An Algorithmic Approach to Chinese Lattice Design

Miroslaw Majewski (馬裕祺) *Jiyan Wang* (王继延)
mirek.majewski@yahoo.com jywang@math.ecnu.edu.cn
School of Arts & Sciences Department of Mathematics
New York Institute of Technology East China Normal University
Abu Dhabi campus, UAE Shanghai, China

Abstract: *Chinese Lattices are popular artifacts, frequently used in Chinese windows, doors and decorative furniture. In this paper we will look at them from an algorithmic point of view. We will be interested in developing the simplest possible algorithms that one can use to create such lattices with a computer or model them with wooden sticks. To develop these algorithms, we will use MATLAB® Symbolic Math Toolbox®. All algorithms in this paper will be written in the form of MuPAD® programs[1]. Throughout the paper, we will use only short snippets of MuPAD code. The complete MuPAD programs will be enclosed in the appendices.*

1 Introduction

The climate of South-East Asia is hot and humid. For this reason, traditional architecture in all the countries in this region is very spacious and contains a lot of openings. In order to protect the inhabitants and to stop birds and small animals entering houses, people protected window and door openings using wooden grids. Later, these wooden grids evolved into a very sophisticated form of applied art known as Chinese lattices. Although the name suggest China as the place of origin of Chinese lattices, we can find them in many other countries in the Far East, e.g. in Korea, Japan or Vietnam. Today, in many Far East countries Chinese lattices are used not only as protection for window openings but also as dividers of large interiors or as decorations on walls and in furniture.

Chinese lattices from China itself are stylistically slightly different to those from Japan or Korea. However, in each case, we usually deal with a creation that can be very intriguing for a mathematician or a computer scientist.

The photograph on the next page shows Chinese lattices in Zhouzhuang village near Shanghai. The village, or rather a small city, is known as the oldest Chinese village on water. The village was restored recently and it contains a number of 800 year-old houses with very traditional Chinese lattices. We can find similar creations in the imperial palace in Nanjing, in the old towns in Shanghai, Beijing and other Chinese towns, as well as in completely new apartments or villas in China.

[1] Since September 2008 MuPAD® is used as the computing engine for the MATLAB® Symbolic Math Toolbox. MATLAB and Symbolic Math Toolbox are registered trademarks of The MathWorks, Inc. MuPAD is registered trademark of SciFace Software GmbH & Co.KG.

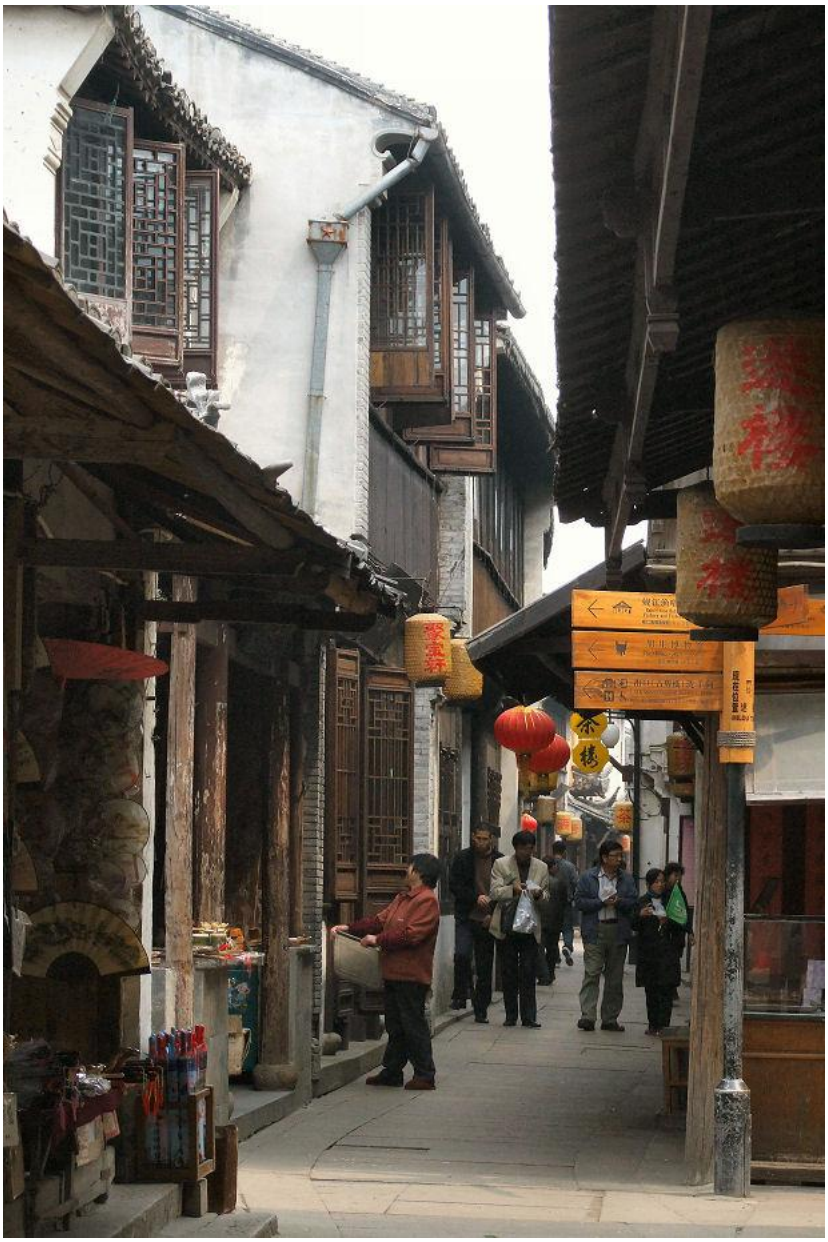


Fig. 1 Street in Zhouzhuang village, most of windows covered by a lattice

Chinese lattices are very interesting objects from a mathematical point of view. They contain many sophisticated mathematical patterns. Analyzing these lattices with Computer Algebra Systems (CAS) often lets us recover the algorithms used to create these patterns. We can model them using any CAS where algorithmic constructions are available. In many cases, a form of turtle graphics and L-systems can be very useful.

In this paper, we will concentrate on three simple examples of Chinese lattices. If we wanted to describe the mathematical nature of some of the more complex lattices, we would need more than a few pages of a journal paper.

In our investigations, we will use MuPAD, a modern Computer Algebra System that is a part of Symbolic Math Toolbox available for Matlab software. MuPAD was developed by a group of scientists from Paderborn University, in Germany. MuPAD's development is now continued by SciFace Software GmbH & Co. KG – a computer company specializing in software for scientific research. MuPAD contains all the tools that we need to model and visualize Chinese lattices (see [3]).

The literature for Chinese lattices is very limited. There are only two books (see [1, 2]) written by Daniel Sheets Dye – former professor at West China Union University Chengtu. These two books contain thousands of drawings of Chinese lattices from various places in West China collected by the author since 1916 until his death on January 16, 1936. There is not much written information about these examples – just some basic classification and location data. In China there is available a reference book (see [4]) on classical Chinese windows and doors with hundreds of photographs showing Chinese lattices. There are very few papers on Chinese lattices and only one or two simple web pages on the Internet.

2 Getting started with algorithms

In this paper we will briefly describe three methods of creating Chinese lattices. However, we are not trying to make any systematic analysis of algorithms used to create such lattices. We also do not claim that the methods described here are the only ones that exist. We will concentrate mostly on two of these methods by showing examples of algorithms. We, the authors, believe that it is worthwhile to investigate the mathematical and algorithmic nature of Chinese lattices and this paper is just a starting point for these investigations.

Grid-based lattices: By analyzing the available images we can observe that many lattices were formed by creating a grid and filling it by a repeating element or elements. The grid can be rectangular, hexagonal, octagonal, triangular, etc. The grid can be uniform or have one, two or more gaps in the middle – the so-called foci of attention. The filling pattern can be placed in a uniform way, i.e. each of its instances oriented the same way, or rotated, or mirrored. A typical example of such a lattice is shown in figure 2. It happens frequently that the whole lattice is filled by a pattern without creating a grid. In such cases, instances of the pattern are joined together without a grid. An example of such a pattern is shown in figure 3.

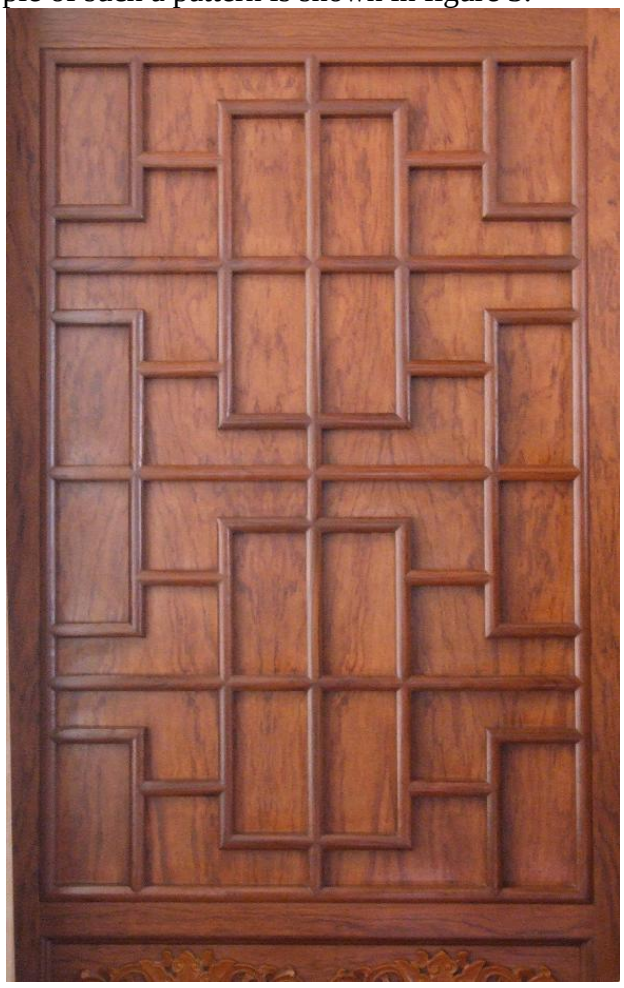


Fig. 2 Rectangular grid with mirrored elements

Fig. 3 Chinese lattice filled with pattern and no base grid

Block-based lattices: There are many Chinese lattices where a simple block was created first and then multiple copies of it assembled in a large lattice with multiple rows and columns. This reminds us tiling techniques where one or more base tiles were used to create a large, sometimes infinite, pattern. In the case of Chinese lattices the pattern is always finite but we can easily imagine that the pattern could extend horizontally and vertically long enough to be considered as infinite. An example of such lattice is shown in figure 4.

Fig. 4 Chinese lattice constructed from multiple identical blocks

Lattices with non-repeating blocks: Finally, there are Chinese lattices where a common grid or repeating blocks cannot be identified. In such cases, the lattice is constructed in a very specific way and its modeling often requires a unique algorithm. In many cases, we can observe similarity to some well known mathematics curves, for example to a space-filling curve, e.g. Peano, Hilbert or Sierpinski curve; or to a specific family of fractals known as L-systems. Figure 5 shows a lattice that was developed in the form similar to a space-filling curve. We can easily imagine that by adding further loops we could produce a curve that would cover the window completely.

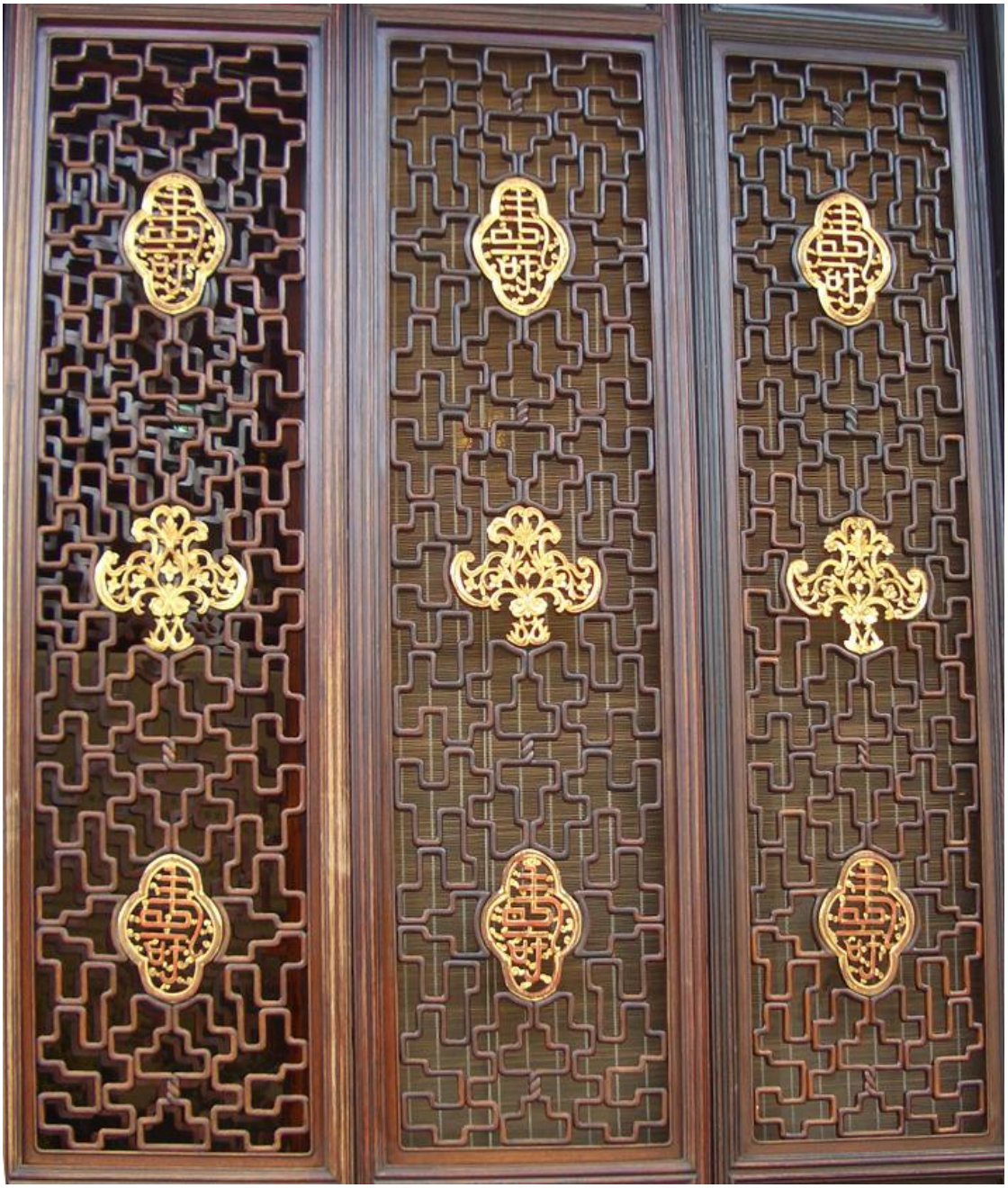


Fig. 5 This lattice was created in the form of a space-filling curve

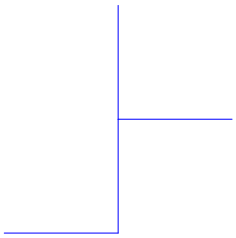
3 Line segments and turtle graphics

Most Chinese lattice patterns can be modeled using line segments in 2D. To do so, we need a set of commands that would allow us to produce line segments and transform the obtained geometry objects. MuPAD gives us two methods of working with line segments.

The coordinate method: a method using coordinates of points in 2D or 3D and the commands `plot::Line2d([x1,y1],[x2,y2])` to create a line segment in 2D, and `plot::Line3d([x1,y1,z1],[x2,y2,z2])` to create line segment in 3D.

The turtle graphics method: a method that does not require coordinates of points, instead using a sequence of instructions describing the movements of the turtle. This may look like the following MuPAD code:

```
Z := plot::Turtle():
Z::right(PI/2):
Z::forward(1):
Z::left(PI/2):
Z::forward(1):
Z::push():
Z::forward(1):
Z::pop():
Z::right(PI/2):
Z::forward(1):
plot(Z);
```



At the start, our turtle is located at the center of the coordinate system and is facing upwards (we call it North). In our code, the first command after initiating the turtle was to order the turtle to turn right (East), and then draw a line segment one unit long; this is then followed by further commands. The concept of the turtle graphic in MuPAD was described in detail in [3].

In many situations, using turtle graphics is more convenient than using the coordinate method – we can more easily follow the pattern of a lattice. In other situations, it will be faster to develop the building blocks using coordinates than with the turtle method. In general, the turtle method will require more code than the coordinate method.

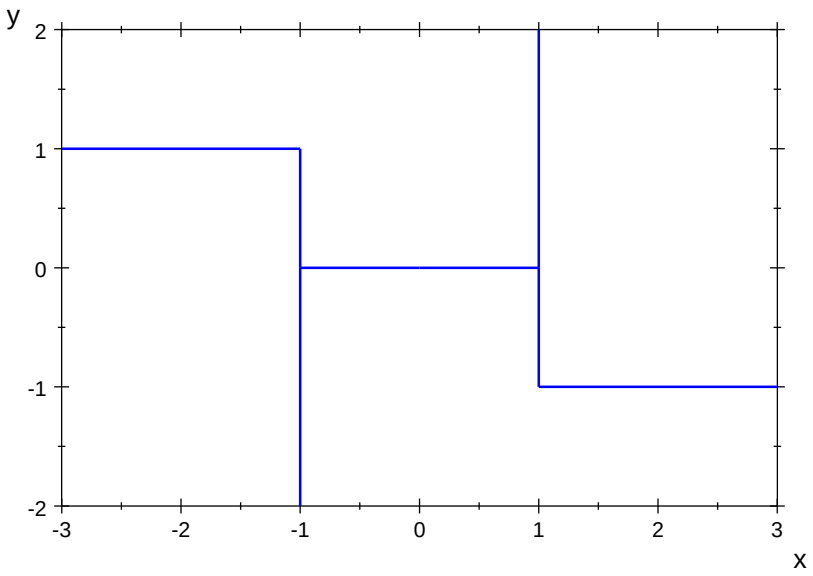
4 Modeling grid-based lattices

Depending on the kind of the grid as well as the complexity of the filling pattern, modeling grid-based lattice can be a more or less time-consuming task. In order to demonstrate how such a lattice can be created we will develop an algorithm for the lattice shown in figure 2. We will start by creating the filling pattern, then we will develop the grid, and finally we will combine both parts of the code in order to produce the complete lattice.

In this example the filling pattern is very simple and it can be obtained by using just a few commands for the turtle. In fact, we can split it into two parts – left and right, model each of them separately and then join them together.

```
R := plot::Turtle(): // here we start creating the right side
R::right(PI/2):
R::forward(1):
R::push():
```

```
R::left(PI/2):
R::forward(2):
R::pop():
R::right(PI/2):
R::forward(1):
R::left(PI/2):
R::forward(2): // here we finished creating the right side
L := plot::Rotate2d(PI, R): // here we create the left side
P1 := plot::Group2d(R,L): // here we assemble the whole pattern
plot(P1, Axes=Boxed)
```



We plotted the pattern and the coordinate system. This way we can see where the pattern is located and what its size is – later we will move it over the plane and we will need exact numbers for translations of the pattern. From the graph we can easily notice that the center of the pattern is exactly in the point (0,0), its width is 6 units and height is 4 units.

In our construction we will need also a mirror copy of the pattern. This can be done by applying a reflection about the line passing through the points (0,1) and (0,-1).

```
P2 := plot::Reflect2d([0,1], [0,-1],P1):
```

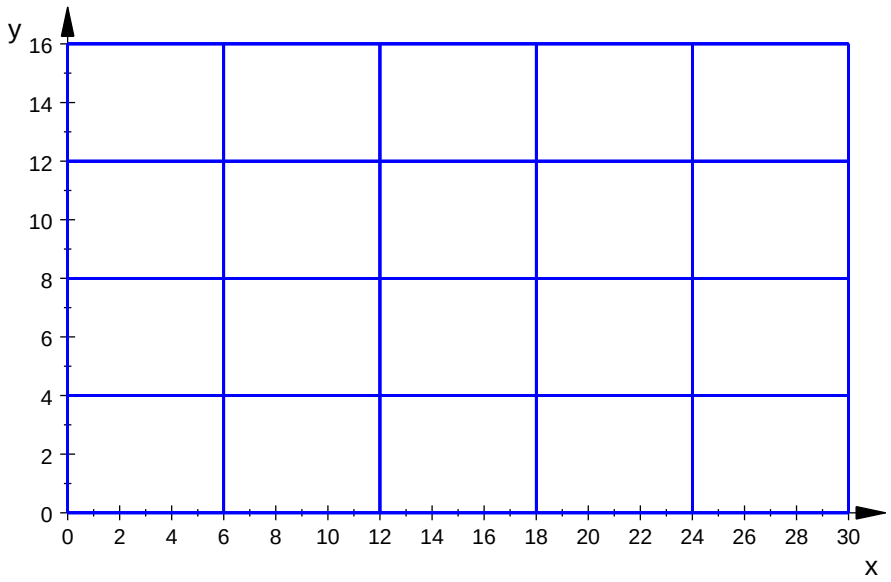
Having both patterns ready, we can think how the grid should be developed. In this example, we can create a simple procedure with the two parameters *n* and *m* to develop a grid with *n*+1 vertical bars and *m*+1 horizontal bars.

```
Grid := proc(n,m)
local grid, i, j, vertical, horizontal;
begin
   grid :=[]:
   for i from 0 to n do
     for j from 0 to m do
       vertical := plot::Line2d([6*i, 0], [6*i, 4*m]):
       horizontal := plot::Line2d([0, 4*j], [6*n, 4*j]):
       grid := grid.[vertical].[horizontal]:
     end_for:
   end_for:
```

```
   return(op(grid))
end_proc:
```

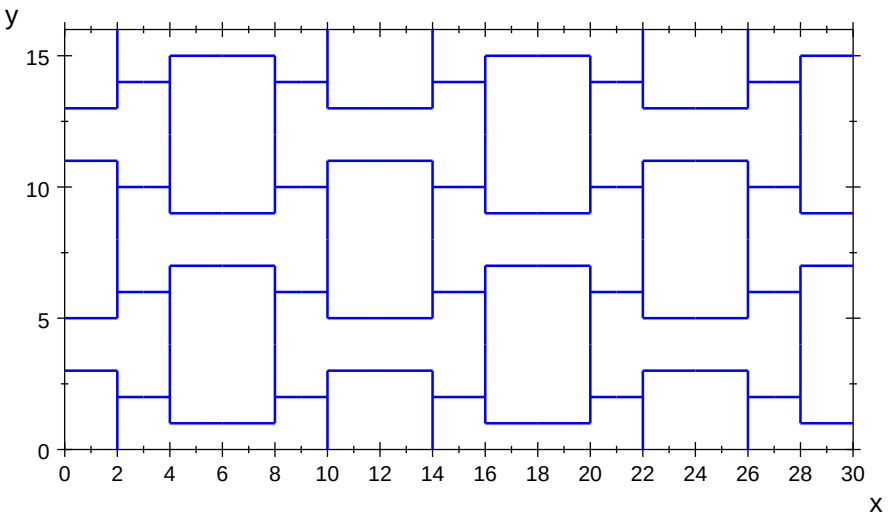
Note, in this procedure we explicitly request that vertical bars will be placed every 6 units and horizontal bars every 4 units – this is related to the dimensions of the filling pattern. The graph below shows what we have obtained with n=5 and m=4.

```
plot(Grid(5,4))
```



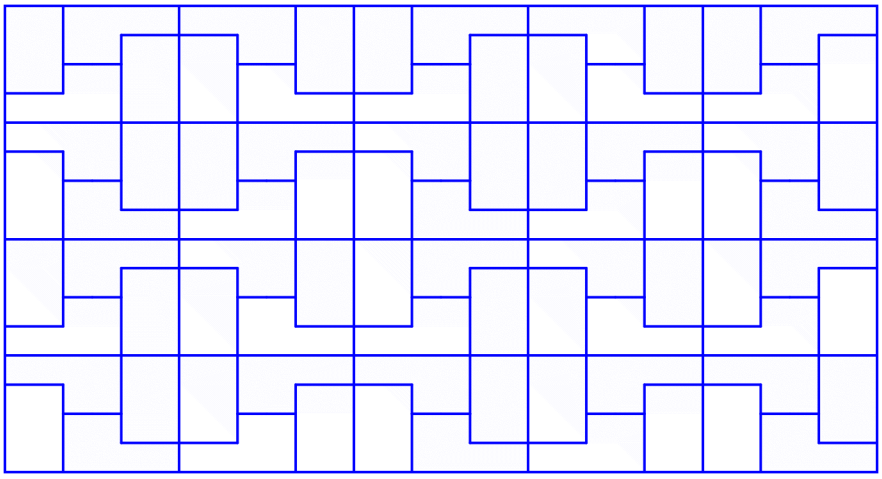
A slightly similar procedure can be used to create the complete filling for the lattice. There are two major changes. We use here the two elements `P1` and its mirror reflection `P2`. We choose them depending on horizontal and vertical coordinates of the grid cell where the pattern will be inserted. For cells where `i+j` is even, we use `P1`, while for cells where `i+j` is odd, we use `P2`. The second change is that each element of the filling is translated additionally 3 units to the right and 2 units up in order to fit it into the right location.

```
Filling := proc(n,m)
local structure, i, j;
begin
   structure := []:
   for i from 0 to n-1 do
     for j from 0 to m-1 do
        if (i+j) mod 2 = 0 then
           element := plot::Translate2d([i*6+3,j*4+2],P1)
        else
           element := plot::Translate2d([i*6+3,j*4+2],P2)
        end_if:
        structure := structure.[element]:
     end_for:
   end_for:
    return(op(structure))
end_proc:

plot(Filling(5,4), Axes=Boxed)
```



Finally, by combining the filling and the grid we will obtain the complete pattern. We will assemble the code into a small procedure – `ChineseLattice(n,m)`. The role of parameters *n* and *m* is the same as shown in the previous procedures – to define number of columns and rows for the whole lattice. Here is the procedure and the obtained output from it.

```
ChineseLattice := proc(n,m)
begin
   plot::Group2d(Grid(n,m), Filling(n,m)):
end_proc:

plot(ChineseLattice(5,4))
```



A complete program for this lattice, with a few minor improvements, is enclosed in appendix 1.

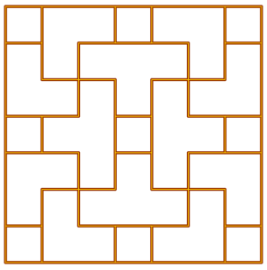
Fig. 6 Lattices on the balcony of Zhouzhuang's entrance gate

5 Block-based lattices – the Zhouzhuang lattice case study

Block-based lattices are very popular in door and window construction. We will investigate this case using a lattice that is frequently used in Shanghai area. For the purpose of this paper, we will refer to it as the Zhouzhuang lattice.

The very first Chinese lattice that everybody can see when entering Zhouzhuang is the lattice on the balcony of the entrance gate (see fig. 6). The same or very similar pattern can be found in many other places in Zhouzhuang. This is the pattern that we will call the Zhouzhuang lattice[2]. The lattice looks very simple and we will use it in our first investigation.

There are five windows, all with an identical lattice. We can also see that each lattice itself is built out of repeating blocks separated by horizontal lines.



Finally each of these blocks (see the picture next to this text) can be decomposed to four, almost identical parts: bottom-left, bottom-right, top-left and top-right. These four parts overlap in the center of the block. The central square is the common element of all four parts.

In fact, the bottom-left and top-right parts are identical, while the bottom-right and top-left parts are a mirror reflection of the bottom-left. Therefore, we only have to create one of these parts and then transform them to obtain the three other parts.

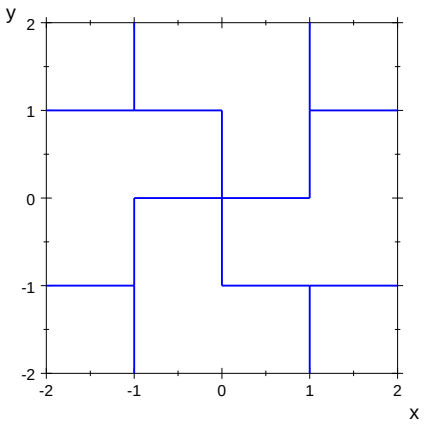
We will start our construction by creating the swastika-like pattern that forms the bottom-left part of a single block.

The MuPAD turtle code for the bottom-left part of the block will look as follows:

```
Z := plot::Turtle():
```

[2] It is important to note that the Zhouzhuang lattice does not exist in Daniel Sheets Dye's book, though a similar pattern is mentioned as L8a ([1], page 203).

```
Z::right(PI/2):
Z::forward(1):
Z::left(PI/2):
Z::forward(1):
Z::push():
Z::forward(1):
Z::pop():
Z::right(PI/2):
Z::forward(1):

Cross1 := plot::Group2d(
   Z,
   plot::Rotate2d(Z,PI/2),
   plot::Rotate2d(Z,PI),
   plot::Rotate2d(Z,3*PI/2)
 ):

plot(Cross1,Axes=Boxed)
```



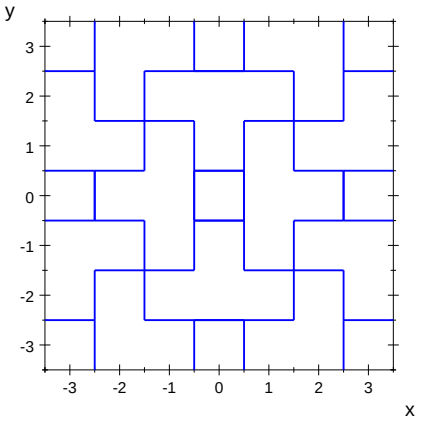
While plotting this pattern, we deliberately forced MuPAD to plot the axes of the coordinate system. This way, we are able to check the size of the pattern and its location. Note – the swastika-like cross was obtained by taking four rotated instances of the very same zigzag that we had created while demonstrating how turtle graphics work.

We can now produce a mirrored version of this cross. We take `Cross1` and we reflect it about the line passing through the points (0,-1) and (0,1).

```
Cross2 := plot::Reflect2d([0,-1],[0,1], Cross1):
```

Having these two elements ready, we can start assembling the interior pattern of the whole block.

```
TL := plot::Translate2d([-1.5,  1.5], Cross2):
TR := plot::Translate2d([ 1.5,  1.5], Cross1):
BL := plot::Translate2d([-1.5, -1.5], Cross1):
BR := plot::Translate2d([ 1.5, -1.5], Cross2):

plot(TL, TR, BL, BR, Axes=Boxed)
```

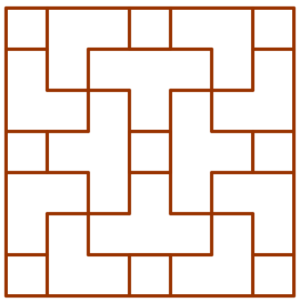


In order to complete the whole block we need a frame around of it. This can be done with a few simple orders for the turtle. We will simplify our work by writing a simple program using a for-do loop.

```
F := plot::Turtle():
for i from 1 to 4 do
   F::forward(7), F::right(PI/2)
end_for:
F := plot::Translate2d([-3.5,-3.5],F):
plot(F, Axes=Boxed)
```

By grouping the obtained parts `BL`, `BR`, `TL`, `TR` and `F` we produce the whole block.

```
Net1 := plot::Group2d(TL, TR, BL, BR, F, LineWidth=4*unit::pt, LineColor=RGB::Brown):
Net2 := plot::Group2d(TL, TR, BL, BR, F, LineWidth=2*unit::pt, LineColor=RGB::Gold):
plot(Net1)
```



Finally, we can develop a procedure that will allow us to replicate the obtained block rightwards and upwards as many times as we wish.

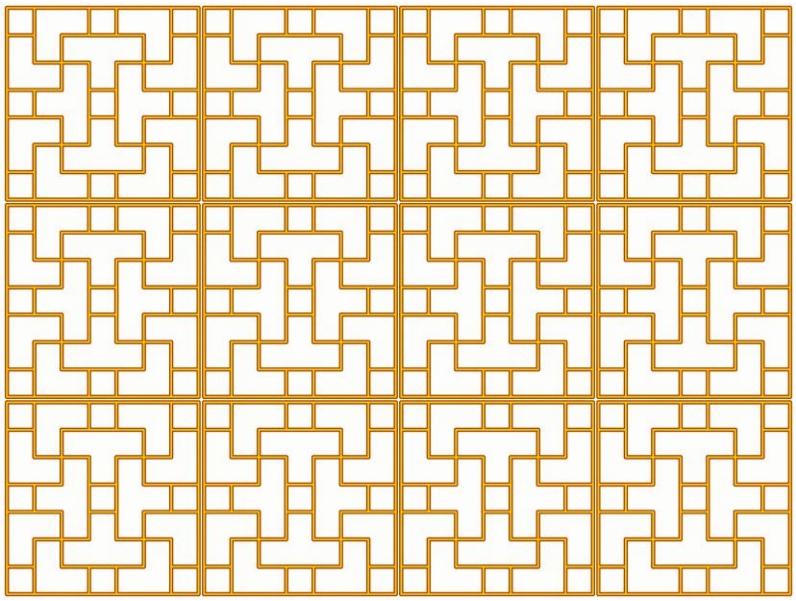
```
Lattice := proc(thing, w, h, n, m, offset)
begin
   objects := []:
   for i from 1 to n do
      for j from 1 to m do
         objects := objects.[plot::Translate2d([i*(w+offset),j*(h+offset)], thing)]
      end_for
   end_for,
   return(op(objects))
```

```
end_proc:
```

The procedure uses six input parameters: `thing` – this is the object that we wish to repeat; `w` and `h` are width and height of the block respectively; `n` and `m` are the number of times we wish to repeat the block to the right and up. Finally `offset` is the small gap that we have to add between two neighboring blocks to avoid overlapping them.

The code below will create a Zhouzhuang lattice with 3 rows and four columns. In order to make the final result more appealing, we produce two blocks: `Net1` with 4 points line width (brown color), and `Net2` with 2 point line width (gold color); then we overlap them.

```
plot(Lattice(Net1, 7, 7, 4, 3, 0.2), Lattice(Net2, 7, 7, 4, 3, 0.2))
```



The complete program for this lattice is enclosed in appendix 2.

6 A lattice based on the Peano curve

While analyzing existing examples of Chinese lattices, we can often spot a pattern that resembles one of the well-known mathematical curves. In this example, we will look on the Peano curve and show how we can construct a Chinese lattice based on this curve. Patterns similar to the lattice developed in this section can be found in Chengtu in Szechwan province (see [1] page 43).

Let us recall the shape and properties of the Peano curve. There are many ways of defining a Peano curve. One of the simplest definitions uses the L-system concept.

In order to create a Peano curve we will need an L-system with the following axioms:

1. Variables: *F*
2. Constants: +, -
3. Start: *F*
4. Processing rule: *F = F+F-F-F-F+F+F+F-F*

Then we proceed through string rewriting iterations. In the step 0 we have the string *F*; in the step 1 the string *F* will be replaced by *F+F-F-F-F+F+F+F-F*; in the step 2 each *F* in the last resulting string will be replaced by *F+F-F-F-F+F+F+F-F-F*; and this process we may continue as many times as we wish. We can learn more about L-systems in [3] and [5].

We can use turtle graphics to illustrate any L-system. The turtle moves a given step forward each time (this is *F*) and can turn left (-) or right (+) at a given angle. For a Peano curve the angle should be $\pi/2$. This way the turtle will draw a graph illustrating the given L-system. Figure 7 shows graphs representing three consecutive approximations of the Peano curve. The angle used here is slightly smaller than $\pi/2$ in order to demonstrate that the curve does not intersect itself.

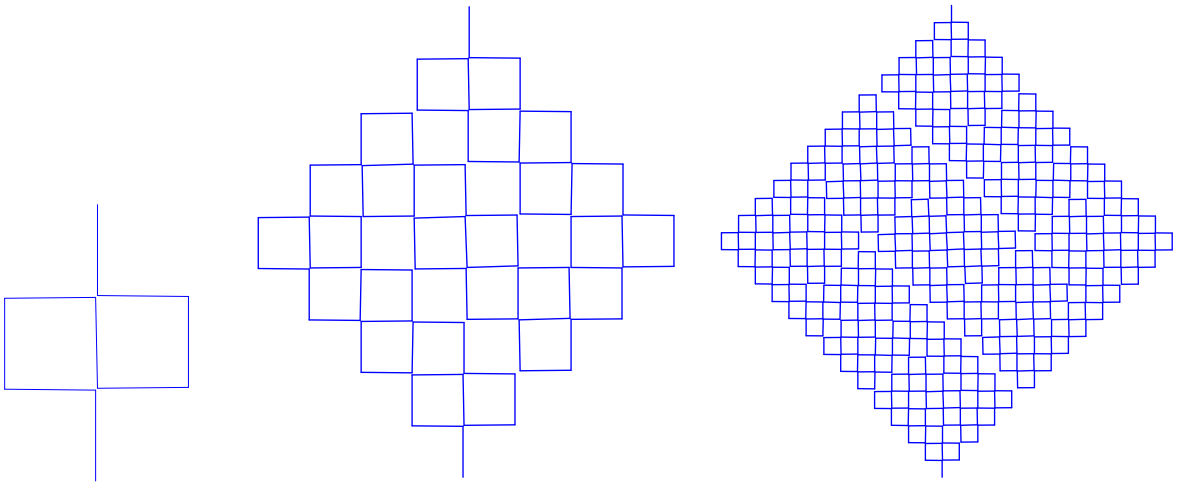


Fig. 7 Three approximations of the Peano curve (1st, 2nd and 3rd generations)

In our construction, we will use the curve obtained in the second generation. This way, it will be easier to demonstrate the concept. However, the curve obtained in any later generation could be used as well, in which case the resulting lattice would be larger and more complex.

Let us start with a simple command to create the Peano curve of second generation in MuPAD.

```
Peano := plot::Lsys(PI/2, "F", "F" = "F+F-F-F-F+F+F+F-F", Generations=2):
```

Now we can produce two instances of the Peano curve – one with line width 4 points, for the outline, and another one with line width 2 points, for the interior of the pattern. This way we will be able to obtain a double line effect.

```
PeanoOutline := plot::modify(Peano, LineWidth=4*unit::point, LineColor=RGB::Indigo):
PeanoInterior := plot::modify(Peano, LineWidth=2*unit::point, LineColor=RGB::Gold):
```

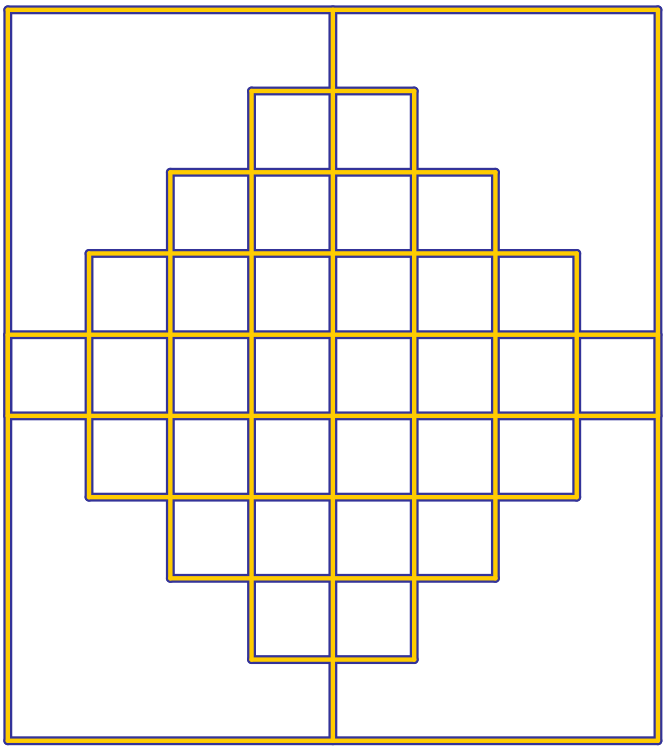
In exactly the same way we will create the frame around.

```
FrameOutline := plot::Rectangle(-4.0..4.0,-0.0..9.0,
   LineWidth=4*unit::pt,
   LineColor=RGB::Indigo
):
FrameInterior := plot::Rectangle(-4.0..4.0,-0.0..9.0,
   LineWidth=2*unit::pt,
   LineColor=RGB::Gold
):
```

Finally, we can combine all obtained elements into a lattice and plot the lattice.

```
PeanoLattice := plot::Group2d(
   FrameOutline,
   PeanoOutline,
```

```
    PeanoInterior,
    FrameInterior
):
plot(PeanoLattice)
```



The obtained lattice can be modified by inserting additional horizontal or vertical bars joining the interior with the frame, and filling the large empty space in each corner. The complete MuPAD program for the Peano lattice is enclosed in appendix 3.

7 Summary

Modeling Chinese lattices with MuPAD can be a very interesting activity. Some of the patterns are relatively easy to develop using simple elements followed by translations and rotations, finally grouping the simple patterns into more complex ones. Some of the lattices may require a completely different approach. For example, many of them will require a continuous turtle path covering the whole lattice. Such a lattice may be then obtained using one or more turtle paths combined together. Then, there are lattices where an application of the L-system method seems to be appropriate. In such cases, higher generations of the L-system may lead to new, very complex patterns.

In every case, the method that we use to model a lattice should be as natural as possible, and probably should mimic the way of thinking of the craftsman designing the real lattice. For example, it is natural to create the main blocks of the interior of a lattice, then join them together and combine with a frame.

8 References

[1] Dye D. S., Chinese Lattice Designs, Dover Publications, Inc., New York, 1974.

[2] Dye D. S., The New Book of Chinese Lattice Designs, Dover Publications, Inc., New York, 1981.
[3] Majewski M., Getting Started with MuPAD, Springer Verlag, Berlin-Heidelberg-New York, 2005.
[4] Weidu Ma, Classical Chinese Doors and Windows, China Architecture & Building Press, Beijing, 2006.
[5] Prusinkiewicz P., Lindemayer A., The Algorithmic Beauty of Plants (the virtual laboratory), Springer, March 27, 1996.

Appendix 1: Complete MuPAD program for the grid-based lattice

```
ChineseLattice01 := proc(n,m)
local R,L, P1, P2, Grid, Filling;
begin
// commands to create one side of the basic pattern
   R := plot::Turtle():
   R::right(PI/2):
   R::forward(1):
   R::push():
   R::left(PI/2):
   R::forward(2):
   R::pop():
   R::right(PI/2):
   R::forward(1):
   R::left(PI/2):
   R::forward(2):              //here is right side of the basic pattern
   L := plot::Rotate2d(PI, R): // here is left side of the basic pattern
// complete basic pattern normal and mirrored version
   P1 := plot::Group2d(R,L):
   P2 := plot::Reflect2d([0,1], [0,-1],P1):

//Sub-procedure to create the frame
   Grid := proc(n,m)
   local grid, i, j, vertical, horizontal;
   begin
      grid :=[]:
      for i from 0 to n do
         for j from 0 to m do
            vertical := plot::Line2d([6*i, 0], [6*i, 4*m]):
            horizontal := plot::Line2d([0, 4*j], [6*n, 4*j]):
            grid := grid.[vertical].[horizontal]:
         end_for:
      end_for:
   return(op(grid))
end_proc:

//Sub-procedure to create filling patterns
Filling := proc(n,m)
local structure, i, j;
begin
   structure := []:
```

```
   for i from 0 to n-1 do
      for j from 0 to m-1 do
         if (i+j) mod 2 = 0 then
            element := plot::Translate2d([i*6+3,j*4+2],P1)
         else
            element := plot::Translate2d([i*6+3,j*4+2],P2)
         end_if:
         structure := structure.[element]:
      end_for:
   end_for:
   return(op(structure))
end_proc:

// Now we group the framework and the filling.
// We create two identical objects, one with thin line and another
// one with thick line. This way we will get a kind of 3D effect.

   plot::Group2d(
      plot::Group2d(Grid(n,m), Filling(n,m),
         LineWidth=3*unit::pt, LineColor=RGB::Indigo),
      plot::Group2d(Grid(n,m), Filling(n,m),
         LineWidth=1*unit::pt, LineColor=RGB::Yellow)
   ):
end_proc:
```

Appendix 2 – Complete MuPAD program for the block-based lattice

```
ChineseLattice02 := proc(n,m)
local Z, Cross1, Cross2, TL, TR, BL, BR, F, MPattern;
begin
   Z := plot::Turtle():
   Z::right(PI/2):
   Z::forward(1):
   Z::left(PI/2):
   Z::forward(1):
   Z::push():
   Z::forward(1):
   Z::pop():
   Z::right(PI/2):
   Z::forward(1):

   Cross1 := plot::Group2d(
      Z,
      plot::Rotate2d(Z,PI/2),
      plot::Rotate2d(Z,PI),
      plot::Rotate2d(Z,3*PI/2)
   ):

   Cross2 := plot::Reflect2d([0,-1],[0,1],Cross1):
   TL := plot::Translate2d([-1.5, 1.5], Cross2):
   TR := plot::Translate2d([1.5,1.5], Cross1):
   BL := plot::Translate2d([-1.5, -1.5], Cross1):
```

```
   BR := plot::Translate2d([1.5,-1.5], Cross2):

// frame around the main pattern
   F := plot::Turtle():
   for i from 1 to 4 do
      F::forward(7), F::right(PI/2)
   end_for:
   F := plot::Translate2d([-3.5,-3.5],F):

   A1 :=  plot::Group2d(TL, TR, BL, BR, F,
             LineWidth=3*unit::pt, LineColor=RGB::Indigo):
   A2 :=  plot::Group2d(TL, TR, BL, BR, F,
             LineWidth=2*unit::pt, LineColor=RGB::Yellow):
   MPattern := plot::Group2d(A1,A2):

   Lattice := proc(thing, w, h, n, m, offset)
     begin
        objects := []:
        for i from 1 to n do
           for j from 1 to m do
              objects := objects.[plot::Translate2d(
                 [i*(w+offset),j*(h+offset)], thing)]
           end_for
        end_for,
      return(op(objects))
  end_proc:

  Lattice(MPattern, 7, 7, n, m, 0.1)
end_proc:
```

Appendix 3 – Complete MuPAD program for Peano lattice

```
PeanoLattice := proc()
local Peano, FrameOutline, FrameInterior, PeanoOutline, PeanoInterior;
begin
   Peano := plot::Lsys(PI/2, "F", "F" = "F+F-F-F-F+F+F+F-F", Generations=2):

   FrameOutline := plot::Rectangle(-4.0..4.0,-0.0..9.0,
      LineWidth=4*unit::pt,
      LineColor=RGB::Indigo
   ):

   FrameInterior := plot::Rectangle(-4.0..4.0,-0.0..9.0,
      LineWidth=2*unit::pt,
      LineColor=RGB::Gold
   ):

   PeanoOutline := plot::modify(Peano,
      LineWidth=4*unit::point,
      LineColor=RGB::Indigo
   ):
   PeanoInterior := plot::modify(Peano,
       LineWidth=2*unit::point,
       LineColor=RGB::Gold
```

```
   ):

   plot::Group2d(
      FrameOutline,
      PeanoOutline,
      PeanoInterior,
      FrameInterior
   ):
end_proc:
```