# A pattern from the Alhambra

Laurence Finston

## Abstract

This article demonstrates the use of GNU 3DLDF for creating graphics based on a pattern from the Alhambra, a complex of fortifications and palaces outside Granada, Spain. It explains and demonstrates how to create a perspective drawing using 3DLDF and the technique of *color replacement*.

## Introduction

The Alhambra is a complex of fortifications and palaces outside Granada, Spain, built from 1238 into the early 17th century, whereby the era of Islamic architecture there, as in the rest of Spain, ended in 1492 with the completion of the *Reconquista*. It is one of the world's best known architectural monuments and famous for its elaborate ornamentation in the Islamic style, often based on *plane tessellations* and *periodic tilings*, which have inspired many architects, designers and artists, notably M.C. Escher, the incomparable modern master of this métier.

Beginning in 1842, Owen Jones began publishing the monumental work *Plans, Elevations, Sections and Details of the Alhambra*, based on work he had done together with Jules Goury, who died of cholera during their stay at the Alhambra. Published in installments, this work has the distinction of being the first "significant" one to use the process of chromolithography for color reproduction [16].

Figure 1 shows Plate XLIX, No. 85, Mosaic in the Divan, Court of the Fishpond, from Goury and Jones, Vol. 2 [10]. It shows a single rapport of the pattern, so the latter is clearly unusually complex.

Figure 2 shows a version of this pattern which I created using GNU 3DLDF and GIMP with *color replacement* (explained below).

## Plane tessellations

"A tessellation or tiling is the covering of a surface, often a plane, using one or more geometric shapes, called tiles, with no overlaps and no gaps." [17]

Of the regular polygons, only three can form a plane tessellation, the equilateral triangle, the square and the regular hexagon (figures 3, 4, and 5 on the following page).

The pattern in question, referred to in the following as pattern 207, is based on the familiar "honeycomb" pattern consisting of regular hexagons.

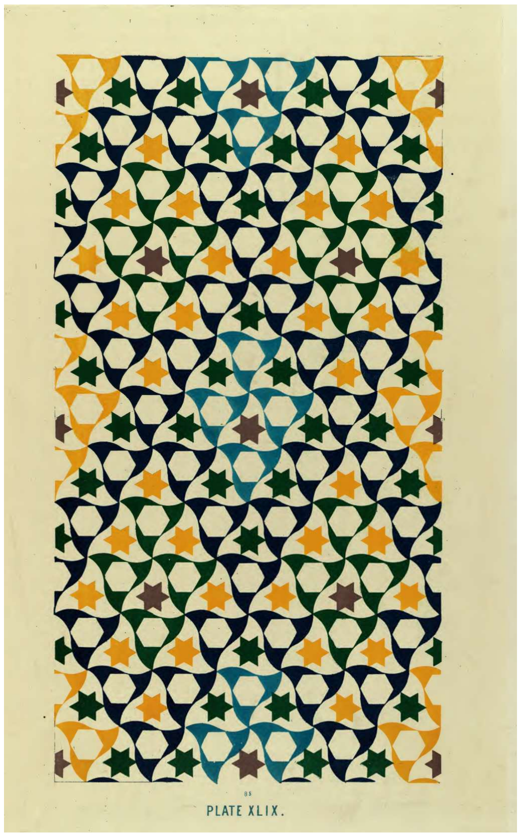

**Fig. 1**: Plate XLIX from [10].

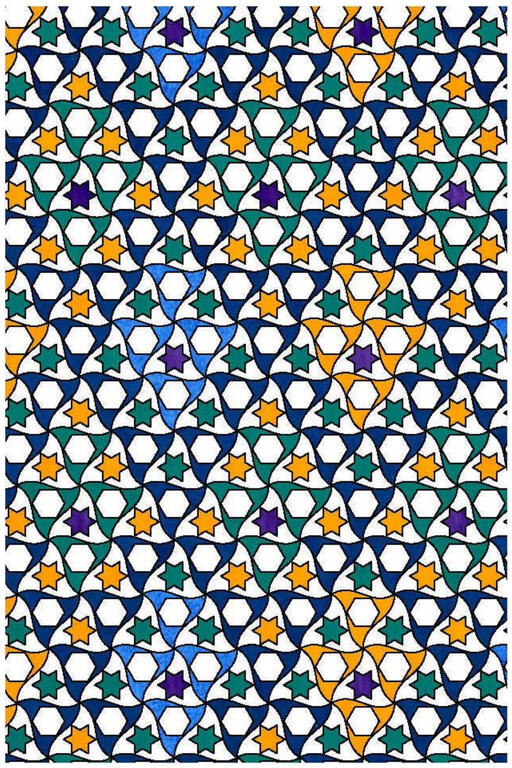

**Fig. 2**: 3DLDF version, color replaced.

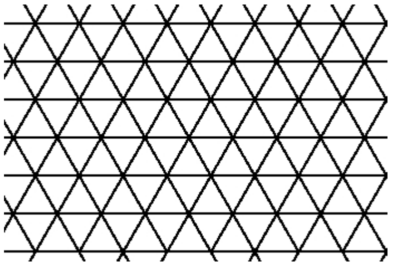
**Fig. 3**: Plane tessellation, equilateral triangles.

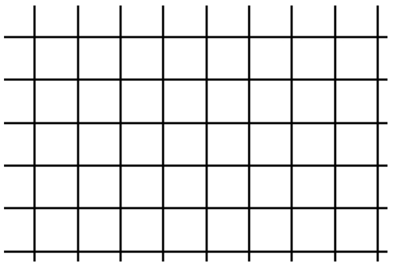
**Fig. 4**: Plane tessellation, squares.

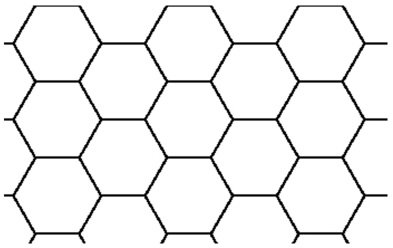
**Fig. 5**: Plane tessellation, regular hexagons.

Since each hexagon is divided into six equilateral triangles, one may say with as much justification that it is based on the plane tessellation using triangles. In pattern 207, the triangles in each hexagon alternately contain a smaller hexagon or a six-pointed star:

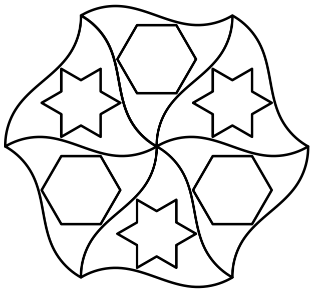
**Fig. 6**: Hexagon with inner figures.

One may thus consider such a hexagon the "basic unit" of the pattern. One way to fill the plane would be to copy the basic unit and shift it to the right and upward repeatedly, then to copy the original unit downward and repeat the procedure. If, however, we consider the "basic unit" to be two hexagons as shown in figure 7, then the plane may be filled using only translations (*shifts*) in the horizontal and vertical directions, which is more convenient from the point of view of programming.

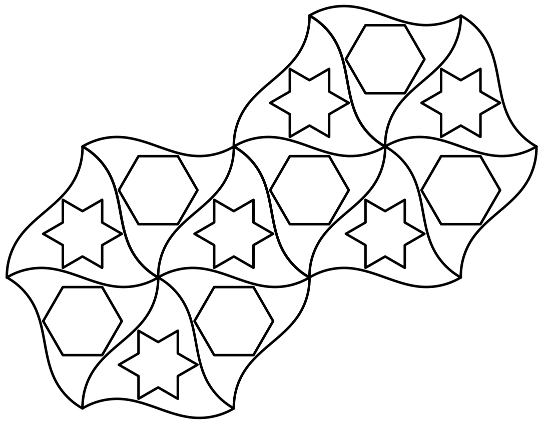
**Fig. 7**: Basic unit of the pattern.

In the hexagons in pattern 207, the straight edges of figure 6 have been replaced by undulating curves. However, since they are symmetrical, they don't affect the plane-filling property of the tessellation.

In the 3DLDF program, the curve is specified in this way:

```
point Z[]; path hex[];
for i = 0 upto 5:
  Z[i] := (2cm, 0) rotated (0, 0, i*60);
endfor;

path p[]; [...]
for i := .25 step .25 until .75:
  Z[6+j] := mediate(Z2, Z1, i) shifted (0, k*m);
  j += 1;
  m -= 1;
endfor;
p0 := Z2 .. {dir 0}Z6{dir 0} .. Z7
   .. {dir 0}Z8{dir 0} .. Z1;
```

This code would be similar in METAPOST except that the mediation operator [] would be used

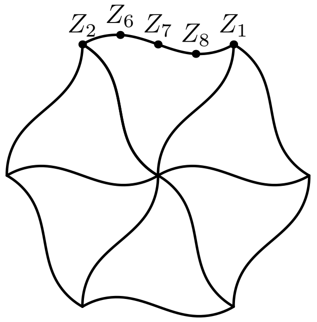

**Fig. 8**: Hexagon with wavy edges.

instead of 3DLDF's **mediate** command. `[]` doesn't exist in 3DLDF because it would cause conflicts in the Bison parser [8].

The direction specifiers used to define **p0** work fine when the drawing is projected onto the x-y plane, which is, in fact, equivalent to using METAPOST in the first place. However, they will produce erroneous results when it is projected using the perspective projection, for reasons explained at length in [8]. Therefore, in order to make **p0** projectable, it must be "resolved", i.e., enough points must be added to it along its length so that it won't "go out of shape" when projected using the perspective projection:

```
path_vector pv;
pv := resolve p0 to 80;
p0 := pv0;
```

Generally speaking, the more extreme the foreshortening, the more points are needed. So far, 80 has proven to be a sufficient number of points for this drawing.

After the first two hexagons with their inscribed figures have been drawn, they must be copied and translated as many times as necessary to create a single rapport, which consists of 6 rows of 4 hexagons, offset as described above. This is fairly simple. Following this, they must be colored, which is not. This task is performed by the macro **tessellate**, which is defined in the file `sub_alhambra_207_1.ldf`, which like all of the source files referred to in this article, and all of the images contained in it, is included in the distribution of 3DLDF.

For maximum flexibility, each of the paths representing the outlines of the triangles and the inscribed figures is assigned to a variable. This makes it possible to access each one individually. For example, the same pattern could be colored in a completely different way. (The outlines of the outer hexagons are not assigned to **path** variables.)

**tessellate** takes eight parameters, one for the pen used for drawing the outlines and seven for the colors in the pattern, including the colors for the outlines and the background. This is the call to **tessellate** for figure 9 (on the following page):

```
tessellate {medium_pen, black, white, blue,
   teal_blue, orange, cyan_cmyk, purple};
```

Using parameters for the colors makes it possible to use any combination of colors desired, and in particular, to make "masks" for *color replacement*, as described below. (In the following, most of the images have been rendered to JPEG bitmaps for ease in processing.) **tessellate** draws the outlines on the picture **v6** and fills the areas of color on picture **v5**. Labels are drawn on the picture **v105**. Again, for maximum flexibility, the paths for each individual row of triangles are also drawn and filled on separate pictures (**row_picture_draw0** … **row_picture_draw12** and **row_picture_fill0** … **row_picture_fill12**).

While it is most logical to consider the rows of the pattern as referring to the hexagons, it is more practical from the point of view of keeping track of locations within it for the purpose of coloring to consider the rows as referring to the triangles. Seen in this way, there are thus 12 rows in a rapport, offset to one another horizontally, but not vertically.

Creating a picture for each row of triangles makes it possible to use the pattern in drawings without necessarily using only complete rapports or having to clip the picture.

## Color in 3DLDF

Color in 3DLDF works similarly to color in METAPOST, but there are a few differences. In METAPOST, **color** (i.e., **(rgb)color** and **cmykcolor**) are two different types; it's not possible to assign a value of one type to a variable of the other. In addition, "greyscale" colors are specified using *numeric expressions*. There is no type **greyscalecolor**.

In 3DLDF, there is only one type, namely **color**, for all three kinds of color and variables of this type can take on RGB, CMYK and greyscale values freely. All **color** objects contain all of the following "parts":

```
red_part green_part blue_part
cyan_part magenta_part yellow_part black_part
grey_part
```

In METAPOST, the spelling "grey" is mostly used rather than "gray". I prefer the spelling "gray", so it is used in 3DLDF except where consistency with METAPOST is desirable, e.g., **grey_part**. However, both spellings may be used for all commands and keywords in 3DLDF, since synonyms are defined for them all.

```
color c[];
c0 := (.5, .5, 0); % RGB
show c0;
>>
color:
name         == c[0]
type         == 3 (RGB_COLOR)
red_part     == 0.50000000
green_part   == 0.50000000
blue_part    == 0.00000000
cyan_part    == 0.00000000
magenta_part == 0.00000000
yellow_part  == 0.00000000
black_part   == 0.00000000
grey_part    == 0.00000000
```

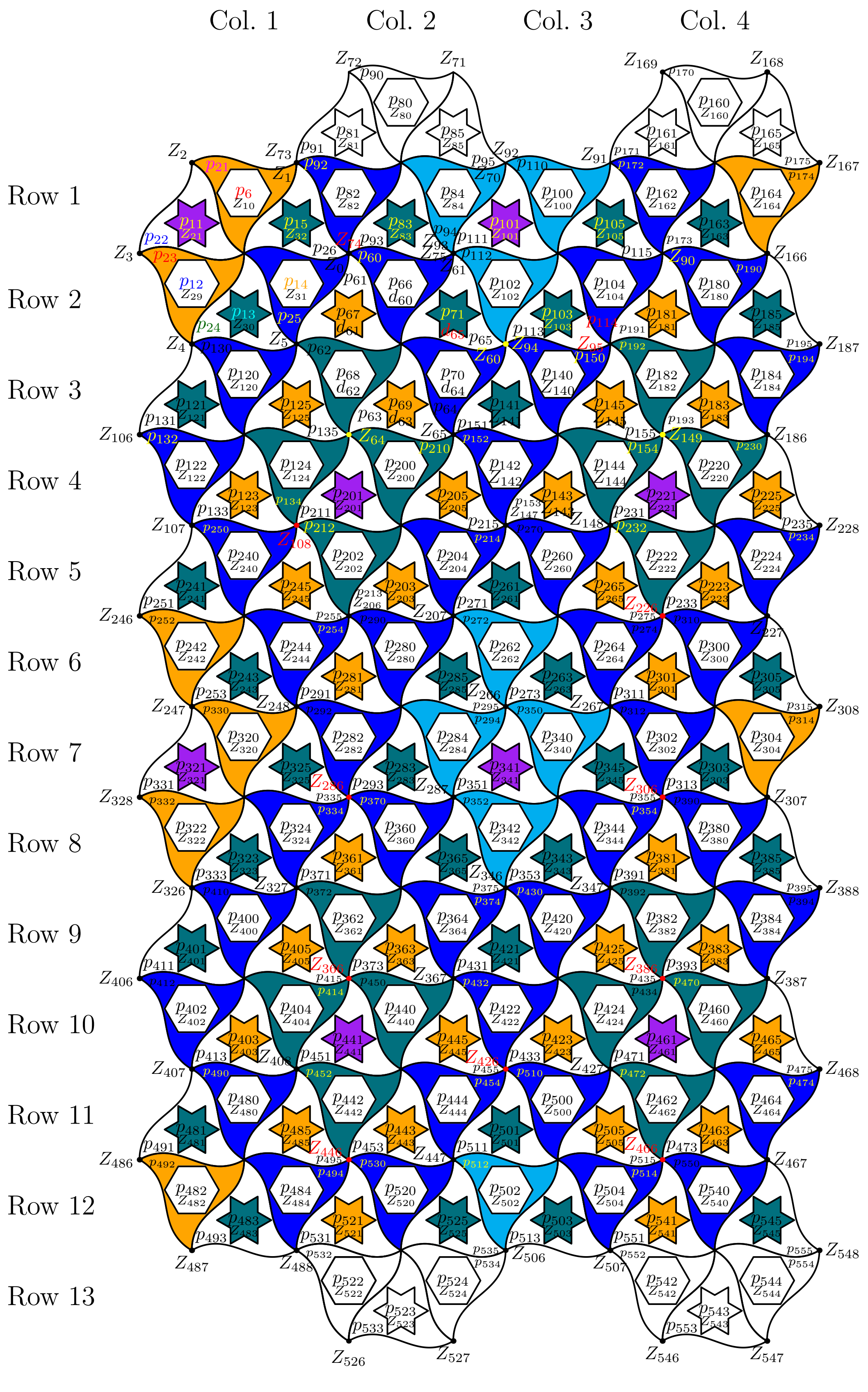

**Fig. 9**: Colored tessellation example with 3DLDF.

```
c1 := (.2, .3, .4, .5); % CMYK
show c1;
>>
color:
name         == c[1]
type         == 2 (CMYK_COLOR)
red_part     == 0.00000000
green_part   == 0.00000000
blue_part    == 0.00000000
cyan_part    == 0.20000000
magenta_part == 0.30000001
yellow_part  == 0.40000001
black_part   == 0.50000000
grey_part    == 0.00000000

c2 := .3; % Greyscale
show c2;
>>
color:
name         == c[2]
type         == 4 (GREYSCALE_COLOR)
red_part     == 0.00000000
green_part   == 0.00000000
blue_part    == 0.00000000
cyan_part    == 0.00000000
magenta_part == 0.00000000
yellow_part  == 0.00000000
black_part   == 0.00000000
grey_part    == 0.30000001
```

When a value is assigned to a **color** variable, 3DLDF "automatically" recognizes what kind of color it's supposed to be, RGB, CMYK or greyscale and the "parts" are set accordingly. If subsequently a value of a different type is assigned to it, it will take on the new type.

A single numeric value is used to specify a grayscale color, a ⟨*numeric list*⟩ with three elements specifies an RGB color and one with four elements, a CMYK color.

While 3DLDF and METAPOST support RGB and CMYK colors and there are other color models supported by other software, ultimately colors will be converted to the model required by the device that displays or prints them. Generally speaking, computer displays require RGB and printers CMYK. Please note that conversion is *not perfect*, nor can it be. Often the results are good, but sometimes the colors don't match well at all. It seems this occurs most often when CMYK containing a non-zero **black_part** is converted to RGB.

In addition, I have found that it doesn't work at all to import images from 3DLDF or METAPOST containing CMYK colors into the video editing software Flowblade for animations: they are simply not displayed. For this purpose, RGB colors must be used instead.

**Color reproduction.**

**RGB colors.** The topics of color in general and color reproduction in particular are complex and it is beyond the scope of this article to discuss them in detail. In short, red, green and blue are the *primary colors* for *additive mixing* and cyan, magenta and yellow are the primary colors for *subtractive mixing.* Additive mixing applies to light and subtractive applies to pigments that absorb light selectively.

Computer displays that could display red, green and blue and mix these colors were finally made possible by the invention of the blue LED. Before this time, the only form of additive mixing in general use was with theatrical gels (i.e., filters) and spotlights. White light, or more accurately, light which the human eye perceives as white, results from mixing red, green and blue light.

The pixels on a computer display, each of which consists of a triplet of tiny LEDs, one red, one green and one blue, do the same thing, but on a much smaller scale and at a much lower intensity.

Mixing red and green light produces yellow, red and blue magenta and green and blue cyan, so yellow, magenta and cyan are the *RGB secondaries* in addition to being the CMYK primaries.

**CMYK colors.** Schoolchildren are taught that the primary colors are red, blue and yellow and that mixing red and blue produces purple (violet), red and yellow produces orange and blue and yellow produces green. This isn't wrong, but the colors used (at least when I was in elementary school) are somewhat "off" compared to what is now considered correct.

Mixing any red and blue paints will produce a purplish color and the same applies to the other combinations. All of the possible combinations of a given set of a single red, blue and yellow paint, respectively, will produce a "color space". It has been determined, theoretically and/or by experiment, that the largest possible color space under real-world constraints can be achieved by using the colors cyan, magenta and yellow in the shades (not coincidentally) used as ink in offset printing.

What happens in practice when mixing two pigments is in effect equivalent to mixing two pure colors and adding gray, i.e., a mixture of black and white pigments. Since the "colorful" pigments, i.e., not black, white or gray, are more expensive than black, white or gray pigments, it is wasteful to use the colorful pigments for this purpose. It is far better, where possible, to use a pigment of the desired shade and to add gray to it.

In addition, while it is theoretically possible to produce black by mixing cyan, magenta and yellow, the result is unlikely to be satisfactory and these pigments are much more expensive than black, which is often simply soot produced by burning wood (charcoal), acetylene gas or animal bones (in the past, scraps of ivory were also used).

For these reasons, the CMY model (which also exists) is generally expanded to CMYK (where "K" stands for "black").

Since working with pigments is subject to real-world constraints, a color space with more possible colors and gradations of color can be achieved by using more pigments, including for the CMYK secondaries and other combinations. There are pigments that have shades that are not reproducible by mixing other pigments.

The use of CMYK in offset printing and in laser and inkjet printers is called a "four-color" process. There are also seven-color and eight-color processes which use additional pigments for the sake of better color reproduction.

**Built-in RGB colors.** These RGB colors are built-in, i.e., they are defined within the C++ code for 3DLDF (in the file `sctpcrt.web`):

```
black white
red green blue
pink
yellow cyan magenta
orange violet purple
yellow_green green_yellow dark_green
blue_violet violet_red
brown
gray light_gray dark_gray
```

Additionally, `grey`, `dark_grey` and `light_grey` are defined as synonyms for `gray`, etc.

Note that cyan, yellow and magenta are defined as secondary RGB colors rather than primary CMYK colors; also, violet and purple are two different colors:

```
show cyan;
>>
color:
name         ==
type         == 3 (RGB_COLOR)
red_part     == 0.00000000
green_part   == 1.00000000
blue_part    == 1.00000000
```
(remaining elements zero)

```
show magenta;
>>
color:
name         ==
type         == 3 (RGB_COLOR)
red_part     == 1.00000000
green_part   == 0.00000000
blue_part    == 1.00000000
```
(remaining elements zero)

```
show yellow;
>>
color:
name         ==
type         == 3 (RGB_COLOR)
red_part     == 1.00000000
green_part   == 1.00000000
```
(remaining elements zero)

```
show violet;
>>
color:
name         ==
type         == 3 (RGB_COLOR)
red_part     == 0.93333334
green_part   == 0.50980395
blue_part    == 0.93333334
```
(remaining elements zero)

```
show purple;
>>
color:
name         ==
type         == 3 (RGB_COLOR)
red_part     == 0.62745100
green_part   == 0.12549020
blue_part    == 0.94117647
```
(remaining elements zero)

**CMYK colors defined in plainldf.lmc.** Numerous CMYK colors are defined in the `plainldf.lmc` source file, which is included in the distribution of 3DLDF. Versions of the CMYK primaries (including black) and secondaries (which, black aside, correspond exactly to the RGB secondaries and primaries):

```
cerulean_blue teal_blue dark_blue
dark_olive_green mauve turquoise
rose_madder lime_green
```

`plainldf.lmc` also defines RGB versions of these colors:

```
cerulean_blue_rgb teal_blue_rgb dark_blue_rgb
dark_olive_green_rgb mauve_rgb turquoise_rgb
rose_madder_rgb lime_green_rgb
```

In addition, `plainldf.lmc` defines CMYK and RGB versions of the colors defined in the `color.pro` and `colordvi.tex` files from the Dvips distribution (located under `/usr/share/texlive` in
`texmf-dist/dvips/base/color.pro`
`texmf-dist/tex/generic/dvips/colordvi.tex`
on my computer). These colors and their names are based on the box of 64 Crayola crayons (as of some past date). The versions from Dvips are CMYK colors. The RGB versions defined in `plainldf.lmc` have the suffix `_rgb`:

```
color GreenYellow; color GreenYellow_rgb;
color Yellow; color Yellow_rgb;
color Goldenrod; color Goldenrod_rgb;
color Dandelion; color Dandelion_rgb;
```
etc.

**Computer-generated vs. painted colors.** One problem with filling paths with computer-generated colors is that, leaving *antialiasing* (explained below) aside for the moment, they are entirely uniform. Neither the colors displayed on a monitor nor those produced by an inkjet or laser printer can compare with the appearance of fine artists' colors on high-quality paper. On the other hand, while they can't compare with the appearance of the originals either, the results of scanning drawings and paintings and displaying and printing them using standard office equipment are often surprisingly good.

Since these results are also just representations of pixels, it would, of course, be theoretically possible to simulate scanning a painted background, for example, using the computer only. In practice, however, it is most likely easier to just scan real painted backgrounds, since it would be difficult to program all of the many variations in appearance that in combination result in their characteristic appearance.

Figures 10 and 11 show scans of watercolor and gouache backgrounds, respectively, which I painted. With a few exceptions, they are in DIN A4 landscape format (297mm × 210mm) with a 15mm margin on each side. They have all been painted on watercolor paper, but not all of the same kind. For most I have used paints made by the company Schmincke from their Horadam lines of watercolor and gouache, respectively. Some of the gouache backgrounds were made using paints from Schmincke's HKS Designers' Gouache line and a couple of the watercolors were made using paints from the company Old Holland. Schmincke and Old Holland are both top-of-line manufacturers of artists' colors.

In figure 2, the computer-generated colors have been replaced by portions of a selection of these backgrounds using GIMP.

**Paint characteristics.** Gouache and watercolor are both made by combining one or more pigments with water and a small amount of gum arabic. However, unlike watercolor, gouache may contain additives to make it opaque, including chalk or white pigment. In traditional watercolor painting, translucent colors are preferred and white paint is never used; white areas in a watercolor painting are achieved by leaving the paper blank. Due to the translucency of the paint and the fact that watercolor doesn't lie on the surface of the paper but rather soaks into it and stains the fibers, the structure of the paper will normally remain visible and contribute significantly to the total impression of the work. Watercolor paints consist almost entirely of pigment so that the colors are very pure. All of these factors contribute to the luminosity that is typical of watercolor paintings. In addition, they often reproduce extremely well.

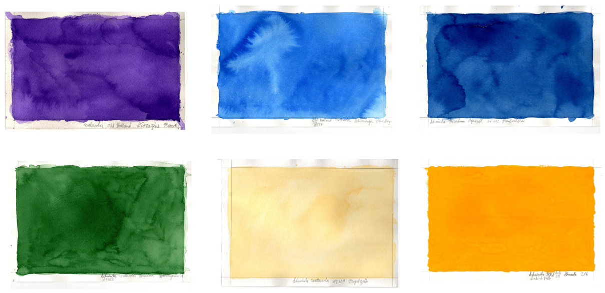
**Fig. 10**: Watercolor backgrounds.

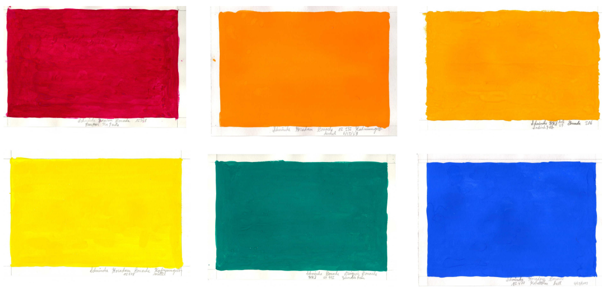
**Fig. 11**: Gouache backgrounds.

Many techniques are possible with watercolors. One is to use multiple layers of diluted washes to create a glazed effect. Light colors are typically achieved not by mixing with white, but by diluting the paint. Accidental effects, where the pigment concentrates in particular areas, are often desirable in watercolors.

Gouache, in contrast, is always opaque and has a matte surface. It doesn't soak into the paper to any appreciable extent but rather forms a layer on top of it. Normally, each area of color, or each individual brushstroke in areas containing more than one color, is meant to be completely uniform with respect to color and opacity. In other words, a poster-like effect, but with a matte surface, is usually desired when using gouache. Light colors are achieved by mixing with white; dark colors may have to be lightened with white or they may be virtually indistinguishable from black.

While it is possible to dilute gouache and the results don't exactly look bad, there's normally not much point in doing this, because one might as well just use watercolor, which, unlike gouache, is intended to be used in this way.

The variations in the watercolor backgrounds I've been making for this and other projects cannot practically be achieved by using the computer. With gouache, the situation is different. Since areas of a single color in gouache paintings are usually intended to be entirely uniform, theoretically, the only advantage of using a gouache background is the difficulty of discovering, inventing or stumbling upon attractive colors just by using the computer. In practice, however, there are variations in the gouache backgrounds and the structure of the paper sometimes does show through, adding variability and interest to what appears at first glance to be a solid block of color.

## Color replacement

Performing color replacement requires a number of steps. First, the image needs to be output multiple times so that the individual versions may be used as *masks*. Figures 12 through 17 show the parts of the pattern for each color individually with all of the other colors replaced by gray.

These images are generated with METAPOST, and must be imported into GIMP. METAPOST can produce output in the form of EPS, SVG or PNG files. It doesn't matter which format is used, but it is essential that the image be imported *without* antialiasing. Antialiasing is a procedure whereby the color of pixels near an edge where areas of two different colors meet may be altered slightly in order to improve the appearance of the edge.

However, color replacement depends on all areas of a given color being entirely uniform, so that they may be accessed by using GIMP's "Select by color" tool. When importing EPS files, at least, into GIMP a menu appears where antialiasing can be enabled or disabled. I usually have METAPOST output EPS files. When generating PNG output, antialiasing can be suppressed by using the corresponding option to **outputformatoptions**. In METAPOST:

```
outputtemplate := "%j%3c.png";
outputformat:="png";"
outputformatoptions :=
   "format=rgba antialias=none";
```

Or in 3DLDF:

```
verbatim_metapost
  "outputtemplate := \"%j%3c.png\";"
  & outputformat:=\"png\";"
  & "outputformatoptions := "
  & \"format=rgba antialias=none\";";
```

When the image is loaded into GIMP, it has a single layer. First, an *alpha channel* (for transparency)

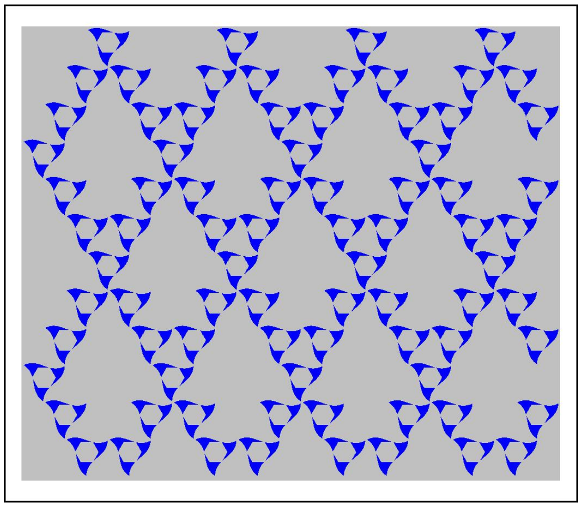
**Fig. 12**: Blue mask

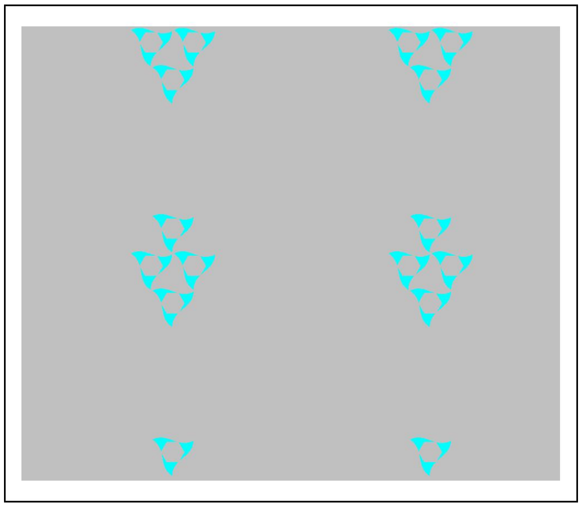
**Fig. 13**: Cyan mask.

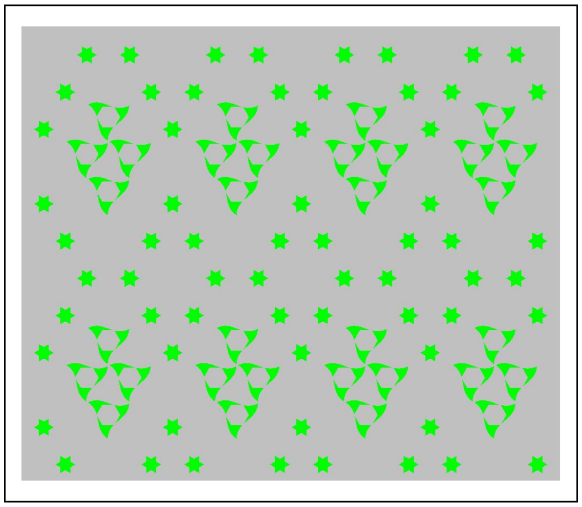
**Fig. 14**: Green mask.

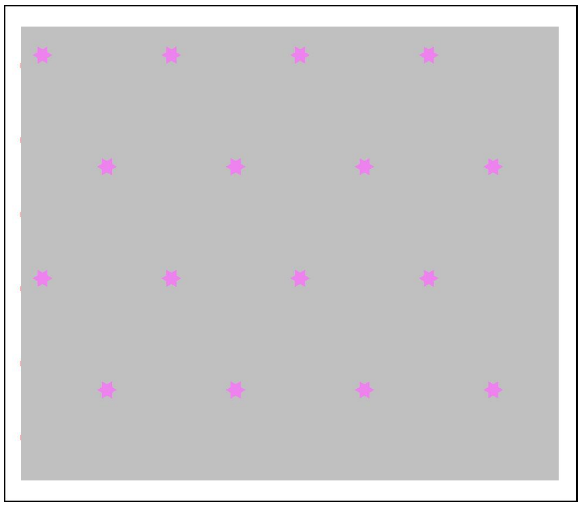
**Fig. 15**: Purple mask.

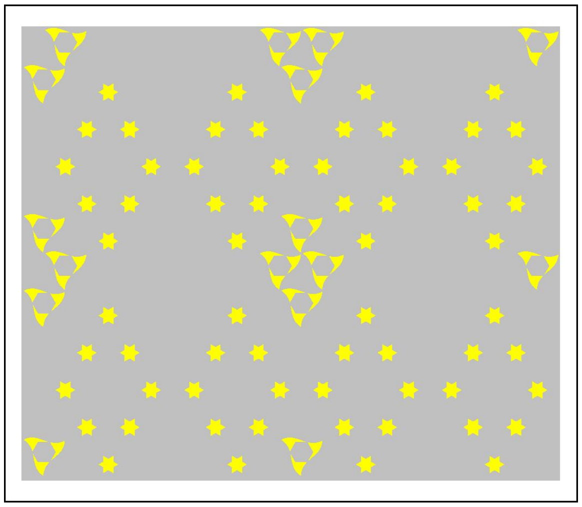
**Fig. 16**: Yellow mask.

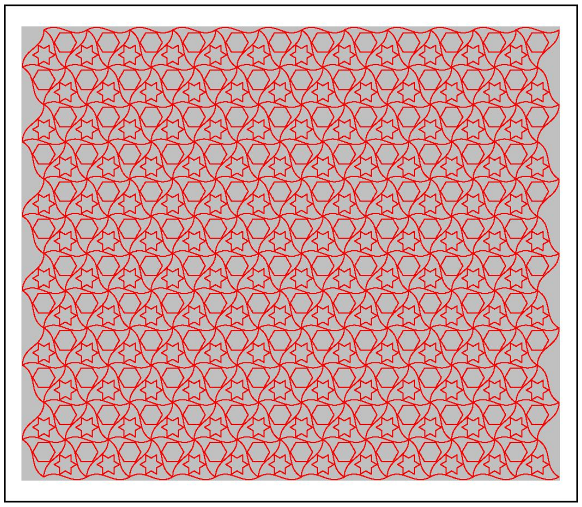
**Fig. 17**: Outlines mask.

must be added by calling the “Add alpha channel” command in the “Layer” menu. Then the layer is duplicated so that both a positive and negative mask may be created. Although a single mask would suffice, it is convenient to have two separate ones.

To make the positive mask, the blue areas are selected by using the “Select by color” tool and clicking on a spot where there are only blue pixels. This selects all of the blue pixels in the image. Then, the selection is inverted so that all of the other portions of the image are selected instead. These are then made transparent by pressing the “Delete” key.

To make the negative mask, the same procedure is followed, except that the selection isn’t inverted, so that the blue areas are made transparent.

In order to fill the blue areas with colors from one of the painted backgrounds, the positive mask is made invisible by clicking on the “eye” symbol next to the name of the layer in the listing of layers in the GIMP window, the negative mask is duplicated and the original negative mask is also made invisible.

Then, the desired file is opened by using the “Open as Layers” command in the “File” drop-down menu. It is placed behind the layer with the copy of the negative mask and the “Merge visible layers” command is executed. The resulting layer contains the portions of the background image corresponding to the blue areas of the pattern, surrounded by gray. (Figure 18.)

Now the gray areas of this layer must be made transparent. However, this cannot be accomplished by simply selecting them by color: This only works when the layer contains areas of color that are completely uniform and distinct. The painted background image is likely to contain pixels of many different colors, including gray. If the “Select by color” tool is used to select gray pixels, pixels within the “blue” areas will also be selected, producing ugly and unusable results. In addition, there is no longer any way to select the areas where the color replacement has taken place directly, again, because these areas now are likely to contain pixels of many different colors and also because the areas aren’t contiguous. Therefore neither the “Select by color” nor the “Fuzzy select” tool will select these areas correctly.

To select only the gray areas in the combined layer, first the layer with the original negative mask must be made visible and clicked on to make it the active layer. Then, the gray areas are selected and the combined layer is chosen as the active layer. When the “Delete” key is now pressed, the gray areas on the combined layer are made transparent.

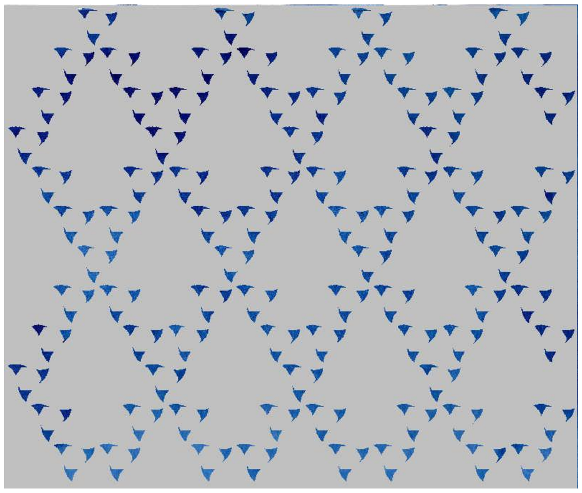

**Fig. 18**: Blue mask, color replaced.

When an area is selected using one layer and the selection is used to make changes to another, the first layer is called a "mask".

The procedure may be varied and the steps don't necessarily have to be performed in exactly the order given, but the principle remains the same and it's six-of-one, half-a-dozen of the other how it's done exactly, as long as the desired result is achieved.

Of course, it's possible to open multiple files containing background images as layers, combine them with copies of the negative mask, make them visible or invisible, use a single file to contain all of the masks, save them in separate files, etc.

When layers have been created for all of the masks, and with painted backgrounds replacing the computer-generated colors in the original image, they may be *composited* to form a "color replaced" image. (Figure 19.)

**The perspective projection.** For the parallel projection of pattern 207 onto the x-y plane alone, there would be no need to use 3DLDF: METAPOST would have sufficed. For making versions of it using the perspective projection, 3DLDF is required. Figures 20 and 21 show pattern 207 lying in the x-z plane and projected using the perspective projection, with the focus set as follows:

```
focus f;
set f with_position (5, 10, -40)
   with_direction (5, 10, 10)
    with_distance 45;
```

(The units are cm.)

Masks are created and color replacement is performed in GIMP in exactly the same way as with the version using the parallel projection, except that clipping is performed in GIMP rather than in 3DLDF. If desired, the image may be cropped using the "Crop to content" command in the "Image" drop-down menu.

3DLDF implements various commands for clipping, which all work by writing a call to METAPOST's **clip** ⟨*picture*⟩ to ⟨*path*⟩ operation. This works fine for parallel projections, but for reasons (as yet) unbeknownst to me, it does not for the perspective projection. I plan on debugging the relevant functions, although using a simple example rather than one containing over 300 paths.

There is no particular advantage to performing the clipping in 3DLDF (or METAPOST), especially since clipping in METAPOST is not completely bulletproof: Clipping doesn't actually remove any objects, it just hides them. They are still present and still affect the size of the bounding box of the image, which is important when including the image in a TeX file, for example. In addition, if a clipped image is included in a PDF file (via TeX or some other way), the areas that are supposed to have been hidden may be displayed anyway. I've had this problem with the Firefox web browser.

## Bibliography

[1] Bongartz, Klaus et al. *Farbige Parkette.* Basel: Birkhäuser Verlag, 1988.

[2] Bonner, Jay. *Islamic Geometric Patterns.* New York: Springer, 2017.

[3] Brend, Barbara. *Islamic Art.* Cambridge, Massachusetts: Harvard University Press, 1991.

[4] Brentjes, Burchard. *Die Kunst der Mauren.* Köln: DuMont Buchverlag, 1992.

[5] Cundy, H. Martyn and Rollet, A. P. *Mathematical Models.* Oxford: Oxford University Press, 1961.

[6] Ernst, Bruno. *Der Zauberspiegel des M.C. Escher.* Berlin: TACO Verlagsgesellschaft und Agentur mbH, 1986.

[7] Escher, M. C. *Graphik und Zeichnungen.* 13. unveränderte deutsche Auflage. München: Heinz Moos Verlag GmbH & Co. KG, 1979.

[8] Finston, Laurence. *An introduction to GNU 3DLDF. TUGboat* 43(3):319–332, 2022. `tug.org/TUGboat/tb43-3/tb135finston-3dldf.pdf`

[9] Gimpl, Karoline. *Andalusien.* DuMont Kunst-Reiseführer. 2. Auflage. Ostfildern: DuMont Reiseverlag, 2012.

[10] Goury, Jules and Jones, Owen. *Plans, Elevations, Sections and Details of the Alhambra.* London: Owen Jones, 1842–1845.

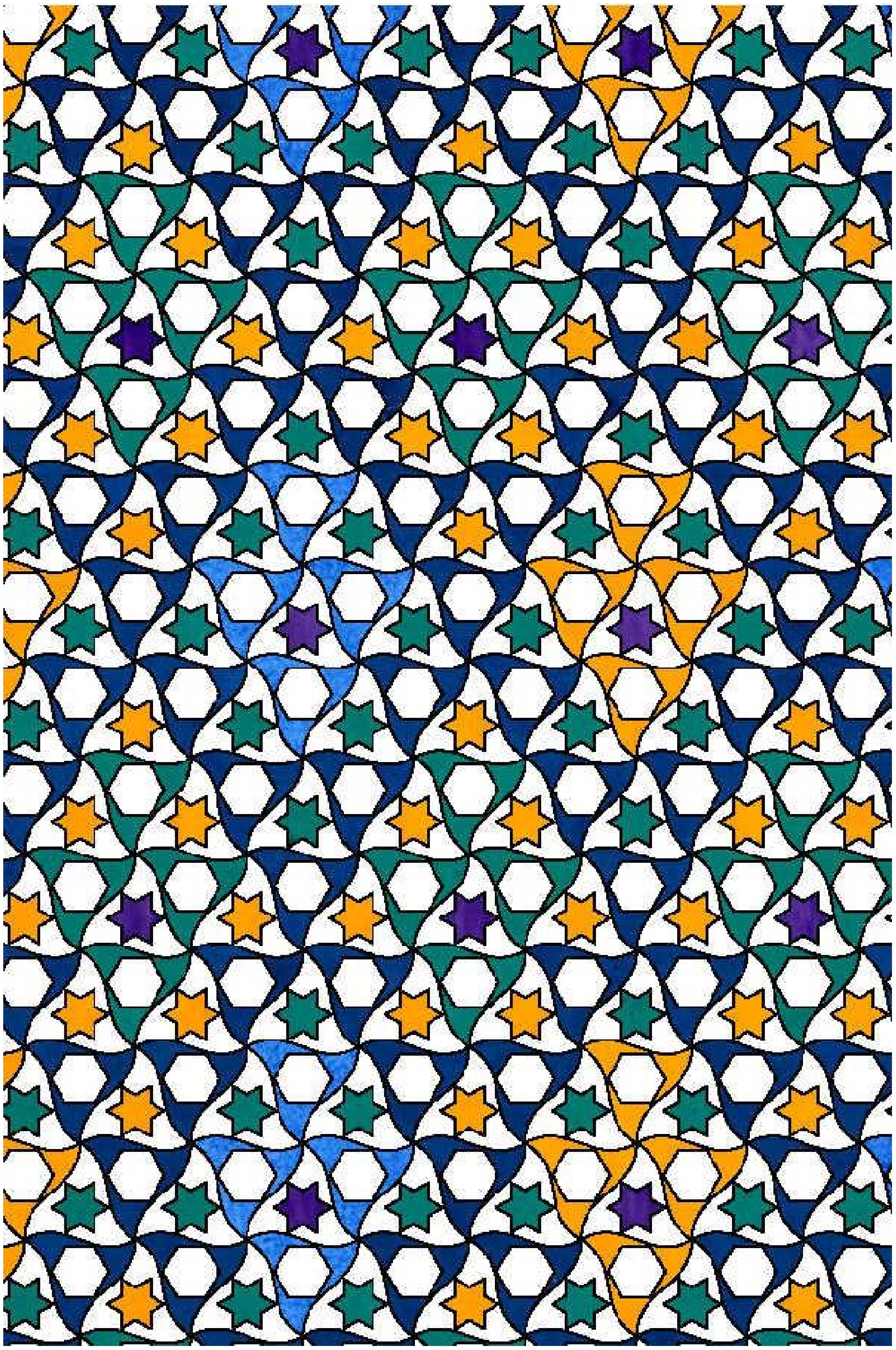

**Fig. 19**: Color replaced parallel projection

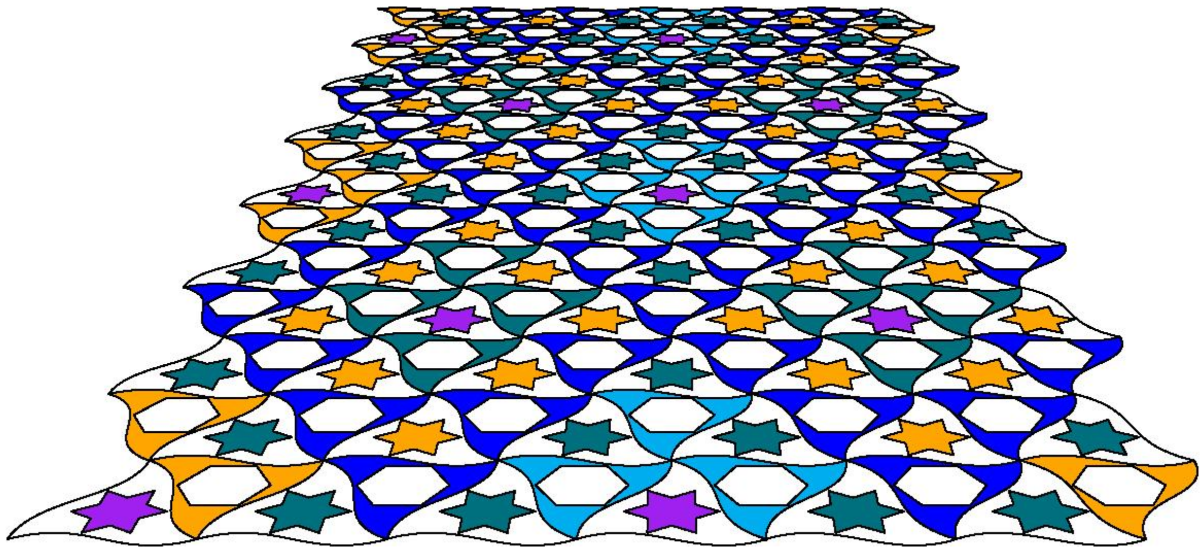
**Fig. 20**: Perspective projection.

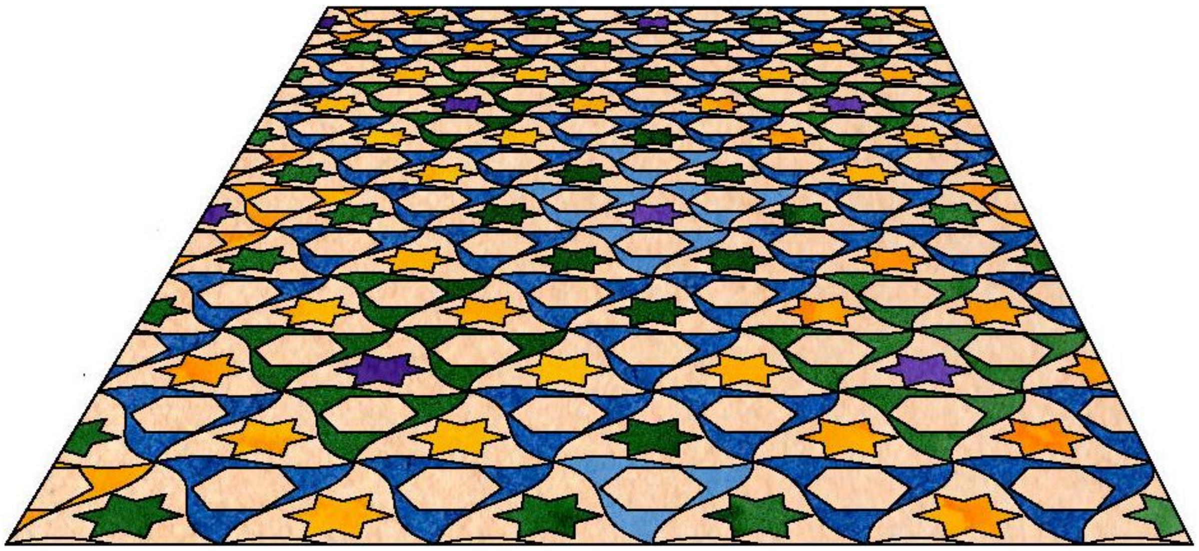
**Fig. 21**: Perspective projection, colors replaced.

[11] Hobby, John D. and the MetaPost development team. *METAPOST, A User's Manual.* 2020. `tug.org/metapost`

[12] Kühnel, Ernst. *Die Kunst des Islam.* Springers Handbuch der Kunstgeschichte. Stuttgart: Alfred Kröner Verlag, 1962.

[13] Küppers, Harald. *Schule der Farben.* Köln: DuMont Buchverlag, 1992.

[14] Lata, Sabine. *Die Alhambra. Geschichte—Architektur—Kunst.* Berlin: Elsengold Verlag GmbH, 2016.

[15] Wikipedia. *Alhambra.* `en.wikipedia.org/wiki/Alhambra`

[16] Wikipedia. *Owen Jones.* `en.wikipedia.org/wiki/Owen_Jones_(architect)`

[17] Wikipedia. *Tessellation.* `en.wikipedia.org/wiki/Tessellation`

[18] Wilson, Eva. *Islamic Designs for Artists and Craftspeople.* Mineola: Dover Publications, Inc., 1988.

[19] Wolfram Mathworld. *Regular Tessellation.* `mathworld.wolfram.com/RegularTessellation.html`

◇ Laurence Finston
Germany
`Laurence dot Finston (at) gmx dot de`