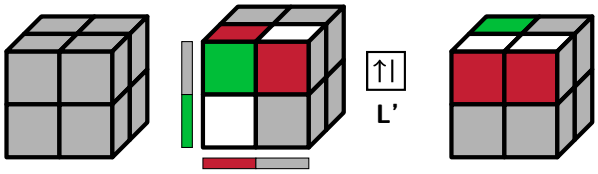
```
\TwoCubeSolvedWB%
\ShowCubeF{4.2cm}{0.6}{\DrawTwoCubeSF}%
```



```
\TwoCubeSolvedWB%
\ShowCube{4.8cm}{0.6}{\DrawTwoCubeF}%
```

3.4 Grey cube

A cube with all of the facelets coloured grey (a 'grey' cube) can be useful as a starting state for instruction. Such a cube is generated by the command `\TwoCubeGreyAll`. We can then allocate colours to those facelets demonstrating some feature of cube manipulation. Here we are showing how to rotate the GWR corner cubie in the bottom layer up into its 'solved' position in the top layer. We also use two cube sidebars to show the colours of the hidden facelets of the corner cubie.



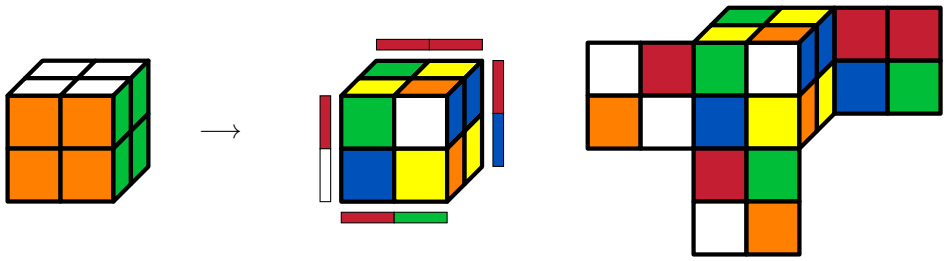
The code for the above figure is as follows:

```
\noindent\hfil%
\TwoCubeGrey
\ShowCube{2cm}{0.6}{\DrawTwoCube}%
%
\TwoFaceUp{X}{X}
          {R}{W}%
\TwoFaceFront{G}{R}
             {W}{X}%
\TwoFaceLeft XX
             XG%
\TwoFaceDown{R}{X}
            {X}{X}%
\ShowCube{2cm}{0.6}{
        \DrawTwoCube
        \DrawTwoCubeSidebarFD{RU}
        \DrawTwoCubeSidebarFL{RU}
        }%
\quad\Two{Lp}\quad
\TwoRotation{Lp}
\ShowCube{2cm}{0.6}{\DrawTwoCube}%
\hfil%
```

Notice that the curly brackets associated with the `\TwoFace..` commands can be omitted providing at least one space is left before the first colour code (see the `\TwoFaceLeft..` command above).

3.5 Scrambled cube

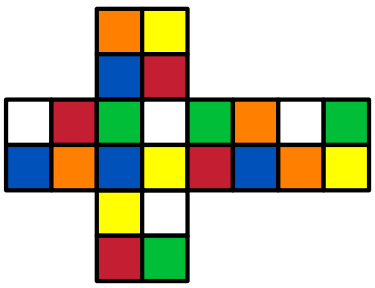
In the following example we scramble the cube using 20 random rotations using ⟨`random,20`⟩ as the argument for the `\TwoRotation{}` command. We have also added a set of cube sidebars to the second cube. The final semi-flat (SF) cube has the same colour configuration and shows the hidden faces, allowing you to check the colours of the sidebars.



```
\noindent\hfil%
\TwoCubeSolvedWY%
\ShowCube{1.8cm}{0.7}{\DrawTwoCubeRU}%
\qquad$\longrightarrow$\qquad%
%---------------
\TwoRotation{random,20}
%----------------
\ShowCube{2.8cm}{0.7}{
   \DrawTwoCubeRU
   \DrawTwoCubeSidebarFL{RU} % Front-Left edge sidebar
   \DrawTwoCubeSidebarFD{RU} % Front-Down edge
   \DrawTwoCubeSidebarBR{RU} % Back-Right edge
   \DrawTwoCubeSidebarBU{RU} % Back-Up edge
   }%
\qquad\ShowCube{5cm}{0.7}{\DrawTwoCubeSF}%
\hfil%
```

A simple scramble

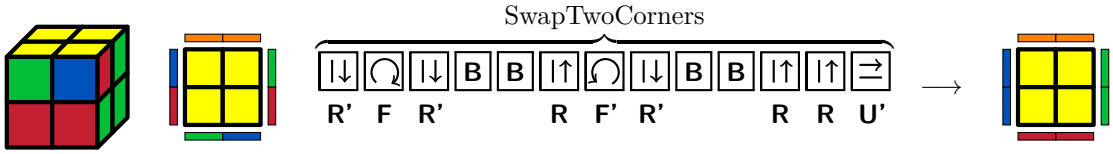
A simple scramble (from a solved position) which makes each face have four different facelets is {**R**, **y**}5 as follows (note that the optional `\TwoRotation` square bracket holds the repeat number 5):



```
\TwoCubeSolvedWB%
\TwoRotation[5]{R,y}%
\ShowCube{5cm}{0.6}{\DrawTwoCubeF}%
```

3.6 Swap two corners

The following example is the header figure for the RUBIKTWOCUBE package documentation, and shows the 'SwapTwoCorners' sequence often necessary in the solution of a 2x2x2 cube.

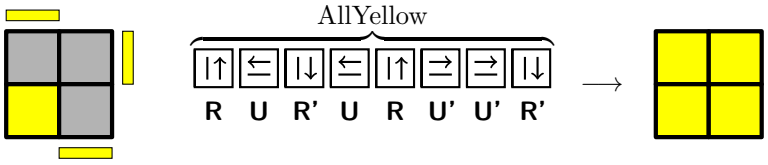


The code for the above figure is as follows:

```
\usepackage{tikz}
\usepackage{rubikcube,rubikrotation,rubikpatterns,rubiktwocube}
...
\newcommand{\SwapTwoCorners}{[swaptwocorners],Rp,F,Rp,B2,R,Fp,Rp,B2,R2,Up}
\newcommand{\swaptwocorners}{\SwapTwoCorners}
...
\noindent
\hfil%
%% set up the cube with corners swapped
\TwoCubeSolvedWY%
\TwoRotation{Rc2}%  turn cube upside down about the R face
\TwoRotation{\swaptwocorners}%
%% now show the cube, before and after swapping the corners
\ShowCube{1.6cm}{0.6}{\DrawTwoCubeRU}%
\quad\ShowCube{1.7cm}{0.5}{\DrawTwoFaceUS}%
\TwoRotation{\swaptwocorners}%
\quad\SequenceBraceA{SwapTwoCorners}{\ShowSequence{}{\Two}{\SequenceLong}}%
\quad$\longrightarrow$%
\quad\ShowCube{1.7cm}{0.5}{\DrawTwoFaceUS}%
\hfil%
```

3.7 OLL example

Here we illustrate an OLL algorithm which orientates the last layer starting from a single yellow facelet in the UFL position.



The code is shown below. Note that we have used the `\NoSidebar{X}` command to disable the drawing of grey sidebars by the `\DrawTwoFaceUS` command[8] and hence only yellow sidebars are drawn. Note that we have placed the `\NoSidebar{X}` command inside the `\ShowCube` environment in order to limit its action locally.

```
\noindent\hfil%
\TwoCubeGreyAll%
\TwoFaceUp{X}{X}%
          {Y}{X}%
\TwoFaceBack  XY XX
\TwoFaceRight XY XX%
\TwoFaceFront XY XX%
\ShowCube{2cm}{0.7}{\NoSidebar{X}\DrawTwoFaceUS}%
\TwoRotation{\allyellow}%
\quad\SequenceBraceA{AllYellow}{\ShowSequence{}{\Two}{\SequenceLong}}%
\quad$\longrightarrow$\quad%
\ShowCube{1.5cm}{0.7}{\NoSidebar{X}\DrawTwoFaceUS}%
\hfil%
```

—— END ——

[8] Note that the `\DrawTwoFaceUS` command is a short-hand equivalent to the `\DrawTwoFaceUpSide` command.