A 3D reconstruction of M.C.Escher's "House of Stairs"

P. Baillehache

March 1, 2020

Contents

1 Introduction

M.C.Escher is a Dutch graphic artist who lived from 1898 to 1972. His art is characterized by a strong bond with mathematics, and that's why I'm particularly fond of it. I've received a few reproduction as a calendar a few year

ago, and among them the "House of Stairs" (ref. XL-51). This monochrom lithograph was made in 1951 and represents the interior of a building criss crossed by stairs on which salamander-like creatures crawl in chain.

As usual with Escher's art, the building looks at first sight odd, distorted, irrealistic. But with care it is possible to understand what's going on, and from there it became interesting to me to try to reproduce it, because as everybody knows "if you want to understand it, code it !". Then I set up the goal of creating a reproduction of this lithograph as a 3D synthesis picture.

If POV-Ray wasn't my natural apriori choice for making synthesis picture, mimicking the mathematical approach of the original would have been a good enough reason to use this software anyway. Thus my challenge became: reach the nearest possible reproduction of the "House of Stairs" using POV-Ray.

2 First analysis of the original lithograph

2.1 Dimensions

The original lithograph measures 238mm by 472mm. At 400dpi it converts to 3748px by 7433px, the size of my final image. For fast rendering during development smaller dimensions will be used: 375x743px.

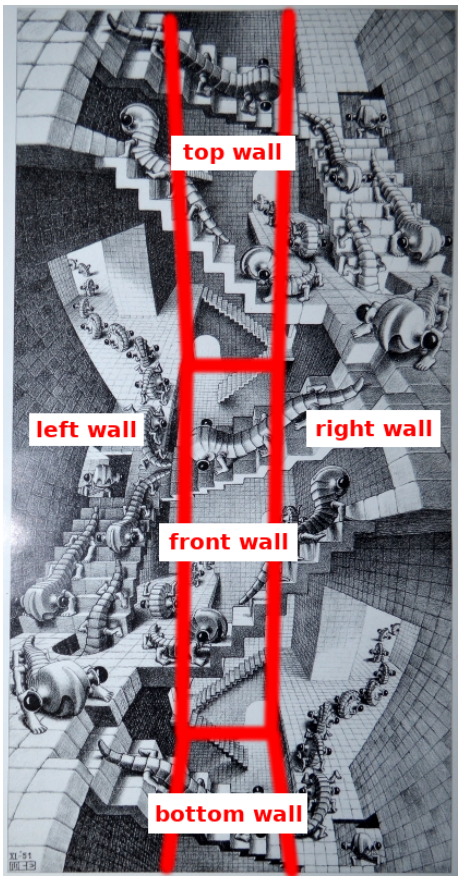
2.2 Colors

The lithograph is monochrom, so a starting approximation for the texture will be white (rgb 255,255,255), a little bit rough, and the shades of gray will come from the lighting. An exception will be made for the eyes of the Curl-ups, which will be in first approximation black, highly polished and reflective.

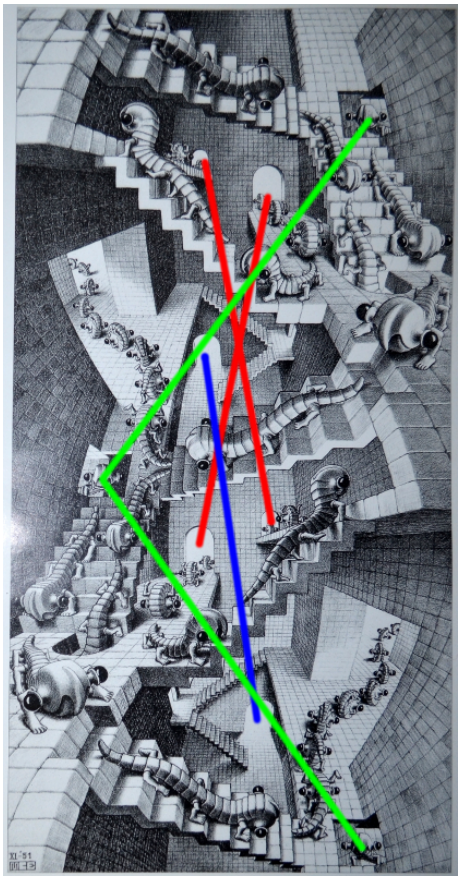
2.3 Building structure

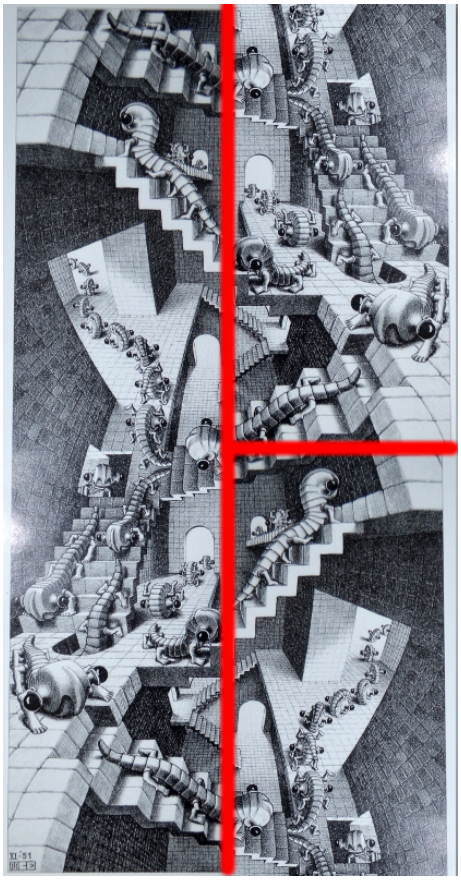
The House of stairs as seen by tiewer is made of 5 faces connected by stairs. Of these faces only one face (the one in front of the viewer) is entirely included in the image. Let's call this face the front wall.

The face on the left side of the image will be called the left wall, the one on the right side of the image will be called the right wall, the one on the center bottom will be called the bottom wall, and the one on the center top of the image will be called the top wall.



One can notice the symmetry in the visible portion of the walls. The top wall is the horizontal mirrored image of the bottom of the front wall, and the bottom wall is the horizontal mirrored image of the top of the front wall. This symmetry applies also to the Curl-ups, with the exception of the top-right and bottom-right corners of the image, where two other Curl-ups should be present to preserve the symmetry.

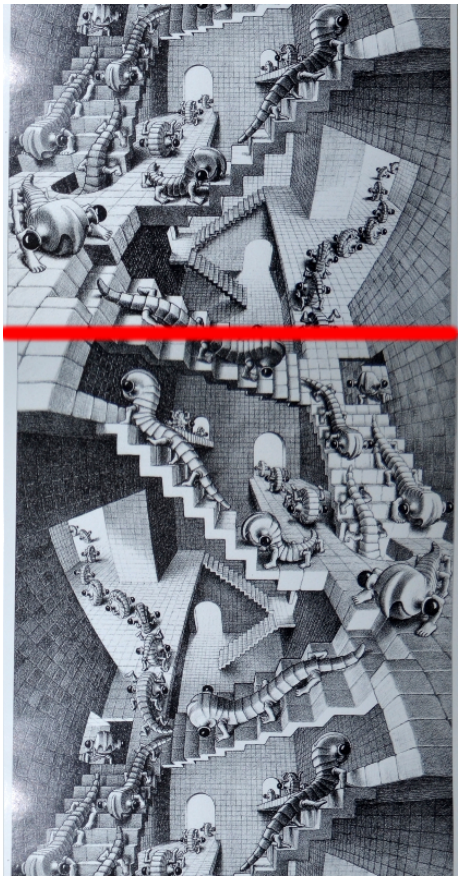




From the two observations above I'll make the following hypothesis:

Hypothesis 1: the right side of the image is the horizontal mirrored image of the left side of the image, shifted along the top-bottom axis.

Next we can also notice that the top of the image repeats the bottom of the image.



Which leads to another hypothesis:

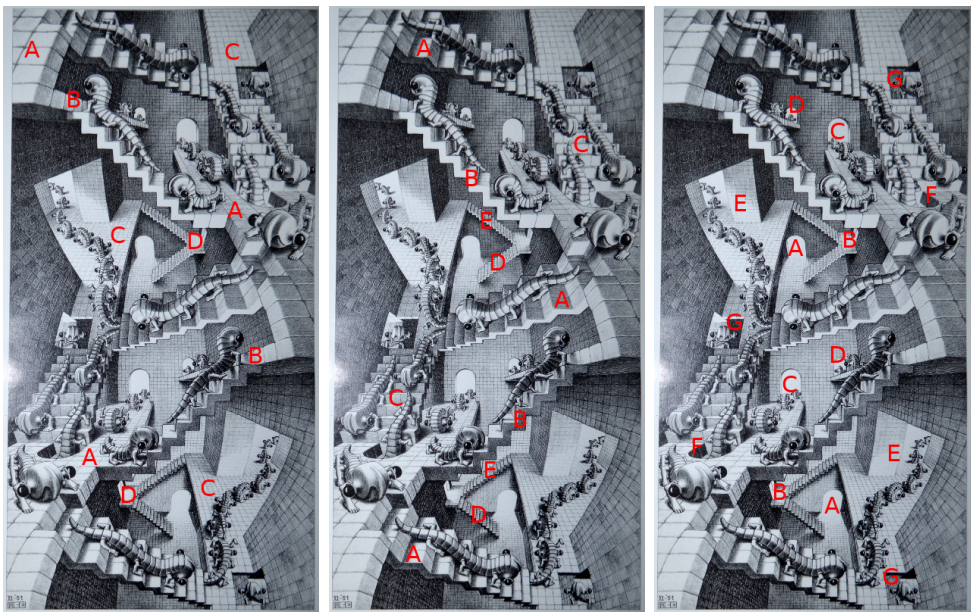
Hypothesis 2: the image repeats infinitely along the top-bottom axis.

On the base of the following observation, I'll make another hypothesis: the walls which looks concave in the image are actually flat rectangles at right angle from one another, and looks concave due to the optical properties of the viewer.

Hypothesis 3: walls are flat rectangles at right angle from one another.

If the previous hypothesis are true, the building is actually a paralellepiped whose left half section is the symmetry of the right half section rotated by 90 degrees relatively to the left-right axis in the image passing through its center.

Lets also define names for the platforms, stairs and doors.



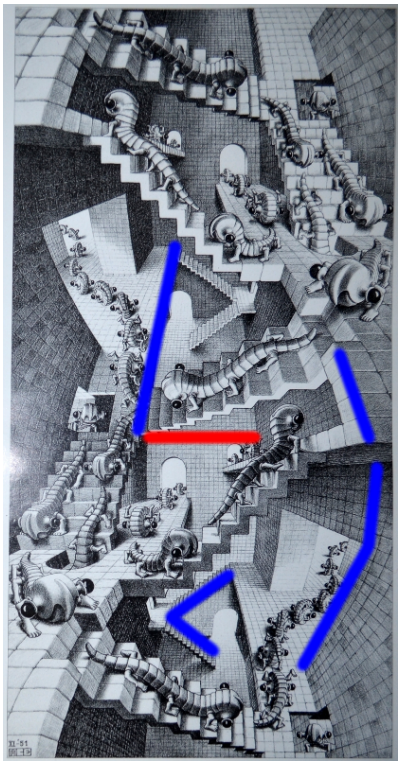
2.4 Building dimensions

The building's wall is almost (exception for the doors and stairs) entirely made of supposedly cubic blocks. Without any other indication for sizes, I'll take it as my reference. So the unit of measure will be one block (1bl) and I make the following hypothesis:

Hypothesis 3: the blocks the walls are made of are cubes.

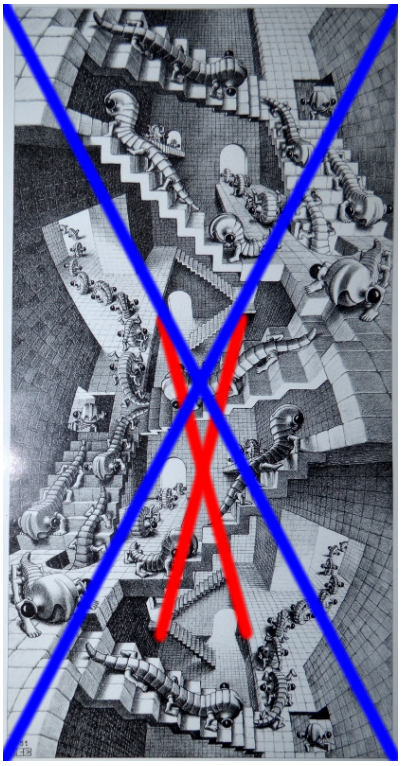
The size of the front wall along the left-right axis can be directly measured: it's 25bl.

However its size along the up-down axis can't directly be measured as the stairs partially hide the front wall. However, it is possible to measure it by using the previous hypothesis: invisible portions of the front wall can be deducted by its equivalent in repeated top/bottom walls and mirrored side of the image. It's 102bl.



2.5 View point

By tracing the diagonals of the lithograph and those of the front wall we see that the viewer stands aligned with the center of the front wall along the left-right axis, but is shifted along the top-down axis toward the bottom of the image.



Without any other information I'll make the following hypothesis:

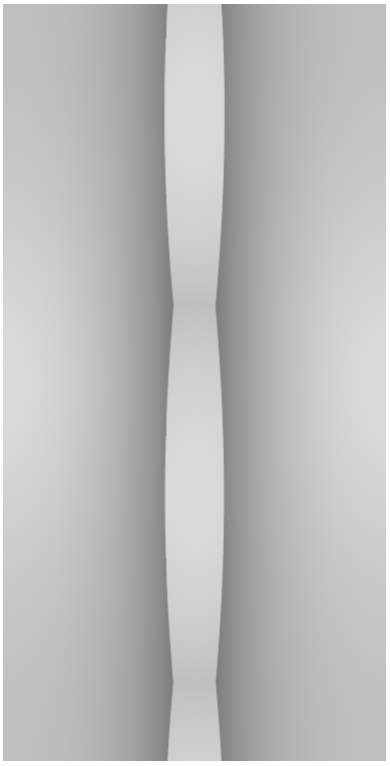
Hypothesis 4: the view point is located at the center of the House of stairs, looking toward the front wall, at its center along left-right axis, and one quarter of its length from its top in the image.

Following the previous hypothesis I can infer some properties of the optic of the viewer: an ultra wide lens spanning 180 degrees along left-right and top-bottom axis.

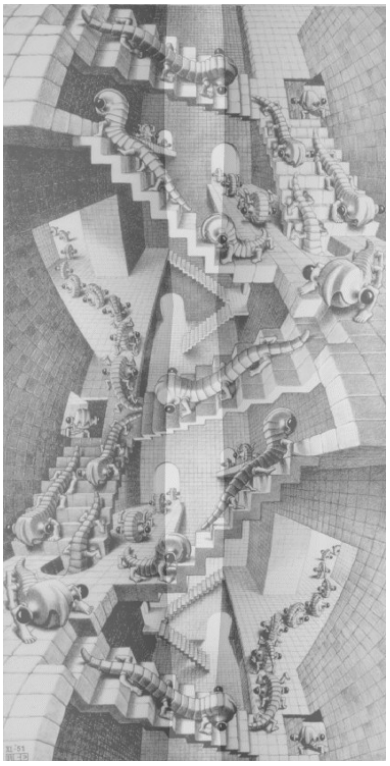
3 Prototype model

I've first make a prototype model of the walls without doors, stairways, platforms and Curl-ups to verify the hypothesis about the optic and dimensions. It comes as follow:

```
#include "colors.inc"

// Unit is one block

// Width of the room (left-right axis of the lithography)
#declare widthRoom = 25.0;
// Height of the room (top-bottom axis of the lithography)
#declare heightRoom = 102.0;

// Texture of the blocks
#declare texBlock = texture {
  pigment { color White }
}

// Camera definition
#declare posCamera = <0.0, 0.0, 0.0>;
#declare lookAt = <0.0, 0.25 * heightRoom, -0.5 * heightRoom>;
camera {
  ultra_wide_angle
  angle 180
  location posCamera
  look_at lookAt
  right x
  up y
}

// Light source
light_source {
  posCamera
  color rgb 1.0
}

// House of stairs
#declare HouseOfStairs = difference {
  box {
    <-0.5 * widthRoom, -0.5 * heightRoom, -0.5 * heightRoom>
    <0.5 * widthRoom, 0.5 * heightRoom, 0.5 * heightRoom>
    scale 1.1
  }
  box {
    <-0.5 * widthRoom, -0.5 * heightRoom, -0.5 * heightRoom>
    <0.5 * widthRoom, 0.5 * heightRoom, 0.5 * heightRoom>
  }
  texture {
    texBlock
  }
}

// Whole scene
object {
  HouseOfStairs
}
```



A superposition with the lithography shows that the model is globally correct but the front, top, bottom walls are half too narrow. The top and bottom of the front wall also don't match the original ones.



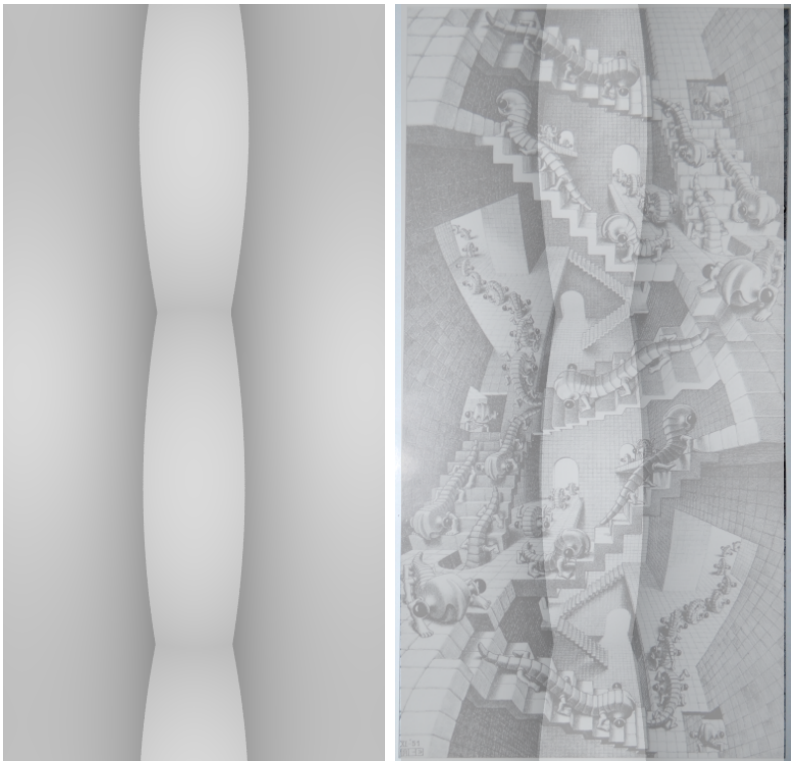
Looking at the block of the platform on the center right of the lithography I add another hypothesis:

Hypothesis 4: one block unit along the left-rght axis is twice as long as one block unit along the top-bottom axis.

Then I tune empirically the angle of the ultra wide lens (200 degrees), and the position of the view point (0.2375 * lengthRoom from the top of the fron wall). Which brings to a satisfying approximation of the room.

```
#include "colors.inc"

// Unit is one block size

// Width of the room (left-right axis of the lithography)
#declare widthRoom = 25.0;
// Length of the room (top-bottom axis of the lithography)
#declare lengthRoom = 102.0;

// Scale of the blocks
#declare scaleBlock = <1.0, 0.5, 0.5>;
```

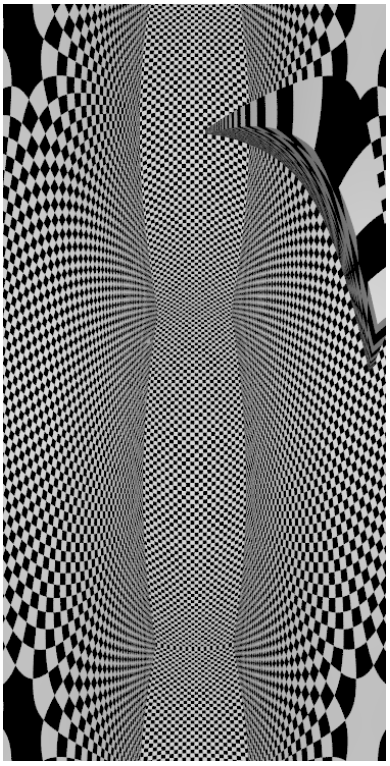
```
// Texture of the blocks
#declare texBlock = texture {
  pigment { color White }
}

// Camera definition
#declare posCamera = <0.0, 0.0, 0.0>;
#declare lookAt = <0.0, 0.2375 * lengthRoom, -0.5 * lengthRoom>;
camera {
  ultra_wide_angle
  angle 200
  location posCamera
  look_at lookAt
  right x
  up y
}

// Light source
light_source {
  posCamera
  color rgb 1.0
}

// House of stairs
#declare HouseOfStairs = difference {
  box {
    -0.5, 0.5
    scale scaleBlock
    scale <widthRoom, lengthRoom, lengthRoom>
    scale 1.1
  }
  box {
    -0.5, 0.5
    scale scaleBlock
    scale <widthRoom, lengthRoom, lengthRoom>
  }
  texture {
    texBlock
  }
}

// Whole scene
object {
  HouseOfStairs
}
```



To give more credit to my hypothesis, I add the right platform and check it matches the lithography. The dimension of the right platform are 50bl long, 10bl large and 1bl thick. It's also 52bl away from the bottom of the front wall. I also replace the white texture with a checker texture to visualize more easily the deformation of the wall through the wide angle lens.

```
#include "colors.inc"

// Unit is one block size

// Width of the room (left-right axis of the lithography)
#declare widthRoom = 25.0;
// Length of the room (top-bottom axis of the lithography)
#declare lengthRoom = 102.0;

// Scale of the blocks
#declare scaleBlock = <1.0, 0.5, 0.5>;

// Texture of the blocks
#declare texBlock = texture {
  //pigment { color White }
  pigment { checker color rgb 0, color rgb 1 }
  scale scaleBlock
}
```

```
// Camera definition
#declare posCamera = <0.0, 0.0, 0.0>;
#declare lookAt = <0.0, 0.2375 * lengthRoom, -0.5 * lengthRoom>;
camera {
  ultra_wide_angle
  angle 200
  location posCamera
  look_at lookAt
  right x
  up y
}

// Light source
light_source {
  posCamera
  color rgb 1.0
}

// House of stairs
#declare HouseOfStairs = union {
  difference {
    box {
      -0.5, 0.5
      scale scaleBlock
      scale <widthRoom, lengthRoom, lengthRoom>
      scale 1.1
    }
    box {
      -0.5, 0.5
      scale scaleBlock
      scale <widthRoom, lengthRoom, lengthRoom>
    }
  }
  box {
    #declare widthPlatform = 10.0;
    #declare lengthPlatform = 50.0;
    #declare heightPlatform = 52.0;
    <-0.5 * widthRoom, 0.5 * lengthRoom, -0.5 * lengthRoom + heightPlatform>
    <-0.5 * widthRoom + widthPlatform, 0.5 * lengthRoom - lengthPlatform,
        -0.5 * lengthRoom + heightPlatform + 1.0>
    scale scaleBlock
  }
  texture {
    texBlock
  }
}

// Whole scene
object {
  HouseOfStairs
}
```

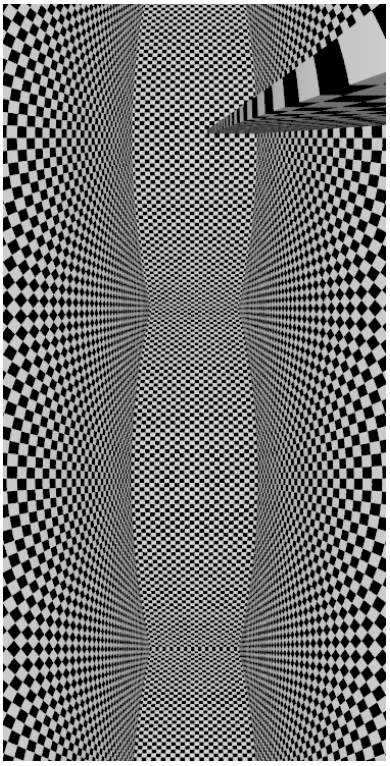


Now I see that the wide angle lens hypothesis is incorrect: in the lithography the horizontal lines are straight but ine the rendered image they are curved. Also, from the position of the platform I see that the camera position is probably not right at the center of the House.

I first try with another type of projection for the camera: the cylinder projection.

```
#include "colors.inc"

// Unit is one block size

// Width of the room (left-right axis of the lithography)
#declare widthRoom = 25.0;
// Length of the room (top-bottom axis of the lithography)
#declare lengthRoom = 102.0;

// Scale of the blocks
#declare scaleBlock = <1.0, 0.5, 0.5>;

// Texture of the blocks
#declare texBlock = texture {
  pigment { checker color rgb 0, color rgb 1 }
```

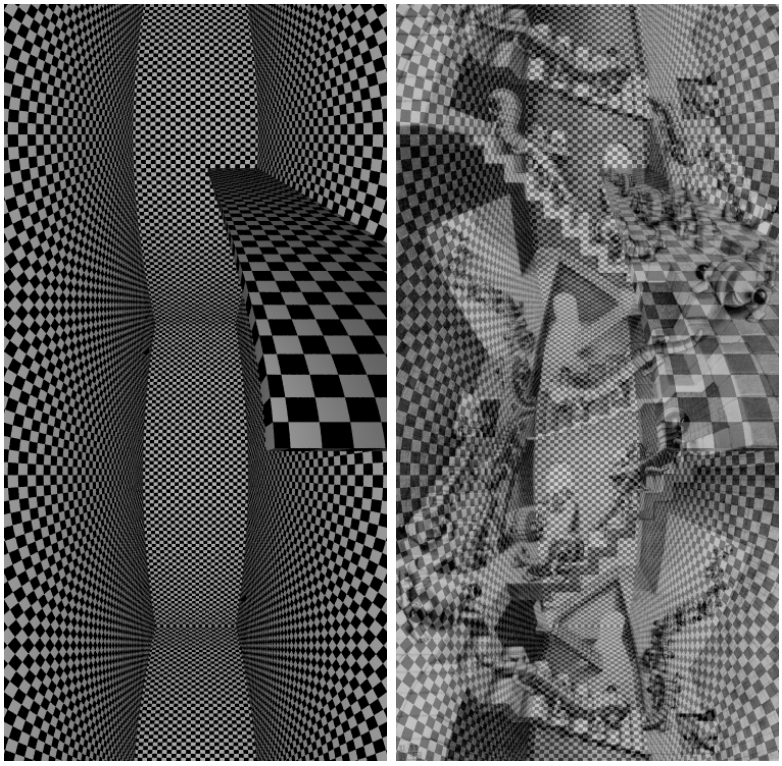
```
  scale scaleBlock
}

// Camera definition
#declare posCamera = <0.0, 0.0, 0.0>;
#declare lookAt = <0.0, 0.2375 * lengthRoom, -0.5 * lengthRoom>;
camera {
  cylinder 2
  angle 200
  location posCamera
  look_at lookAt
  right x * 3.0
  up y
}

// Light source
light_source {
  posCamera
  color rgb 1.0
}

// House of stairs
#declare HouseOfStairs = union {
  difference {
    box {
      -0.5, 0.5
      scale scaleBlock
      scale <widthRoom, lengthRoom, lengthRoom>
      scale 1.1
    }
    box {
      -0.5, 0.5
      scale scaleBlock
      scale <widthRoom, lengthRoom, lengthRoom>
    }
  }
  box {
    #declare widthPlatform = 10.0;
    #declare lengthPlatform = 50.0;
    #declare heightPlatform = 52.0;
    <-0.5 * widthRoom, 0.5 * lengthRoom, -0.5 * lengthRoom + heightPlatform>
    <-0.5 * widthRoom + widthPlatform, 0.5 * lengthRoom - lengthPlatform,
        -0.5 * lengthRoom + heightPlatform + 1.0>
    scale scaleBlock
  }
  texture {
    texBlock
  }
}

// Whole scene
object {
  HouseOfStairs
}
```



The projection looks much better. Now I try to improve the camera position.

```
#include "colors.inc"

// Unit is one block size

// Width of the room (left-right axis of the lithography)
#declare widthRoom = 25.0;
// Length of the room (top-bottom axis of the lithography)
#declare lengthRoom = 102.0;

// Scale of the blocks
#declare scaleBlock = <1.0, 0.5, 0.5>;

// Texture of the blocks
#declare texBlock = texture {
  pigment { checker color rgb 0, color rgb 1 }
  scale scaleBlock
}

// Camera definition
#declare posCamera = <0.0, 0.0, 5.0>;
#declare lookAt = <0.0, 0.25 * lengthRoom, -0.5 * lengthRoom>;
camera {
  cylinder 2
```

```
  angle 200
  location posCamera
  look_at lookAt
  right x * 3.0
  up y
}

// Light source
light_source {
  posCamera
  color rgb 1.0
}

// House of stairs
#declare HouseOfStairs = union {
  difference {
    box {
      -0.5, 0.5
      scale scaleBlock
      scale <widthRoom, lengthRoom, lengthRoom>
      scale 1.1
    }
    box {
      -0.5, 0.5
      scale scaleBlock
      scale <widthRoom, lengthRoom, lengthRoom>
    }
  }
  box {
    #declare widthPlatform = 10.0;
    #declare lengthPlatform = 50.0;
    #declare heightPlatform = 52.0;
    <-0.5 * widthRoom, 0.5 * lengthRoom, -0.5 * lengthRoom + heightPlatform>
    <-0.5 * widthRoom + widthPlatform, 0.5 * lengthRoom - lengthPlatform,
        -0.5 * lengthRoom + heightPlatform + 1.0>
    scale scaleBlock
  }
  texture {
    texBlock
  }
}

// Whole scene
object {
  HouseOfStairs
}
```

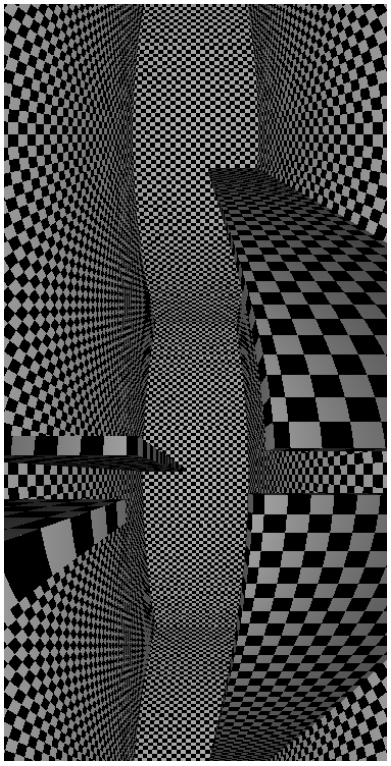


Now the rendered image matches the lithography enough for now and I want to confirm the symmetry hypothesis. Then I change the script to create the House by repeating, rotating and mirroring the platform and one fourth of the walls.

```
#include "colors.inc"

// Unit is one block size

// Width of the room (left-right axis of the lithography)
#declare widthRoom = 25.0;
// Length of the room (top-bottom axis of the lithography)
#declare lengthRoom = 102.0;

// Scale of the blocks
#declare scaleBlock = <1.0, 0.5, 0.5>;

// Texture of the blocks
#declare texBlock = texture {
  pigment { checker color rgb 0, color rgb 1 }
  scale scaleBlock
}

// Camera definition
#declare posCamera = <0.0, 0.0, 5.0>;
```

```
#declare lookAt = <0.0, 0.25 * lengthRoom, -0.5 * lengthRoom>;
camera {
  cylinder 2
  angle 200
  location posCamera
  look_at lookAt
  right x * 3.0
  up y
}

// Light source
light_source {
  posCamera
  color rgb 1.0
}

// Walls
#declare Walls = intersection {
  difference {
    box {
      -0.5, 0.5
      scale scaleBlock
      scale <widthRoom, lengthRoom, lengthRoom>
      scale 1.1
    }
    box {
      -0.5, 0.5
      scale scaleBlock
      scale <widthRoom, lengthRoom, lengthRoom>
    }
  }
  box {
    <-0.5, 0.0, 0.0>, 0.5
    scale scaleBlock
    scale <widthRoom, lengthRoom, lengthRoom>
  }
}

// Platform
#declare Platform = box {
  #declare widthPlatform = 10.0;
  #declare lengthPlatform = 50.0;
  #declare heightPlatform = 52.0;
  <-0.5 * widthRoom, 0.5 * lengthRoom, -0.5 * lengthRoom + heightPlatform>
  <-0.5 * widthRoom + widthPlatform, 0.5 * lengthRoom - lengthPlatform, -0.5
       * lengthRoom + heightPlatform + 1.0>
  scale scaleBlock
}

// House of stairs
#declare HouseOfStairs = union {
  #declare iQuarter = 0;
  #while (iQuarter < 4)
    union {
      object { Walls }
      object { Platform }
      rotate x * 90.0 * iQuarter
      #if (iQuarter = 1 | iQuarter = 3)
        scale <-1.0, 1.0, 1.0>
      #end
    }
    #declare iQuarter = iQuarter + 1;
```

```
  #end
  texture {
    texBlock
  }
}

// Whole scene
object {
  HouseOfStairs
}
```

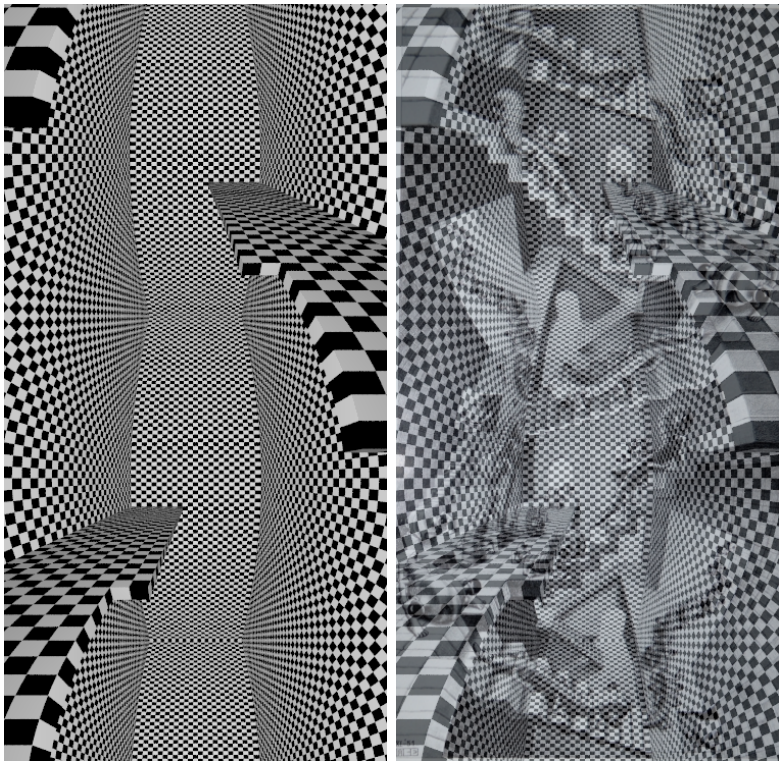


The repeated platforms doesn't match the ones in the lithography. My guess is that I've mistaken its dimensions (and consequently the position of the camera should also be wrong). I now try to correct this.

```
#include "colors.inc"

// Unit is one block size

// Width of the room (left-right axis of the lithography)
#declare widthRoom = 25.0;
// Length of the room (top-bottom axis of the lithography)
```

```
#declare lengthRoom = 102.0;

// Scale of the blocks
#declare scaleBlock = <1.0, 0.5, 0.5>;

// Texture of the blocks
#declare texBlock = texture {
  pigment { checker color rgb 0, color rgb 1 }
  translate 0.5*x
  scale scaleBlock
}

// Camera definition
#declare posCamera = <0.0, 0.0, 0.0>;
#declare lookAt = <0.0, 0.25 * lengthRoom, -0.5 * lengthRoom>;
camera {
  cylinder 2
  angle 210
  location posCamera
  look_at lookAt
  right x * 2.9
  up y
}

// Light source
light_source {
  posCamera
  color rgb 1.0
}

// Walls
#declare Walls = intersection {
  difference {
    box {
      -0.5, 0.5
      scale scaleBlock
      scale <widthRoom, lengthRoom, lengthRoom>
      scale 1.1
    }
    box {
      -0.5, 0.5
      scale scaleBlock
      scale <widthRoom, lengthRoom, lengthRoom>
    }
  }
  box {
    <-0.5, 0.0, 0.0>, 0.5
    scale scaleBlock
    scale <widthRoom, lengthRoom, lengthRoom>
  }
}

// Platform
#declare Platform = difference {
  #declare widthPlatform = 10.0;
  #declare lengthPlatform = 50.0;
  #declare heightPlatform = 43.0;
  box {
    <-0.5 * widthRoom, 0.5 * lengthRoom, -0.5 * lengthRoom + heightPlatform>
    <-0.5 * widthRoom + widthPlatform, 0.5 * lengthRoom - lengthPlatform,
        -0.5 * lengthRoom + heightPlatform + 1.0>
    scale scaleBlock
```

```
  }
  box {
    <-0.5 * widthRoom + widthPlatform - 2, 0.5 * lengthRoom - lengthPlatform
         + 11, -0.5 * lengthRoom + heightPlatform - 0.1>
    <-0.5 * widthRoom + widthPlatform + 1, 0.5 * lengthRoom - lengthPlatform
         - 1, -0.5 * lengthRoom + heightPlatform + 1.1>
    scale scaleBlock
  }
}

// House of stairs
#declare HouseOfStairs = union {
  #declare iQuarter = 0;
  #while (iQuarter < 4)
    union {
      object { Walls }
      object { Platform }
      rotate x * 90.0 * iQuarter
      #if (iQuarter = 1 | iQuarter = 3)
        scale <-1.0, 1.0, 1.0>
      #end
    }
    #declare iQuarter = iQuarter + 1;
  #end
  texture {
    texBlock
  }
}

// Whole scene
object {
  HouseOfStairs
}
```

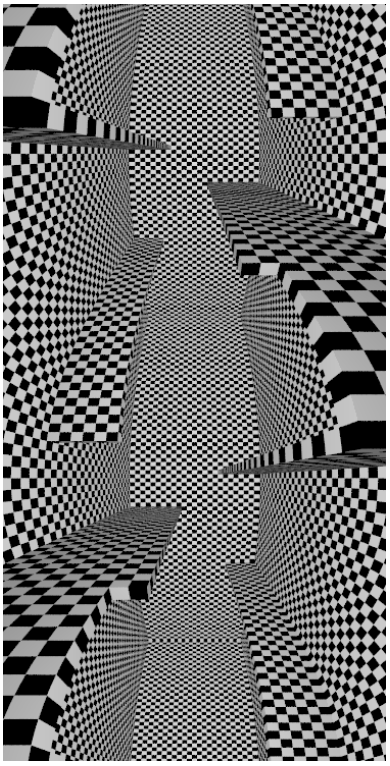


Getting closer and closer to the right dimensions for the House and parameters for the camera. Now I add the two other platforms, again using symetry. Their dimensions can be easily deducted from the first platform.

```
#include "colors.inc"

// Unit is one block size

// Width of the room (left-right axis of the lithography)
#declare widthRoom = 25.0;
// Length of the room (top-bottom axis of the lithography)
#declare lengthRoom = 102.0;

// Scale of the blocks
#declare scaleBlock = <1.0, 0.5, 0.5>;

// Texture of the blocks
#declare texBlock = texture {
  pigment { checker color rgb 0, color rgb 1 }
  translate 0.5*x
  scale scaleBlock
}

// Camera definition
#declare posCamera = <0.0, 0.0, 0.0>;
```

```
#declare lookAt = <0.0, 0.25 * lengthRoom, -0.5 * lengthRoom>;
camera {
  cylinder 2
  angle 210
  location posCamera
  look_at lookAt
  right x * 2.9
  up y
}

// Light source
light_source {
  posCamera
  color rgb 1.0
}

// Walls
#declare Walls = intersection {
  difference {
    box {
      -0.5, 0.5
      scale scaleBlock
      scale <widthRoom, lengthRoom, lengthRoom>
      scale 1.1
    }
    box {
      -0.5, 0.5
      scale scaleBlock
      scale <widthRoom, lengthRoom, lengthRoom>
    }
  }
  box {
    <-0.5, 0.0, 0.0>, 0.5
    scale scaleBlock
    scale <widthRoom, lengthRoom, lengthRoom>
  }
}

// Platforms
#declare PlatformA = difference {
  #declare widthPlatformA = 10.0;
  #declare lengthPlatformA = 50.0;
  #declare heightPlatformA = 43.0;
  box {
    <-0.5 * widthRoom, 0.5 * lengthRoom, -0.5 * lengthRoom + heightPlatformA
        >
    <-0.5 * widthRoom + widthPlatformA, 0.5 * lengthRoom - lengthPlatformA,
        -0.5 * lengthRoom + heightPlatformA + 1.0>
    scale scaleBlock
  }
  box {
    <-0.5 * widthRoom + widthPlatformA - 2, 0.5 * lengthRoom -
        lengthPlatformA + 11, -0.5 * lengthRoom + heightPlatformA - 0.1>
    <-0.5 * widthRoom + widthPlatformA + 1, 0.5 * lengthRoom -
        lengthPlatformA - 1, -0.5 * lengthRoom + heightPlatformA + 1.1>
    scale scaleBlock
  }
}

#declare PlatformB = box {
  #declare widthPlatformB = widthPlatformA - 2.0;
  #declare lengthPlatformB = heightPlatformA;
```

```
  #declare heightPlatformB = lengthPlatformA - 1.0;
  <-0.5 * widthRoom, -0.5 * lengthRoom + heightPlatformB, 0.5 * lengthRoom>
  <-0.5 * widthRoom + widthPlatformB, -0.5 * lengthRoom + heightPlatformB +
      1.0, 0.5 * lengthRoom - lengthPlatformB>
  scale scaleBlock
}

#declare PlatformC = box {
  #declare widthPlatformC = 6.0;
  #declare lengthPlatformC = lengthPlatformA - 3.0;
  #declare heightPlatformC = heightPlatformA - 15.0;
  <0.5 * widthRoom, 0.5 * lengthRoom, -0.5 * lengthRoom + heightPlatformC>
  <0.5 * widthRoom - widthPlatformC, 0.5 * lengthRoom - lengthPlatformC,
      -0.5 * lengthRoom + heightPlatformC + 1.0>
  scale scaleBlock
}

// House of stairs
#declare HouseOfStairs = union {
  #declare iQuarter = 0;
  #while (iQuarter < 4)
    union {
      object { Walls }
      object { PlatformA }
      object { PlatformB }
      object { PlatformC }
      rotate x * 90.0 * iQuarter
      #if (iQuarter = 1 | iQuarter = 3)
        scale <-1.0, 1.0, 1.0>
      #end
    }
    #declare iQuarter = iQuarter + 1;
  #end
  texture {
    texBlock
  }
}

// Whole scene
object {
  HouseOfStairs
}
```



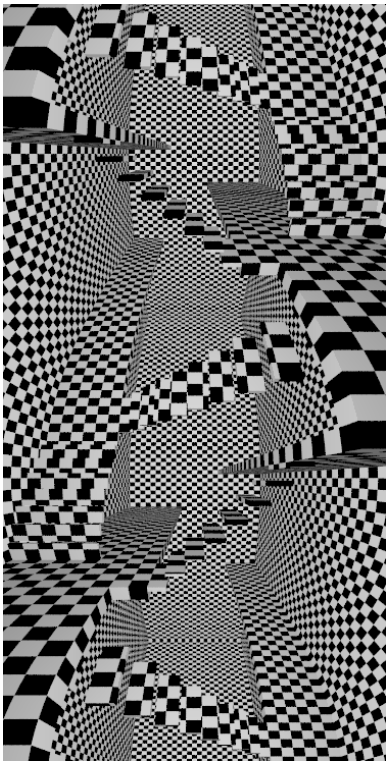
Next, I confirm and eventually correct the dimensions of the platforms by adding some simple models for the stairs connecting the platforms.

```
#include "colors.inc"

// Unit is one block size

// Width of the room (left-right axis of the lithography)
#declare widthRoom = 25.0;
// Length of the room (top-bottom axis of the lithography)
#declare lengthRoom = 102.0;

// Scale of the blocks
#declare scaleBlock = <1.0, 0.5, 0.5>;

// Texture of the blocks
#declare texBlock = texture {
  pigment { checker color rgb 0, color rgb 1 }
  translate 0.5*x
  scale scaleBlock
}

// Camera definition
#declare posCamera = <0.0, 0.0, 0.0>;
#declare lookAt = <0.0, 0.25 * lengthRoom, -0.5 * lengthRoom>;
camera {
```

```
  cylinder 2
  angle 210
  location posCamera
  look_at lookAt
  right x * 2.9
  up y
}

// Light source
light_source {
  posCamera
  color rgb 1.0
}

// Walls
#declare Walls = intersection {
  difference {
    box {
      -0.5, 0.5
      scale scaleBlock
      scale <widthRoom, lengthRoom, lengthRoom>
      scale 1.1
    }
    box {
      -0.5, 0.5
      scale scaleBlock
      scale <widthRoom, lengthRoom, lengthRoom>
    }
  }
  box {
    <-0.5, 0.0, 0.0>, <0.5, 0.5, -0.5>
    scale scaleBlock
    scale <widthRoom, lengthRoom, lengthRoom>
  }
}

// Platforms
#declare PlatformA = difference {
  #declare widthPlatformA = 10.0;
  #declare lengthPlatformA = 50.0;
  #declare heightPlatformA = 43.0;
  box {
    <
      -0.5 * widthRoom,
      0.5 * lengthRoom,
      -0.5 * lengthRoom + heightPlatformA
    >
    <
      -0.5 * widthRoom + widthPlatformA,
      0.5 * lengthRoom - lengthPlatformA,
      -0.5 * lengthRoom + heightPlatformA + 1.0
    >
    scale scaleBlock
  }
  box {
    <
      -0.5 * widthRoom + widthPlatformA - 2,
      0.5 * lengthRoom - lengthPlatformA + 11,
      -0.5 * lengthRoom + heightPlatformA - 0.1
    >
    <
      -0.5 * widthRoom + widthPlatformA + 1,
```

```
      0.5 * lengthRoom - lengthPlatformA - 1,
      -0.5 * lengthRoom + heightPlatformA + 1.1
    >
    scale scaleBlock
  }
}

#declare PlatformB = box {
  #declare widthPlatformB = widthPlatformA - 2.0;
  #declare lengthPlatformB = heightPlatformA;
  #declare heightPlatformB = lengthPlatformA - 1.0;
  <
    -0.5 * widthRoom,
    -0.5 * lengthRoom + heightPlatformB,
    0.5 * lengthRoom
  >
  <
    -0.5 * widthRoom + widthPlatformB,
    -0.5 * lengthRoom + heightPlatformB + 1.0,
    0.5 * lengthRoom - lengthPlatformB
  >
  scale scaleBlock
}

#declare PlatformC = box {
  #declare widthPlatformC = 6.0;
  #declare lengthPlatformC = lengthPlatformA - 4.0;
  #declare heightPlatformC = heightPlatformA - 15.0;
  <
    0.5 * widthRoom,
    0.5 * lengthRoom,
    -0.5 * lengthRoom + heightPlatformC
  >
  <
    0.5 * widthRoom - widthPlatformC,
    0.5 * lengthRoom - lengthPlatformC,
    -0.5 * lengthRoom + heightPlatformC + 1.0
  >
  scale scaleBlock
}

// Stairs
#macro MakeStairs(widthStair, nbStair, startPos, upVec, rightVec, frontVec)
  union {
    #declare oneStairBottom = -0.5 * widthStair * rightVec + 2.0 / 3.0 *
        vnormalize(upVec);
    #declare oneStairTop = 0.5 * widthStair * rightVec + vnormalize(upVec) +
         frontVec;
    #declare iStair = 0;
    #while (iStair < nbStair)
      box {
        oneStairBottom
        oneStairTop
        scale scaleBlock
        translate (frontVec + upVec) * iStair * scaleBlock
      }
      #declare iStair = iStair + 1;
    #end
    translate startPos * scaleBlock
  }
#end
```

```
#declare widthStairsA = 4;
#declare nbStairsA = 10;
#declare slopeUpStairsA = (heightPlatformA - heightPlatformC - 1) / (
    nbStairsA - 1);
#declare slopeFrontStairsA = (widthRoom - widthPlatformC - (widthPlatformA -
     2.0)) / (nbStairsA - 1);
#declare posStairsA = <0.5 * widthRoom - widthPlatformC, 0.5 * lengthRoom -
    lengthPlatformC + 2.0, -0.5 * lengthRoom + heightPlatformC>;
#declare StairsA = MakeStairs(widthStairsA, nbStairsA, posStairsA, z *
    slopeUpStairsA, -y, -x * slopeFrontStairsA);

#declare widthStairsB = 4;
#declare nbStairsB = 8;
#declare slopeUpStairsB = 2.0 * (heightPlatformB - heightPlatformA - 2) / (
    nbStairsB - 1);
#declare slopeFrontStairsB = (widthRoom - widthPlatformB - widthPlatformA) /
     (nbStairsB - 1);
#declare posStairsB = <0.5 * widthRoom - widthPlatformA, -0.5 * lengthRoom +
     heightPlatformA, 0.5 * lengthRoom - lengthPlatformA - 15.0>;
#declare StairsB = MakeStairs(widthStairsB, nbStairsB, posStairsB, y *
    slopeUpStairsB, z, -x * slopeFrontStairsB);

#declare widthStairsC = 6;
#declare nbStairsC = 9;
#declare slopeUpStairsC = (lengthPlatformA - 16.0 - heightPlatformC) / (
    nbStairsC - 1);
#declare slopeFrontStairsC = (lengthRoom - (lengthPlatformC +
    heightPlatformA + 1.0)) / (nbStairsC - 1);
#declare posStairsC = <0.5 * widthRoom - 3.0,
    0.5 * lengthRoom - lengthPlatformC,
    -0.5 * lengthRoom + heightPlatformC>;
#declare StairsC = MakeStairs(widthStairsC, nbStairsC, posStairsC, z *
    slopeUpStairsC, x, -y * slopeFrontStairsC);

// House of stairs
#declare HouseOfStairs = union {
  #declare iQuarter = 0;
  #while (iQuarter < 4)
    union {
      object { Walls }
      object { PlatformA }
      object { PlatformB }
      object { PlatformC }
      object { StairsA }
      object { StairsB }
      object { StairsC }
      rotate x * 90.0 * iQuarter
      #if (iQuarter = 1 | iQuarter = 3)
        scale <-1.0, 1.0, 1.0>
      #end
    }
    #declare iQuarter = iQuarter + 1;
  #end
  texture {
    texBlock
  }
}

// Whole scene
object {
  HouseOfStairs
}
```



The stairs and platform fit very well altogether, but there is some deviation between the image and the lithography. I believe it will be solved by tuning again the camera parameters.

I think it's enough for the prototype. The remaining stairs, doors and external walls should not cause problems given what matches so well in the prototype. So I move on to the real model.

4 House model

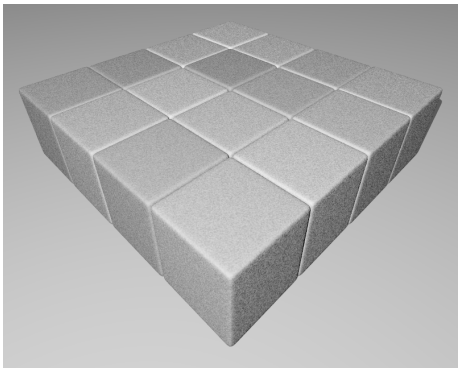
4.1 Block model

As the basic element of the house is the block I start by creating its model and a macro to generate walls made of these blocks.

```
#include "colors.inc"
```

```
// Unit is one block size

background { color rgb 1.0 }
plane {y, 0 pigment {color rgb 1.0}}

// Camera definition
#declare posCamera = <3.0, 3.0, 3.0>;
#declare lookAt = <0.5, 0.0, 0.5>;
camera {
  location posCamera
  look_at lookAt
}

// Light source
light_source {
  posCamera + x + y
  color rgb 1.0
}

#macro RandScaleBlock(c)
  (1.0 + (0.5 - rand(seedBlock)) * c)
#end

// Block
#declare seedBlock = seed(0);
#macro Block()
  superellipsoid {
    #declare BlockRoundness = 0.075;
    <BlockRoundness, BlockRoundness>
    texture {
      pigment {color rgb 0.75 * RandScaleBlock(0.5)}
      normal { bumps 0.25 scale .02 }
      finish { ambient 0.0 diffuse 1.0 }
    }
    scale 0.5 * RandScaleBlock(0.025)
  }
#end

#macro MakeWall(PosMin, PosMax)
  union {
    #declare wallX = PosMin.x;
    #while (wallX < PosMax.x)
      #declare wallY = PosMin.y;
      #while (wallY < PosMax.y)
        #declare wallZ = PosMin.z;
        #while (wallZ < PosMax.z)
          object {
            Block()
            translate <wallX + 0.5, wallY + 0.5, wallZ + 0.5>
          }
          #declare wallZ = wallZ + 1.0;
        #end
        #declare wallY = wallY + 1.0;
      #end
      #declare wallX = wallX + 1.0;
    #end
  }
#end

#declare Wall = MakeWall(<-2.0, 0.0, -2.0>, <2.0, 1.0, 2.0>);

object {
```

```
  Wall
}
```



4.2 Cleaning the prototype script

In preparation for the refactoring of the prototype of the House of Stairs I clean its script.

blocks.inc:

```
// Seed for the randomization of blocks' texture and size

#declare seedBlock = seed(0);

// Return a random value in range [1.0 - r / 2, 1.0 + r / 2]

#macro RandAroungOne(r)
  (1.0 + (0.5 - rand(seedBlock)) * r)
#end

// Return one block

#macro Block()
  #local blockRoundness = 0.075;
  superellipsoid {
    <blockRoundness, blockRoundness>
    texture {
      pigment {color rgb 0.75 * RandAroungOne(0.5)}
      normal { bumps 0.25 scale .02 }
      finish { ambient 0.0 diffuse 1.0 }
    }
    scale 0.5 * RandAroungOne(0.025)
  }
#end

// Make the wall equivalent to box {posMin, posMax} made of blocks
```

```
#macro MakeWall(
    posMin,
    posMax)
  union {
    #local wallX = posMin.x;
    #while (wallX < posMax.x)
      #local wallY = posMin.y;
      #while (wallY < posMax.y)
        #local wallZ = posMin.z;
        #while (wallZ < posMax.z)
          object {
            Block()
            translate <wallX + 0.5, wallY + 0.5, wallZ + 0.5>
          }
          #declare wallZ = wallZ + 1.0;
        #end
        #declare wallY = wallY + 1.0;
      #end
      #declare wallX = wallX + 1.0;
    #end
  }
#end

// Return one stair
#macro MakeStairs(
    widthStair,
    nbStair,
    startPos,
    upVec,
    rightVec,
    frontVec)
  union {
    #local oneStairBottom =
      -0.5 * widthStair * rightVec + 2.0 / 3.0 * vnormalize(upVec);
    #local oneStairTop =
      0.5 * widthStair * rightVec + vnormalize(upVec) + frontVec;
    #local iStair = 0;
    #while (iStair < nbStair)
      box {
        oneStairBottom
        oneStairTop
        translate (frontVec + upVec) * iStair
      }
      #declare iStair = iStair + 1;
    #end
    translate startPos
  }
#end
```

XL-51.pov:

```
// ------ Includes ------

// Include the macros relative to blocks

#include "blocks.inc"

// ------ Scene elements' dimensions ------

// Unit is one block size
```

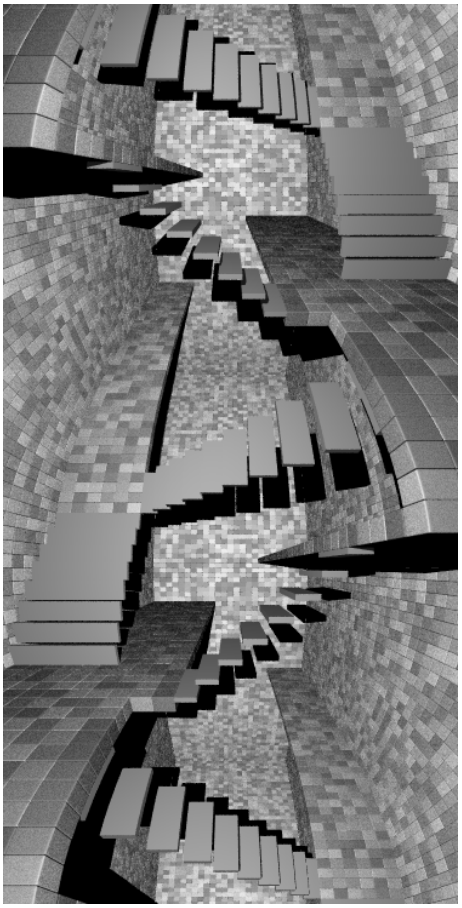
```
// Scale of one block

#declare scaleBlock = <1.0, 0.5, 0.5>;

// Width of the room (left-right axis of the lithography)

#declare widthRoom = 25.0;

// Length of the room (top-bottom axis of the lithography)

#declare lengthRoom = 102.0;

// Platforms' dimensions

#declare widthPlatformA  = 10.0;
#declare lengthPlatformA = 50.0;
#declare heightPlatformA = 43.0;

#declare widthPlatformB  = widthPlatformA - 2.0;
#declare lengthPlatformB = heightPlatformA;
#declare heightPlatformB = lengthPlatformA - 1.0;

#declare widthPlatformC  = 6.0;
#declare lengthPlatformC = lengthPlatformA - 4.0;
#declare heightPlatformC = heightPlatformA - 15.0;

// Stairs' dimensions

#declare widthStairsA = 4;
#declare nbStairsA = 10;
#declare slopeUpStairsA =
  (heightPlatformA - heightPlatformC - 1) / (nbStairsA - 1);
#declare slopeFrontStairsA =
  (widthRoom - widthPlatformC - (widthPlatformA - 2.0)) /
  (nbStairsA - 1);
#declare posStairsA =
  <
     0.5 * widthRoom - widthPlatformC,
     0.5 * lengthRoom - lengthPlatformC + 2.0,
    -0.5 * lengthRoom + heightPlatformC
  >;

#declare widthStairsB = 4;
#declare nbStairsB = 8;
#declare slopeUpStairsB =
  2.0 * (heightPlatformB - heightPlatformA - 2) / (nbStairsB - 1);
#declare slopeFrontStairsB =
  (widthRoom - widthPlatformB - widthPlatformA) / (nbStairsB - 1);
#declare posStairsB =
  <
     0.5 * widthRoom - widthPlatformA,
    -0.5 * lengthRoom + heightPlatformA,
     0.5 * lengthRoom - lengthPlatformA - 15.0
  >;

#declare widthStairsC = 6;
#declare nbStairsC = 9;
#declare slopeUpStairsC =
  (lengthPlatformA - 16.0 - heightPlatformC) / (nbStairsC - 1);
#declare slopeFrontStairsC =
  (lengthRoom - (lengthPlatformC + heightPlatformA + 1.0)) /
  (nbStairsC - 1);
```

```
#declare posStairsC =
  <
     0.5 * widthRoom - 3.0,
     0.5 * lengthRoom - lengthPlatformC,
    -0.5 * lengthRoom + heightPlatformC
  >;

// ------- Side walls definition -------

#declare SideWallA = union {
  MakeWall(
    <
      -0.5 * widthRoom - 1.0,
       0.0,
       0.0
    >,
    <
      -0.5 * widthRoom,
       0.5 * lengthRoom,
       0.5 * lengthRoom
    >)
  scale scaleBlock
}

#declare SideWallB = union {
  MakeWall(
    <
      0.5 * widthRoom,
      0.0,
      0.0
    >,
    <
      0.5 * widthRoom + 1.0,
      0.5 * lengthRoom,
      0.5 * lengthRoom
    >)
  scale scaleBlock
}

#declare SideWallC = union {
  MakeWall(
    <
      -0.5 * widthRoom,
       0.0,
       0.5 * lengthRoom
    >,
    <
      0.5 * widthRoom,
      0.5 * lengthRoom,
      0.5 * lengthRoom + 1.0
    >)
  scale scaleBlock
}

#declare SideWallD = union {
  MakeWall(
    <
      -0.5 * widthRoom,
       0.5 * lengthRoom,
       0.0
    >,
    <
```

```
      0.5 * widthRoom,
      0.5 * lengthRoom + 1.0,
      0.5 * lengthRoom
    >)
  scale scaleBlock
}

#declare SideWalls = union {
  object { SideWallA }
  object { SideWallB }
  object { SideWallC }
  object { SideWallD }
}

// ------- Platforms definition -------

#declare PlatformA = union {
  MakeWall(
    <
      -0.5 * widthRoom,
       0.5 * lengthRoom - lengthPlatformA + 11,
      -0.5 * lengthRoom + heightPlatformA
    >,
    <
      -0.5 * widthRoom + widthPlatformA,
       0.5 * lengthRoom,
      -0.5 * lengthRoom + heightPlatformA + 1.0
    >)
  MakeWall(
    <
      -0.5 * widthRoom,
       0.5 * lengthRoom - lengthPlatformA,
      -0.5 * lengthRoom + heightPlatformA
    >,
    <
      -0.5 * widthRoom + widthPlatformA - 2,
       0.5 * lengthRoom - lengthPlatformA + 11,
      -0.5 * lengthRoom + heightPlatformA + 1.0
    >)
  scale scaleBlock
}

#declare PlatformB = union {
  MakeWall(
    <
      -0.5 * widthRoom,
      -0.5 * lengthRoom + heightPlatformB,
       0.5 * lengthRoom - lengthPlatformB
    >,
    <
      -0.5 * widthRoom + widthPlatformB,
      -0.5 * lengthRoom + heightPlatformB + 1.0,
       0.5 * lengthRoom
    >)
  scale scaleBlock
}

#declare PlatformC = union {
  MakeWall(
    <
       0.5 * widthRoom - widthPlatformC,
       0.5 * lengthRoom - lengthPlatformC,
```

```
      -0.5 * lengthRoom + heightPlatformC
    >,
    <
       0.5 * widthRoom,
       0.5 * lengthRoom,
      -0.5 * lengthRoom + heightPlatformC + 1.0
    >)
  scale scaleBlock
}

#declare Platforms = union {
  object { PlatformA }
  object { PlatformB }
  object { PlatformC }
}

// ------ Stairs definition ------

#declare StairsA = object {
  MakeStairs(
     widthStairsA,
     nbStairsA,
     posStairsA,
     z * slopeUpStairsA,
    -y,
    -x * slopeFrontStairsA)
  scale scaleBlock
}

#declare StairsB = object {
  MakeStairs(
     widthStairsB,
     nbStairsB,
     posStairsB,
     y * slopeUpStairsB,
     z,
    -x * slopeFrontStairsB)
  scale scaleBlock
}

#declare StairsC = object {
  MakeStairs(
     widthStairsC,
     nbStairsC,
     posStairsC,
     z * slopeUpStairsC,
     x,
    -y * slopeFrontStairsC)
  scale scaleBlock
}

#declare Stairs = union {
  object { StairsA }
  object { StairsB }
  object { StairsC }
  texture {pigment {color rgb 1.0}}
}

// ------ House of Stairs definition ------

#declare HouseOfStairs = union {
```

```
  // Loop on the four quarters of the house
  #declare iQuarter = 0;
  #while (iQuarter < 4)
    union {

      // Elements of one quarter
      object { SideWalls }
      object { Platforms }
      object { Stairs }

      // Rotate the quarter along the horizontal axis
      rotate x * 90.0 * iQuarter

      // Apply the symmetry to the odd quarters
      #if (iQuarter = 1 | iQuarter = 3)
        scale <-1.0, 1.0, 1.0>
      #end
    }
    #declare iQuarter = iQuarter + 1;
  #end
}

// ------ Camera ------

#declare posCamera = <0.0, 0.0, 0.0>;
#declare lookAt = <0.0, 0.25 * lengthRoom, -0.5 * lengthRoom>;

camera {
  cylinder 2
  angle 210
  location posCamera
  look_at lookAt
  right x * 2.9
  up y
}

// ------ Light ------

light_source {
  posCamera + 1.0
  color rgb 1.0
}

// ------ Scene ------

object {
  HouseOfStairs
}
```



4.3 Stair model

I finish the model for the stairs.

blocks.inc:

```
// Seed for the randomization of blocks' texture and size
#declare seedBlock = seed(0);
// Roundness coefficient of the blocks
#declare blockRoundness = 0.075;
// Randomness coefficient for the block dimension
#declare blockRandomness = 0.025;
// Thickness coefficient of the stairs
```

```
#declare stairThickness = 1.0 / 2.0;

// Return a random value in range [1.0 - r / 2, 1.0 + r / 2]

#macro RandAroungOne(r)
  (1.0 + (0.5 - rand(seedBlock)) * r)
#end

// Return the texture of the blocks

#macro BlockTex()
  texture {
    pigment {
      color rgb (1.0 - blockRandomness) * RandAroungOne(0.25)
    }
    normal {
      bumps 0.25 scale .02
    }
    finish {
      ambient 0.0 diffuse 1.0
    }
  }
#end

// Return the scaling for the randomization of the size of the blocks

#macro BlockRandScale()
  0.5 * RandAroungOne(blockRandomness)
#end

// Return one block

#macro Block()
  superellipsoid {
    <blockRoundness, blockRoundness>
    BlockTex()
    scale BlockRandScale()
  }
#end

// Make the wall equivalent to box {posMin, posMax} made of blocks

#macro MakeWall(
    posMin,
    posMax)
  union {
    #local wallX = posMin.x;
    #while (wallX < posMax.x)
      #local wallY = posMin.y;
      #while (wallY < posMax.y)
        #local wallZ = posMin.z;
        #while (wallZ < posMax.z)
          object {
            Block()
            translate <wallX + 0.5, wallY + 0.5, wallZ + 0.5>
          }
          #declare wallZ = wallZ + 1.0;
        #end
        #declare wallY = wallY + 1.0;
      #end
      #declare wallX = wallX + 1.0;
```

```
    #end
  }
#end

// Return one stair

#macro MakeStairs(
    widthStair,
    nbStair,
    startPos,
    upVec,
    rightVec,
    frontVec)
  union {
    #local oneStairBottom =
      -0.5 * widthStair * rightVec + (1.0 - stairThickness) * vnormalize(
          upVec);
    #local oneStairTop =
      0.5 * widthStair * rightVec + vnormalize(upVec) + frontVec;
    #local iStair = 0;
    #while (iStair < nbStair)
      #local jStair = 0;
      #while (jStair < widthStair)
        superellipsoid {
          <blockRoundness, blockRoundness>
          scale 0.5
          translate 0.5
          scale vnormalize(rightVec) + upVec * stairThickness + frontVec *
              (1.0 + blockRoundness)
          translate upVec * (1.0 - stairThickness)
          translate rightVec * (jStair - 0.5 * widthStair)
          translate (frontVec + upVec) * iStair
        }
        #if (iStair > 0)
          superellipsoid {
            <blockRoundness, blockRoundness>
            scale 0.5
            translate 0.5
            scale vnormalize(rightVec) + upVec * (1.0 + stairThickness) +
                frontVec * stairThickness
            translate -1.0 * upVec * stairThickness
            translate rightVec * (jStair - 0.5 * widthStair)
            translate (frontVec + upVec) * iStair
          }
        #end
        #declare jStair = jStair + 1;
      #end
      //box {
      //  oneStairBottom
      //  oneStairTop
      //  translate (frontVec + upVec) * iStair
      //}
      #declare iStair = iStair + 1;
    #end
    translate startPos
    BlockTex()
  }
#end
```

XL-51.pov:

```
// ------ Includes ------

// Include the macros relative to blocks

#include "blocks.inc"

// ------ Scene elements' dimensions ------

// Unit is one block size

// Scale of one block

#declare scaleBlock = <1.0, 0.5, 0.5>;

// Width of the room (left-right axis of the lithography)

#declare widthRoom = 25.0;

// Length of the room (top-bottom axis of the lithography)

#declare lengthRoom = 102.0;

// Platforms' dimensions

#declare widthPlatformA  = 10.0;
#declare lengthPlatformA = 50.0;
#declare heightPlatformA = 43.0;

#declare widthPlatformB  = widthPlatformA - 2.0;
#declare lengthPlatformB = heightPlatformA;
#declare heightPlatformB = lengthPlatformA - 1.0;

#declare widthPlatformC  = 6.0;
#declare lengthPlatformC = lengthPlatformA - 4.0;
#declare heightPlatformC = heightPlatformA - 15.0;

// Stairs' dimensions

#declare widthStairsA = 4;
#declare nbStairsA = 10;
#declare slopeUpStairsA =
  (heightPlatformA - heightPlatformC - 1) / (nbStairsA - 1);
#declare slopeFrontStairsA =
  (widthRoom - widthPlatformC - (widthPlatformA - 2.0)) /
  (nbStairsA - 1);
#declare posStairsA =
  <
     0.5 * widthRoom - widthPlatformC,
     0.5 * lengthRoom - lengthPlatformC + 2.0,
    -0.5 * lengthRoom + heightPlatformC - 0.5
  >;

#declare widthStairsB = 4;
#declare nbStairsB = 8;
#declare slopeUpStairsB =
  2.0 * (heightPlatformB - heightPlatformA - 2) / (nbStairsB - 1);
#declare slopeFrontStairsB =
  (widthRoom - widthPlatformB - widthPlatformA) / (nbStairsB - 1);
#declare posStairsB =
  <
     0.5 * widthRoom - widthPlatformA,
    -0.5 * lengthRoom + heightPlatformA,
```

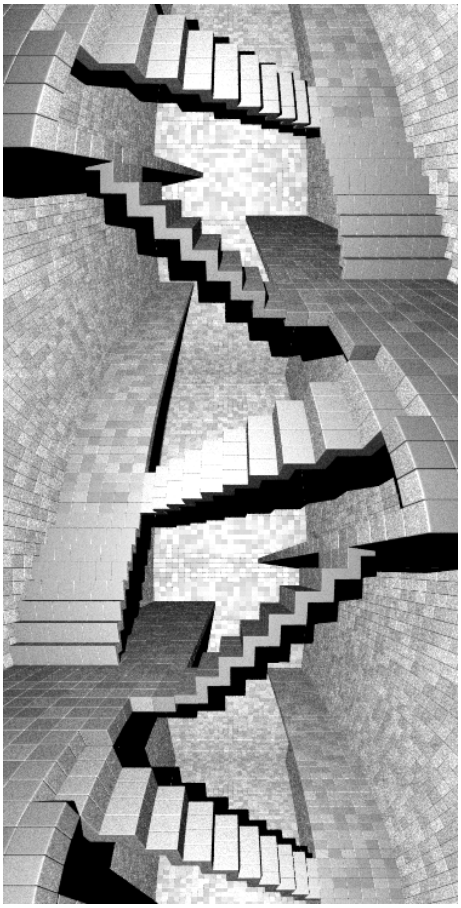
```
     0.5 * lengthRoom - lengthPlatformA - 15.0
  >;

#declare widthStairsC = 6;
#declare nbStairsC = 9;
#declare slopeUpStairsC =
  (lengthPlatformA - 16.0 - heightPlatformC) / (nbStairsC - 1);
#declare slopeFrontStairsC =
  (lengthRoom - (lengthPlatformC + heightPlatformA + 1.0)) /
  (nbStairsC - 1);
#declare posStairsC =
  <
     0.5 * widthRoom - 3.0,
     0.5 * lengthRoom - lengthPlatformC,
    -0.5 * lengthRoom + heightPlatformC + 0.25
  >;

// ------- Side walls definition -------

#declare SideWallA = union {
  MakeWall(
    <
      -0.5 * widthRoom - 1.0,
       0.0,
       0.0
    >,
    <
      -0.5 * widthRoom,
       0.5 * lengthRoom,
       0.5 * lengthRoom
    >)
  scale scaleBlock
}

#declare SideWallB = union {
  MakeWall(
    <
      0.5 * widthRoom,
      0.0,
      0.0
    >,
    <
      0.5 * widthRoom + 1.0,
      0.5 * lengthRoom,
      0.5 * lengthRoom
    >)
  scale scaleBlock
}

#declare SideWallC = union {
  MakeWall(
    <
      -0.5 * widthRoom,
       0.0,
       0.5 * lengthRoom
    >,
    <
      0.5 * widthRoom,
      0.5 * lengthRoom,
      0.5 * lengthRoom + 1.0
    >)
  scale scaleBlock
```

```
}

#declare SideWallD = union {
  MakeWall(
    <
      -0.5 * widthRoom,
       0.5 * lengthRoom,
       0.0
    >,
    <
      0.5 * widthRoom,
      0.5 * lengthRoom + 1.0,
      0.5 * lengthRoom
    >)
  scale scaleBlock
}

#declare SideWalls = union {
  object { SideWallA }
  object { SideWallB }
  object { SideWallC }
  object { SideWallD }
}

// ------- Platforms definition -------

#declare PlatformA = union {
  MakeWall(
    <
      -0.5 * widthRoom,
       0.5 * lengthRoom - lengthPlatformA + 11,
      -0.5 * lengthRoom + heightPlatformA
    >,
    <
      -0.5 * widthRoom + widthPlatformA,
       0.5 * lengthRoom,
      -0.5 * lengthRoom + heightPlatformA + 1.0
    >)
  MakeWall(
    <
      -0.5 * widthRoom,
       0.5 * lengthRoom - lengthPlatformA,
      -0.5 * lengthRoom + heightPlatformA
    >,
    <
      -0.5 * widthRoom + widthPlatformA - 3,
       0.5 * lengthRoom - lengthPlatformA + 11,
      -0.5 * lengthRoom + heightPlatformA + 1.0
    >)


  MakeWall(
    <
      -0.5 * widthRoom + widthPlatformA - 3,
       0.5 * lengthRoom - lengthPlatformA,
      -0.5 * lengthRoom + heightPlatformA
    >,
    <
      -0.5 * widthRoom + widthPlatformA - 2,
       0.5 * lengthRoom - lengthPlatformA + 4,
      -0.5 * lengthRoom + heightPlatformA + 1.0
    >)
```

```
  MakeWall(
    <
      -0.5 * widthRoom + widthPlatformA - 3,
       0.5 * lengthRoom - lengthPlatformA + 8,
      -0.5 * lengthRoom + heightPlatformA
    >,
    <
      -0.5 * widthRoom + widthPlatformA - 2,
       0.5 * lengthRoom - lengthPlatformA + 11,
      -0.5 * lengthRoom + heightPlatformA + 1.0
    >)


  scale scaleBlock
}

#declare PlatformB = union {
  MakeWall(
    <
      -0.5 * widthRoom,
      -0.5 * lengthRoom + heightPlatformB,
       0.5 * lengthRoom - lengthPlatformB
    >,
    <
      -0.5 * widthRoom + widthPlatformB,
      -0.5 * lengthRoom + heightPlatformB + 1.0,
       0.5 * lengthRoom
    >)
  scale scaleBlock
}

#declare PlatformC = union {
  MakeWall(
    <
       0.5 * widthRoom - widthPlatformC,
       0.5 * lengthRoom - lengthPlatformC,
      -0.5 * lengthRoom + heightPlatformC
    >,
    <
       0.5 * widthRoom,
       0.5 * lengthRoom,
      -0.5 * lengthRoom + heightPlatformC + 1.0
    >)
  scale scaleBlock
}

#declare Platforms = union {
  object { PlatformA }
  object { PlatformB }
  object { PlatformC }
}

// ------ Stairs definition ------

#declare StairsA = object {
  MakeStairs(
     widthStairsA,
     nbStairsA,
     posStairsA,
     z * slopeUpStairsA,
    -y,
    -x * slopeFrontStairsA)
```

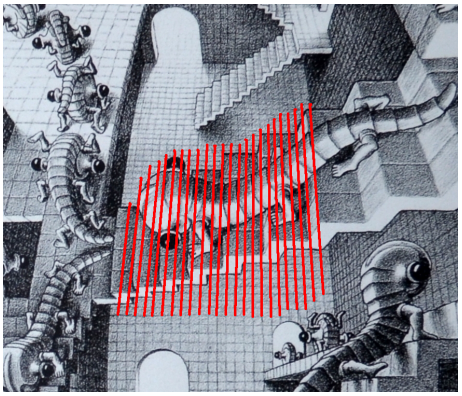
```
  scale scaleBlock
}

#declare StairsB = object {
  MakeStairs(
     widthStairsB,
     nbStairsB,
     posStairsB,
     y * slopeUpStairsB,
     z,
    -x * slopeFrontStairsB)
  scale scaleBlock
}

#declare StairsC = object {
  MakeStairs(
     widthStairsC,
     nbStairsC,
     posStairsC,
     z * slopeUpStairsC,
     x,
    -y * slopeFrontStairsC)
  scale scaleBlock
}

#declare Stairs = union {
  object { StairsA }
  object { StairsB }
  object { StairsC }
  texture {pigment {color rgb 1.0}}
}

// ------ House of Stairs definition ------

#declare HouseOfStairs = union {

  // Loop on the four quarters of the house
  #declare iQuarter = 0;
  #while (iQuarter < 4)
    union {

      // Elements of one quarter
      object { SideWalls }
      object { Platforms }
      object { Stairs }

      // Rotate the quarter along the horizontal axis
      rotate x * 90.0 * iQuarter

      // Apply the symmetry to the odd quarters
      #if (iQuarter = 1 | iQuarter = 3)
        scale <-1.0, 1.0, 1.0>
      #end
    }
    #declare iQuarter = iQuarter + 1;
  #end
}

// ------ Camera ------

#declare posCamera = <0.0, 0.0, 0.0>;
#declare lookAt = <0.0, 0.25 * lengthRoom, -0.5 * lengthRoom>;
```

```
camera {
  cylinder 2
  angle 210
  location posCamera
  look_at lookAt
  right x * 2.9
  up y
}

// ------ Light ------

light_source {
  posCamera + 1.0
  color rgb 1.0
}

// ------ Scene ------

object {
  HouseOfStairs
}
```

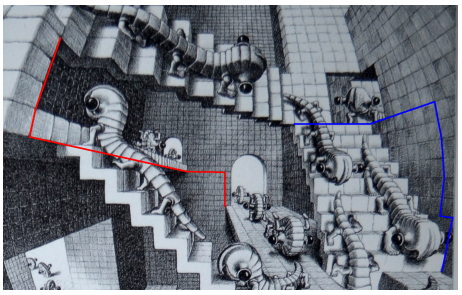


It looks nice but looking closely at the stairs make me realize it doesn't match the lithography. There is a problem in dimensions or position of the platforms. I try to fix that, refactor completely the dimensions of my model, and doing so start to find inconsistencies in the drawing.

First, the horizontal dimension of the front wall doens't match in its upper part and center part (24 blocks against 25 blocks).



Second, the red and blue lines below should have same dimension along the depth axis but they haven't (10 blocks against 9 blocks).



I'm really embarrassed. If the drawing is mathematically incorrect I have no way to reproduced it exactly in Pov-Ray... I have no choice but to find a compromise. I rework once again the dimensions of the model, and add the last latform and stairs, which reveals even more inconsistencies.

```
// ------ Includes ------

// Include the macros relative to blocks

#include "blocks.inc"

// ------ Scene elements' dimensions ------

// Unit is one block size

// Scale of one block

#declare scaleBlock = <1.0, 0.5, 0.5>;

// Measured platforms' and stairs' dimensions

#declare widthPlatformA = 10;

#declare lengthPlatformC = 46;
#declare widthPlatformC = 6;

#declare widthPlatformD = 4;

#declare widthGapPlatformAPlatformB = 7;
#declare heightGapPlatformAPlatformB = 6;
#declare heightGapPlatformBPlatformC = 24;

#declare widthStairsA = 4;
#declare nbStairsA = 10;

#declare widthStairsB = 4;
#declare nbStairsB = 8;

#declare nbStairsC = 12;
#declare widthStairsC = 6;
#declare slopeUpStairsC = 1.0;
#declare slopeFrontStairsC = 1.0;

#declare nbStairsD = 13;
#declare widthStairsD = 6;
#declare slopeUpStairsD = 1.0;
#declare slopeFrontStairsD = 1.0;

#declare nbStairsE = 15;
#declare widthStairsE = 6;
#declare slopeUpStairsE = 1.0;
#declare slopeFrontStairsE = 1.0;

// Deducted platforms' and stairs' dimensions

#declare heightPlatformC = nbStairsD + nbStairsE - 1;

#declare heightPlatformD = nbStairsD - 1;
#declare lengthPlatformD = widthStairsD + widthStairsE;
```

```
#declare widthRoom = // (right->left axis of the front wall)
  widthPlatformC + widthPlatformD + nbStairsE;

#declare lengthPlatformA = lengthPlatformC + 4;

#declare widthPlatformB = widthPlatformA - 2;
#declare heightPlatformB =
  heightPlatformC + heightGapPlatformBPlatformC;

#declare heightPlatformA =
  heightPlatformB - heightGapPlatformAPlatformB;

#declare lengthPlatformB = heightPlatformA;

#declare lengthRoom = // (bottom->top axis of the front wall)
  heightPlatformB + lengthPlatformA;

#declare posStairsA =
  <
     0.5 * widthRoom - widthPlatformC,
     0.5 * lengthRoom - (lengthPlatformC - widthStairsA / 2),
    -0.5 * lengthRoom + heightPlatformC
  >;
#declare slopeFrontStairsA =
  (widthRoom - widthPlatformC - widthPlatformB + 1) / nbStairsA;
#declare slopeUpStairsA =
  (heightPlatformA - heightPlatformC) / nbStairsA;

#declare posStairsB =
  <
    -0.5 * widthRoom + widthPlatformA,
     0.5 * lengthRoom - lengthPlatformA + 11 + widthStairsB / 2,
    -0.5 * lengthRoom + heightPlatformA
  >;
#declare slopeFrontStairsB =
  (widthRoom - widthPlatformB - widthPlatformA) / nbStairsB;
#declare slopeUpStairsB =
  (heightGapPlatformAPlatformB) / nbStairsB;

#declare posStairsC =
  <
     0.5 * widthRoom - widthStairsC / 2,
     0.5 * lengthRoom - lengthPlatformC,
    -0.5 * lengthRoom + heightPlatformC
  >;

#declare posStairsD =
  <
     -0.5 * widthRoom + widthPlatformD + nbStairsD,
     0.5 * lengthRoom - lengthPlatformD + widthStairsD / 2,
    -0.5 * lengthRoom - 1
      // -1 because the first stair is hidden in the wall
  >;

#declare posStairsE =
  <
     -0.5 * widthRoom + widthPlatformD,
     0.5 * lengthRoom - widthStairsE / 2,
    -0.5 * lengthRoom + heightPlatformD
  >;
```

```
// ------- Side walls definition -------

#declare lengthRoomSideWall = lengthRoom + 1;

#declare SideWallA = union {
  MakeWall(
    <
      -0.5 * widthRoom - 1.0,
       0.0,
       -0.5 * lengthRoomSideWall
    >,
    <
      -0.5 * widthRoom,
       0.5 * lengthRoomSideWall,
       0.0
    >)
  scale scaleBlock
}

#declare SideWallB = union {
  MakeWall(
    <
      0.5 * widthRoom,
      0.0,
      -0.5 * lengthRoomSideWall
    >,
    <
      0.5 * widthRoom + 1.0,
      0.5 * lengthRoomSideWall,
      0.0
    >)
  scale scaleBlock
}

#declare SideWallC = union {
  MakeWall(
    <
      -0.5 * widthRoom,
       0.0,
       -0.5 * lengthRoomSideWall - 1.0
    >,
    <
      0.5 * widthRoom,
      0.5 * lengthRoomSideWall,
      -0.5 * lengthRoomSideWall
    >)
  scale scaleBlock
}

#declare SideWallD = union {
  MakeWall(
    <
      -0.5 * widthRoom,
       0.5 * lengthRoomSideWall,
      -0.5 * lengthRoomSideWall
    >,
    <
      0.5 * widthRoom,
      0.5 * lengthRoomSideWall + 1.0,
       0.0
    >)
  scale scaleBlock
```

```
}

#declare SideWalls = union {
  object { SideWallA }
  object { SideWallB }
  object { SideWallC }
  object { SideWallD }
}

// ------- Platforms definition -------

#declare PlatformA = union {
  MakeWall(
    <
      -0.5 * widthRoom,
       0.5 * lengthRoom - lengthPlatformA + 11,
      -0.5 * lengthRoom + heightPlatformA
    >,
    <
      -0.5 * widthRoom + widthPlatformA,
       0.5 * lengthRoom,
      -0.5 * lengthRoom + heightPlatformA + 1.0
    >)
  MakeWall(
    <
      -0.5 * widthRoom,
       0.5 * lengthRoom - lengthPlatformA,
      -0.5 * lengthRoom + heightPlatformA
    >,
    <
      -0.5 * widthRoom + widthPlatformA - 3,
       0.5 * lengthRoom - lengthPlatformA + 11,
      -0.5 * lengthRoom + heightPlatformA + 1.0
    >)


  MakeWall(
    <
      -0.5 * widthRoom + widthPlatformA - 3,
       0.5 * lengthRoom - lengthPlatformA,
      -0.5 * lengthRoom + heightPlatformA
    >,
    <
      -0.5 * widthRoom + widthPlatformA - 2,
       0.5 * lengthRoom - lengthPlatformA + 4,
      -0.5 * lengthRoom + heightPlatformA + 1.0
    >)
  MakeWall(
    <
      -0.5 * widthRoom + widthPlatformA - 3,
       0.5 * lengthRoom - lengthPlatformA + 8,
      -0.5 * lengthRoom + heightPlatformA
    >,
    <
      -0.5 * widthRoom + widthPlatformA - 2,
       0.5 * lengthRoom - lengthPlatformA + 11,
      -0.5 * lengthRoom + heightPlatformA + 1.0
    >)


  scale scaleBlock
}
```

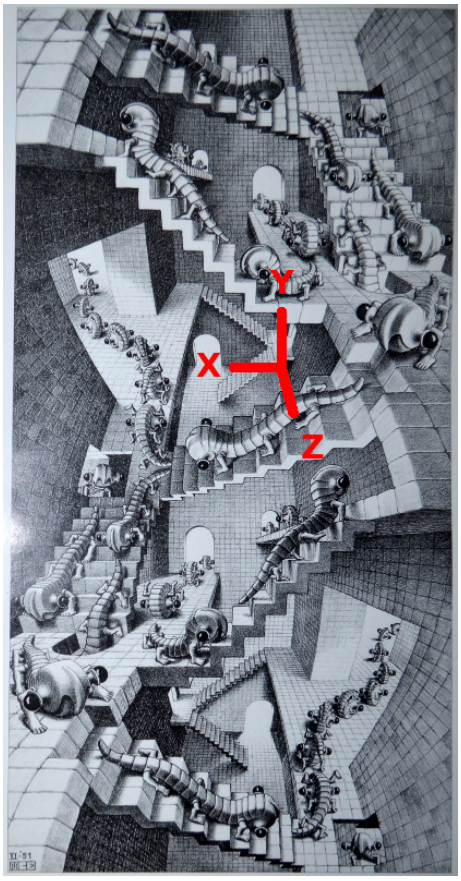
```
#declare PlatformB = union {
  MakeWall(
    <
      -0.5 * widthRoom,
      -0.5 * lengthRoom + heightPlatformB,
       -0.5 * lengthRoom
    >,
    <
      -0.5 * widthRoom + widthPlatformB,
      -0.5 * lengthRoom + heightPlatformB + 1.0,
       -0.5 * lengthRoom + lengthPlatformB
    >)
  scale scaleBlock
}

#declare PlatformC = union {
  MakeWall(
    <
       0.5 * widthRoom - widthPlatformC,
       0.5 * lengthRoom - lengthPlatformC,
      -0.5 * lengthRoom + heightPlatformC
    >,
    <
       0.5 * widthRoom,
       0.5 * lengthRoom,
      -0.5 * lengthRoom + heightPlatformC + 1.0
    >)
  scale scaleBlock
}

#declare PlatformD = union {
  MakeWall(
    <
       -0.5 * widthRoom,
       0.5 * lengthRoom - lengthPlatformD,
      -0.5 * lengthRoom + heightPlatformD
    >,
    <
       -0.5 * widthRoom + widthPlatformD,
       0.5 * lengthRoom,
      -0.5 * lengthRoom + heightPlatformD + 1.0
    >)
  scale scaleBlock
}

#declare Platforms = union {
  object { PlatformA }
  object { PlatformB }
  object { PlatformC }
  object { PlatformD }
}

// ------ Stairs definition ------

#declare StairsA = object {
  MakeStairs(
     widthStairsA,
     nbStairsA,
     posStairsA,
     z * slopeUpStairsA,
    -y,
```

```
    -x * slopeFrontStairsA)
  scale scaleBlock
}

#declare StairsB = object {
  MakeStairs(
    widthStairsB,
    nbStairsB,
    posStairsB,
    z * slopeUpStairsB,
    y,
    x * slopeFrontStairsB)
  scale scaleBlock
}

#declare StairsC = object {
  MakeStairs(
     widthStairsC,
     nbStairsC,
     posStairsC,
     z * slopeUpStairsC,
     x,
    -y * slopeFrontStairsC)
  scale scaleBlock
}

#declare StairsD = object {
  MakeStairs(
    widthStairsD,
    nbStairsD + 1, // +1 because the first stair is hidden in the wall
    posStairsD,
    z * slopeUpStairsD,
    y,
    -x * slopeFrontStairsD)
  scale scaleBlock
}

#declare StairsE = object {
  MakeStairs(
    widthStairsE,
    nbStairsE,
    posStairsE,
    z * slopeUpStairsE,
    y,
    x * slopeFrontStairsE)
  scale scaleBlock
}

#declare Stairs = union {
  object { StairsA }
  object { StairsB }
  object { StairsC }
  object { StairsD }
  object { StairsE }
  texture {pigment {color rgb 1.0}}
}

// ------ House of Stairs definition ------

#declare HouseOfStairs = union {

  // Loop on the four quarters of the house
```

```
  #declare iQuarter = 0;
  #while (iQuarter < 4)
    union {

      // Elements of one quarter
      object { SideWalls }
      object { Platforms }
      object { Stairs }

      // Rotate the quarter along the horizontal axis
      rotate x * 90.0 * iQuarter

      // Apply the symmetry to the odd quarters
      #if (iQuarter = 1 | iQuarter = 3)
        scale <-1.0, 1.0, 1.0>
      #end
    }
    #declare iQuarter = iQuarter + 1;
  #end
}

// ------ Camera ------

#declare posCamera = <0.0, 0.0, 0.0>;
#declare lookAt = <0.0, 0.2 * lengthRoom, -0.5 * lengthRoom>;

camera {
  cylinder 2
  angle 210
  location posCamera
  look_at lookAt
  right x * 3.3 //2.9
  up y
}

// ------ Light ------

light_source {
  posCamera
  color rgb 1.0
}

// ------ Scene ------

object {
  HouseOfStairs
}
```

4.4 Moving the origin to the corner of the House

Even if there are inconsistencies in the drawing I hope to find a better model and feel like there are problems in my model too. Then, I don't want to give up yet and try another approach. Until now I've placed the origin of the model at the center of the House, where the camera is. However it makes complication as the length of the House seems to be an odd number, and leads to half block in the center of the front wall when splitting it into two to apply symmetry. Thus, I want to try to rework the model by putting the origin of the world in the corner of the House.



blocks.inc:

```
// Seed for the randomization of blocks' texture and size

#declare seedBlock = seed(0);

// Roundness coefficient of the blocks

#declare blockRoundness = 0.075;

// Randomness coefficient for the block dimension

#declare blockRandomness = 0.025;

// Thickness coefficient of the stairs

#declare stairThickness = 1.0 / 2.0;

// Return a random value in range [1.0 - r / 2, 1.0 + r / 2]

#macro RandAroungOne(r)
  (1.0 + (0.5 - rand(seedBlock)) * r)
#end
```

```
// Return the texture of the blocks

#macro BlockTex()
  texture {
    pigment {
      color rgb (1.0 - blockRandomness) * RandAroungOne(0.25)
    }
    normal {
      bumps 0.25 scale .02
    }
    finish {
      ambient 0.0 diffuse 1.0
    }
  }
#end

// Return the scaling for the randomization of the size of the blocks

#macro BlockRandScale()
  0.5 * RandAroungOne(blockRandomness)
#end

// Return one block

#macro Block()
  superellipsoid {
    <blockRoundness, blockRoundness>
    BlockTex()
    scale BlockRandScale()
  }
#end

// Make the wall equivalent to box {posMin, posMax} made of blocks

#macro MakeWall(
    posMin,
    posMax)
  union {
    #local wallX = posMin.x;
    #while (wallX < posMax.x)
      #local wallY = posMin.y;
      #while (wallY < posMax.y)
        #local wallZ = posMin.z;
        #while (wallZ < posMax.z)
          object {
            Block()
            translate <wallX + 0.5, wallY + 0.5, wallZ + 0.5>
          }
          #declare wallZ = wallZ + 1.0;
        #end
        #declare wallY = wallY + 1.0;
      #end
      #declare wallX = wallX + 1.0;
    #end
  }
#end

// Return one stair

#macro MakeStairsSimple(
    widthStair,
```

```
    nbStair,
    startPos,
    upVec,
    rightVec,
    frontVec)
  union {
    #local oneStairBottom =
      -0.5 * widthStair * rightVec;
    #local oneStairTop =
      0.5 * widthStair * rightVec + upVec + frontVec;
    #local iStair = 0;
    #while (iStair < nbStair)
      box {
        oneStairBottom
        oneStairTop
        translate (frontVec + upVec) * iStair
      }
      #declare iStair = iStair + 1;
    #end
    translate startPos
    BlockTex()
  }
#end

#macro MakeStairs(
    widthStair,
    nbStair,
    startPos,
    upVec,
    rightVec,
    frontVec)
  union {
    #local oneStairBottom =
      -0.5 * widthStair * rightVec + (1.0 - stairThickness) * vnormalize(
          upVec);
    #local oneStairTop =
      0.5 * widthStair * rightVec + vnormalize(upVec) + frontVec;
    #local iStair = 0;
    #while (iStair < nbStair)
      #local jStair = 0;
      #while (jStair < widthStair)
        superellipsoid {
          <blockRoundness, blockRoundness>
          scale 0.5
          translate 0.5
          scale vnormalize(rightVec) + upVec * stairThickness + frontVec *
              (1.0 + blockRoundness)
          translate upVec * (1.0 - stairThickness)
          translate rightVec * (jStair - 0.5 * widthStair)
          translate (frontVec + upVec) * iStair
        }
        #if (iStair > 0)
          superellipsoid {
            <blockRoundness, blockRoundness>
            scale 0.5
            translate 0.5
            scale vnormalize(rightVec) + upVec * (1.0 + stairThickness) +
                frontVec * stairThickness
            translate -1.0 * upVec * stairThickness
            translate rightVec * (jStair - 0.5 * widthStair)
            translate (frontVec + upVec) * iStair
          }
```

```
        #end
        #declare jStair = jStair + 1;
      #end
      //box {
      //  oneStairBottom
      //  oneStairTop
      //  translate (frontVec + upVec) * iStair
      //}
      #declare iStair = iStair + 1;
    #end
    translate startPos
    BlockTex()
  }
#end
```

XL-51.pov:

```
// ------ Includes ------

// Include the macros relative to blocks

#include "blocks.inc"

// ------ Scene elements' dimensions ------

// Unit is one block size

// Scale of one block

#declare scaleBlock = <1.0, 0.5, 0.5>;

// Measured platforms' and stairs' dimensions

#declare nbStairsA = 10;
#declare widthStairsA = 4;

#declare widthStairsB = 4;
#declare nbStairsB = 8;
#declare slopeFrontStairsB = 1;

#declare widthStairsC = 6;
#declare nbStairsC = 9;
#declare slopeUpStairsC = 1;
#declare slopeFrontStairsC = 1;

#declare nbStairsD = 12;
#declare widthStairsD = 6;
#declare slopeUpStairsD = 1;
#declare slopeFrontStairsD = 1;

#declare nbStairsE = 15;
#declare widthStairsE = 6;
#declare slopeUpStairsE = 1;
#declare slopeFrontStairsE = 1;

#declare widthPlatformC = 6;

#declare widthPlatformD = 4;

#declare widthDoorE = 19;
#declare heightDoorE = 18;
```

```
// Empirically deducted dimensions

#declare lengthPlatformC = 44;

// Deducted platforms' and stairs' dimensions

#declare widthRoom = widthPlatformD + nbStairsE + widthPlatformC;

#declare heightPlatformD = nbStairsD;
#declare lengthPlatformD = widthStairsD + widthStairsE;

#declare posStairsD = <
  widthPlatformD + nbStairsD,
  0,
  widthStairsE + widthStairsD / 2
>;

#declare posStairsE = <
  widthPlatformD,
  nbStairsD + 1,
  widthStairsE / 2
>;

#declare heightPlatformC = nbStairsD + nbStairsE;

#declare heightGapPlatformBPlatformC = heightDoorE + 6;

#declare posStairsA = <
    widthRoom - widthPlatformC,
    heightPlatformC,
    lengthPlatformC + widthStairsA / 2
>;

#declare widthPlatformB = widthStairsC + 2;
#declare heightPlatformB = lengthPlatformC + widthStairsA + 3;
#declare lengthPlatformB = heightPlatformC + nbStairsC + widthStairsB + 2 +
    4;

#declare lengthRoom = heightPlatformB + heightPlatformC +
    heightGapPlatformBPlatformC + 2;

#declare posStairsC = <
    widthRoom - widthStairsC / 2,
    heightPlatformC,
    lengthPlatformC + widthStairsA
>;

#declare widthPlatformA = widthPlatformB + 2;
#declare lengthPlatformA = heightPlatformC + nbStairsC;
#declare heightPlatformA = lengthPlatformB;

#declare heightGapPlatformAPlatformB = 6;

#declare slopeUpStairsA = (heightPlatformA - heightPlatformC) / nbStairsA;
#declare slopeFrontStairsA = (widthRoom - widthPlatformA - widthPlatformC +
    3) / nbStairsA;

#declare posStairsB = <
    widthPlatformA,
    heightPlatformA,
    lengthPlatformA - widthStairsB / 2 + 5
```

```
>;

#declare slopeUpStairsB = heightGapPlatformAPlatformB / nbStairsB;

// ------- Side walls definition -------

#declare SideWallRight = union {
  MakeWall(
    <
      -1,
      0,
      0
    >,
    <
      0,
      (lengthRoom - 2) / 2,
      (lengthRoom + 1) / 2
    >)
  scale scaleBlock
}

#declare SideWallUp = union {
  MakeWall(
    <
      0,
      0,
      -1
    >,
    <
      widthRoom,
      (lengthRoom - 2) / 2,
      0
    >)
  scale scaleBlock
}

#declare SideWallDown = union {
  MakeWall(
    <
      0,
      -1,
      0
    >,
    <
      widthRoom,
      0,
      lengthRoom / 2 + 1
    >)
  scale scaleBlock
}

#declare SideWallLeft = union {
  MakeWall(
    <
      widthRoom,
      0,
      0
    >,
    <
      widthRoom + 1,
      (lengthRoom - 2) / 2,
      (lengthRoom + 1) / 2
```

```
    >)
  scale scaleBlock
}

#declare SideWalls = union {
  object { SideWallRight }
  object { SideWallUp }
  object { SideWallDown }
  object { SideWallLeft }
}

// ------- Platforms definition -------

#declare PlatformA = union {
  MakeWall(
    <
      0,
      heightPlatformA,
      -10
    >,
    <
      widthPlatformA,
      heightPlatformA + 1,
      lengthPlatformA
    >)
  MakeWall(
    <
      widthPlatformA - 4,
      heightPlatformA,
      lengthPlatformA
    >,
    <
      widthPlatformA,
      heightPlatformA + 1,
      lengthPlatformA + 4
    >)
  MakeWall(
    <
      0,
      heightPlatformA,
      lengthPlatformA
    >,
    <
      3,
      heightPlatformA + 1,
      lengthPlatformA + 4
    >)
  MakeWall(
    <
      0,
      heightPlatformA,
      lengthPlatformA + 4
    >,
    <
      widthPlatformA,
      heightPlatformA + 1,
      lengthPlatformA + 5
    >)
  MakeWall(
    <
      0,
      heightPlatformA,
```

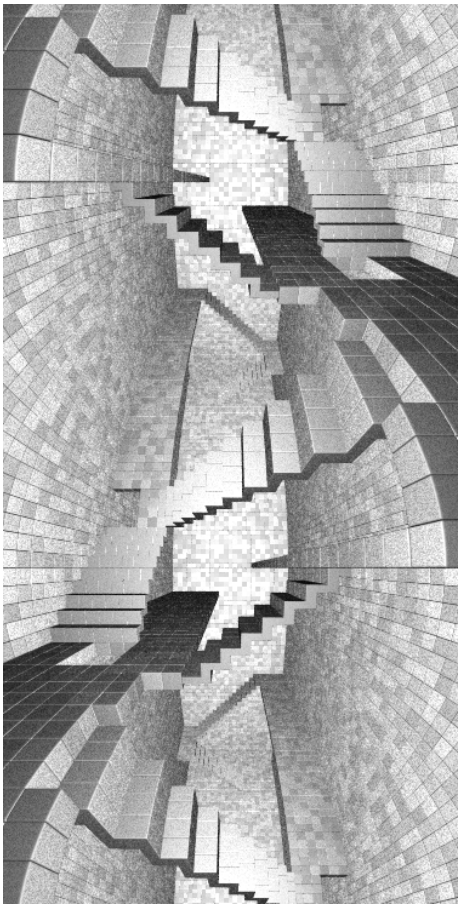
```
      lengthPlatformA + 5
    >,
    <
      widthPlatformB,
      heightPlatformA + 1,
      lengthPlatformA + 8
    >)
  MakeWall(
    <
      0,
      heightPlatformA,
      lengthPlatformA + 8
    >,
    <
      widthPlatformB - 1,
      heightPlatformA + 1,
      lengthPlatformA + 12
    >)
  MakeWall(
    <
      0,
      heightPlatformA,
      lengthPlatformA + 12
    >,
    <
      widthPlatformB,
      heightPlatformA + 1,
      lengthPlatformA + 16
    >)
  scale scaleBlock
}

#declare PlatformB = union {
  MakeWall(
    <
      0,
      0,
      heightPlatformB
    >,
    <
      widthPlatformB,
      lengthPlatformB,
      heightPlatformB + 1
    >)
  scale scaleBlock
}

#declare PlatformC = union {
  MakeWall(
    <
      widthRoom - widthPlatformC,
      heightPlatformC,
      0
    >,
    <
      widthRoom,
      heightPlatformC + 1,
      lengthPlatformC
    >)
  MakeWall(
    <
      widthRoom - widthPlatformC,
```

```
      heightPlatformC,
      lengthPlatformC
    >,
    <
      widthRoom - 3,
      heightPlatformC + 1,
      lengthPlatformC + widthStairsA
    >)
  scale scaleBlock
}

#declare PlatformD = union {
  MakeWall(
    <
      -widthPlatformD,
      heightPlatformD,
      0
    >,
    <
      widthPlatformD,
      heightPlatformD + 1,
      lengthPlatformD
    >)
  scale scaleBlock
}

#declare Platforms = union {
  object { PlatformA }
  object { PlatformB }
  object { PlatformC }
  object { PlatformD }
}

// ------ Stairs definition ------

#declare StairsA = object {
  MakeStairs(
     widthStairsA,
     nbStairsA,
     posStairsA,
     y * slopeUpStairsA,
     z,
    -x * slopeFrontStairsA)
  scale scaleBlock
}

#declare StairsB = object {
  MakeStairs(
    widthStairsB,
    nbStairsB,
    posStairsB,
    y * slopeUpStairsB,
    z,
    x * slopeFrontStairsB)
  scale scaleBlock
}

#declare StairsC = object {
  MakeStairs(
    widthStairsC,
    nbStairsC,
    posStairsC,
```

```
    y * slopeUpStairsC,
    x,
    z * slopeFrontStairsC)
  scale scaleBlock
}

#declare StairsD = object {
  MakeStairs(
     widthStairsD,
     nbStairsD,
     posStairsD,
     y * slopeUpStairsD,
     z,
    -x * slopeFrontStairsD)
  scale scaleBlock
}

#declare StairsE = object {
  MakeStairs(
    widthStairsE,
    nbStairsE,
    posStairsE,
    y * slopeUpStairsE,
    z,
    x * slopeFrontStairsE)
  scale scaleBlock
}

#declare Stairs = union {
  object { StairsA }
  object { StairsB }
  object { StairsC }
  object { StairsD }
  object { StairsE }
  texture {pigment {color rgb 1.0}}
}

// ------ House of Stairs definition ------

#declare HouseOfStairs = union {

  // Loop on the four quarters of the house
  #declare iQuarter = 0;
  #declare nbQuarter = 4;
  #while (iQuarter < nbQuarter)
    union {

      // Elements of one quarter
      object { SideWalls }
      object { Platforms }
      object { Stairs }

      // Rotate the quarter along the horizontal axis
      translate <0, -lengthRoom / 2, -lengthRoom /2> * scaleBlock
      rotate x * 90.0 * iQuarter
      translate <0, lengthRoom / 2, lengthRoom /2> * scaleBlock

      // Apply the symmetry to the odd quarters
      #if (iQuarter = 1 | iQuarter = 3)
        scale <-1.0, 1.0, 1.0>
        translate widthRoom * x * scaleBlock
      #end
```

```
    }
    #declare iQuarter = iQuarter + 1;
  #end
}

// ------ Camera ------

#declare posCamera = <widthRoom / 2, lengthRoom / 2, lengthRoom / 2> *
    scaleBlock;
#declare lookAt = <widthRoom / 2, 0, 0.25 * lengthRoom> * scaleBlock;

camera {
  cylinder 2
  angle 210
  location posCamera
  look_at lookAt
  right x * 3.8
  up y
}

// ------ Light ------

light_source {
  posCamera
  color rgb 1.0
}

// ------ Scene ------

object {
  HouseOfStairs
}

// ------ Frame ------

#declare QuarterFrame = union {
  #local r = 0.1;
  cylinder {
    <0, 0, 0>,
    <widthRoom, 0, 0>,
    r
  }
  cylinder {
    <0, 0, 0>,
    <0, lengthRoom / 2, 0>,
    r
  }
  cylinder {
    <0, 0, 0>,
    <0, 0, lengthRoom / 2>,
    r
  }
  cylinder {
    <widthRoom, 0, 0>,
    <widthRoom, lengthRoom / 2, 0>,
    r
  }
  cylinder {
    <widthRoom, 0, 0>,
    <widthRoom, 0, lengthRoom / 2>,
    r
  }
```

```
  cylinder {
    <0, lengthRoom / 2, 0>,
    <widthRoom, lengthRoom / 2, 0>,
    r
  }
  cylinder {
    <0, 0, lengthRoom / 2>,
    <widthRoom, 0, lengthRoom / 2>,
    r
  }
  cylinder {
    <0, lengthRoom / 2, 0>,
    <0, lengthRoom / 2, lengthRoom / 2>,
    r
  }
  cylinder {
    <0, 0, lengthRoom / 2>,
    <0, lengthRoom / 2, lengthRoom / 2>,
    r
  }
  cylinder {
    <widthRoom, lengthRoom / 2, 0>,
    <widthRoom, lengthRoom / 2, lengthRoom / 2>,
    r
  }
  cylinder {
    <widthRoom, 0, lengthRoom / 2>,
    <widthRoom, lengthRoom / 2, lengthRoom / 2>,
    r
  }
  texture {pigment {color rgb x}}
  scale scaleBlock
}

//object { QuarterFrame }
```



There are still parts not matching the lithography but this model is the one that please me the most until now and there is no way to match it perfectly due to inconsistencies. I could certainly find a solution if I give up on the symetric generation of the House, i.e. not building the House by modeling only one quarter and rotating and applying a symmetry to get the others, and instead modeling each element individually.

But I think it would not be respectful of the spirit of Escher. He was telling of himself he's not an artist but a mathematician, so I believe a model mathematically correct is nearer to what he was actually thinking of.

By the way the mistakes in his lithography don't decrease my love for his work at all. Rather, going myself through the complexity of what he made by hand while I'm doing it with a computer makes me more than ever

respectful and appreciative of his genius.

4.5 Finalization of the model

Now I add the doors, external walls. I also do some minute arrangement to the dimensions of the stairs for a perfect fit and extend the platform outside the House where necessary. And the final model for the House is finished.

blocks.inc:

```
// Seed for the randomization of blocks' texture and size

#declare seedBlock = seed(0);

// Roundness coefficient of the blocks

#declare blockRoundness = 0.075;

// Randomness coefficient for the block dimension

#declare blockRandomness = 0.025;

// Thickness coefficient of the stairs

#declare stairThickness = 1.0 / 2.0;

// Return a random value in range [1.0 - r / 2, 1.0 + r / 2]

#macro RandAroungOne(r)
  (1.0 + (0.5 - rand(seedBlock)) * r)
#end

// Return the texture of the blocks

#macro BlockTex()
  texture {
    pigment {
      color rgb (1.0 - blockRandomness) * RandAroungOne(0.25)
    }
    normal {
      bumps 0.25 scale .02
    }
    finish {
      ambient 0.0 diffuse 1.0
    }
  }
#end

// Return the scaling for the randomization of the size of the blocks

#macro BlockRandScale()
  0.5 * RandAroungOne(blockRandomness)
#end

// Return one block
```

```
#macro Block()
  superellipsoid {
    <blockRoundness, blockRoundness>
    BlockTex()
    scale BlockRandScale()
  }
#end

// Make the wall equivalent to box {posMin, posMax} made of blocks

#macro MakeWall(
    posMin,
    posMax)
  union {
    #local wallX = posMin.x;
    #while (wallX < posMax.x)
      #local wallY = posMin.y;
      #while (wallY < posMax.y)
        #local wallZ = posMin.z;
        #while (wallZ < posMax.z)
          object {
            Block()
            translate <wallX + 0.5, wallY + 0.5, wallZ + 0.5>
          }
          #declare wallZ = wallZ + 1.0;
        #end
        #declare wallY = wallY + 1.0;
      #end
      #declare wallX = wallX + 1.0;
    #end
  }
#end

// Return one stair

#macro MakeStairsSimple(
    widthStair,
    nbStair,
    startPos,
    upVec,
    rightVec,
    frontVec)
  union {
    #local oneStairBottom =
      -0.5 * widthStair * rightVec;
    #local oneStairTop =
      0.5 * widthStair * rightVec + upVec + frontVec;
    #local iStair = 0;
    #while (iStair < nbStair)
      box {
        oneStairBottom
        oneStairTop
        translate (frontVec + upVec) * iStair
      }
      #declare iStair = iStair + 1;
    #end
    translate startPos
    BlockTex()
  }
#end

#macro MakeStairs(
```

```
    widthStair,
    nbStair,
    startPos,
    upVec,
    rightVec,
    frontVec)
  union {
    #local oneStairBottom =
      -0.5 * widthStair * rightVec + (1.0 - stairThickness) * vnormalize(
          upVec);
    #local oneStairTop =
      0.5 * widthStair * rightVec + vnormalize(upVec) + frontVec;
    #local iStair = 0;
    #while (iStair < nbStair)
      #local jStair = 0;
      #while (jStair < widthStair)
        superellipsoid {
          <blockRoundness, blockRoundness>
          scale 0.5
          translate 0.5
          scale vnormalize(rightVec) + upVec * stairThickness + frontVec *
              (1.0 + blockRoundness)
          translate upVec * (1.0 - stairThickness)
          translate rightVec * (jStair - 0.5 * widthStair)
          translate (frontVec + upVec) * iStair
        }
        #if (iStair > 0)
          superellipsoid {
            <blockRoundness, blockRoundness>
            scale 0.5
            translate 0.5
            scale vnormalize(rightVec) + upVec * (1.0 + stairThickness) +
                frontVec * stairThickness
            translate -1.0 * upVec * stairThickness
            translate rightVec * (jStair - 0.5 * widthStair)
            translate (frontVec + upVec) * iStair
          }
        #end
        #declare jStair = jStair + 1;
      #end
      //box {
      //  oneStairBottom
      //  oneStairTop
      //  translate (frontVec + upVec) * iStair
      //}
      #declare iStair = iStair + 1;
    #end
    translate startPos
    BlockTex()
  }
#end
```

XL-51.pov:

```
// ------ Includes ------

// Include the macros relative to blocks

#include "blocks.inc"

// Rendering parameters
```

```
background { color rgb 1.0 }

// ------ Scene elements' dimensions ------

// Unit is one block size

// Scale of one block

#declare scaleBlock = <1.0, 0.5, 0.5>;

// Measured platforms' and stairs' dimensions

#declare widthStairsA = 4;

#declare widthStairsB = 4;
#declare nbStairsB = 8;
#declare slopeFrontStairsB = 1;

#declare widthStairsC = 6;
#declare nbStairsC = 9;
#declare slopeUpStairsC = 1;
#declare slopeFrontStairsC = 1;

#declare nbStairsD = 12;
#declare widthStairsD = 6;
#declare slopeUpStairsD = 1;
#declare slopeFrontStairsD = 1;

#declare nbStairsE = 15;
#declare widthStairsE = 6;
#declare slopeUpStairsE = 1;
#declare slopeFrontStairsE = 1;

#declare widthPlatformC = 6;

#declare widthPlatformD = 4;

#declare heightDoorA = 11;
#declare widthDoorA = 7;

#declare heightDoorB = 8;
#declare widthDoorB = 6;

#declare heightDoorC = 9;
#declare widthDoorC = 6;

#declare heightDoorD = 6;
#declare widthDoorD = 4;

#declare widthDoorE = 19;
#declare heightDoorE = 18;

#declare widthDoorG = 4;
#declare heightDoorG = 2;

// Empirically deducted or modified dimensions to correct
// inconsistencies

#declare lengthPlatformC = 44;
#declare nbStairsA = 12;
```

```
// Deducted platforms' and stairs' dimensions

#declare widthRoom = widthPlatformD + nbStairsE + widthPlatformC;

#declare heightPlatformD = nbStairsD;
#declare lengthPlatformD = widthStairsD + widthStairsE;

#declare posStairsD = <
  widthPlatformD + nbStairsD,
  0,
  widthStairsE + widthStairsD / 2
>;

#declare posStairsE = <
  widthPlatformD,
  nbStairsD + 1,
  widthStairsE / 2
>;

#declare heightPlatformC = nbStairsD + nbStairsE;

#declare heightGapPlatformBPlatformC = heightDoorE + 6;

#declare posStairsA = <
    widthRoom - widthPlatformC,
    heightPlatformC,
    lengthPlatformC + widthStairsA / 2
>;

#declare widthPlatformB = widthStairsC + 2;
#declare heightPlatformB = lengthPlatformC + widthStairsA + 3;
#declare lengthPlatformB = heightPlatformC + nbStairsC + widthStairsB + 2 +
    4;

#declare lengthRoom = heightPlatformB + heightPlatformC +
    heightGapPlatformBPlatformC + 2;

#declare posStairsC = <
    widthRoom - widthStairsC / 2,
    heightPlatformC,
    lengthPlatformC + widthStairsA
>;

#declare widthPlatformA = widthPlatformB + 2;
#declare lengthPlatformA = heightPlatformC + nbStairsC;
#declare heightPlatformA = lengthPlatformB;

#declare heightGapPlatformAPlatformB = 6;

#declare slopeUpStairsA = (heightPlatformA - heightPlatformC) / nbStairsA;
#declare slopeFrontStairsA = (widthRoom - widthPlatformA - widthPlatformC +
    3) / nbStairsA;

#declare posStairsB = <
    widthPlatformA,
    heightPlatformA,
    lengthPlatformA - widthStairsB / 2 + 5
>;

#declare slopeUpStairsB = heightGapPlatformAPlatformB / nbStairsB;

// ------- Side walls definition -------
```

```
#declare SideWallRight = union {
  MakeWall(
    <
      -1,
      0,
      0
    >,
    <
      0,
      (lengthRoom - 2) / 2,
      3
    >)
  MakeWall(
    <
      -1,
      0,
      3
    >,
    <
      0,
      heightPlatformD,
      3 + widthDoorB
    >)
  MakeWall(
    <
      -1,
      heightPlatformD + 1 + heightDoorB,
      3
    >,
    <
      0,
      (lengthRoom - 2) / 2,
      3 + widthDoorB
    >)
  MakeWall(
    <
      -1,
      0,
      3 + widthDoorB
    >,
    <
      0,
      (lengthRoom - 2) / 2,
      (lengthRoom + 1) / 2
    >)
  scale scaleBlock
}

#declare SideWallUp = union {
  difference {
    MakeWall(
      <
        0,
        heightDoorA - 4,
        -1
      >,
      <
        widthRoom,
        (lengthRoom - 2) / 2 - 4,
        0
      >)
```

```
    cylinder {
      <0, 0, -10>, <0, 0, 10>, 1
      scale <widthDoorA / 2, 4, 1>
      translate <
        widthRoom - 2 - widthDoorA / 2,
        heightDoorA - 4,
        0>
    }
  }
  MakeWall(
    <
      0,
      (lengthRoom - 2) / 2 - 4,
      -1
    >,
    <
      3,
      (lengthRoom - 2) / 2,
      0
    >)
  MakeWall(
    <
      3 + widthDoorC,
      (lengthRoom - 2) / 2 - 4,
      -1
    >,
    <
      widthRoom,
      (lengthRoom - 2) / 2,
      0
    >)
  MakeWall(
    <
      0,
      0,
      -1
    >,
    <
      widthRoom - 2 - widthDoorA,
      heightDoorA - 4,
      0
    >)
  MakeWall(
    <
      widthRoom - 2,
      0,
      -1
    >,
    <
      widthRoom,
      heightDoorA - 4,
      0
    >)
  scale scaleBlock
}

#declare SideWallDown = union {
  MakeWall(
    <
      0,
      -1,
      -lengthPlatformD * 2
```

```
  >,
  <
    3,
    0,
    lengthRoom / 2 + 1
  >)
difference {
  MakeWall(
    <
      3,
      -1,
      -lengthPlatformD * 2
    >,
    <
      3 + widthDoorD,
      0,
      lengthRoom / 2 - heightDoorD + 2
    >)
  cylinder {
    <0, -10, 0> <0, 10, 0> 1
    scale <widthDoorD / 2, 1, 2>
    translate <
      3 + widthDoorD / 2,
      0,
      lengthRoom / 2 - heightDoorD + 2>
  }
}
MakeWall(
  <
    3,
    -1,
    lengthRoom / 2
  >,
  <
    3 + widthDoorD,
    0,
    lengthRoom / 2 + 1
  >)
MakeWall(
  <
    3 + widthDoorD,
    -1,
    -lengthPlatformD * 2
  >,
  <
    widthRoom - 3 - widthDoorC,
    0,
    lengthRoom / 2 + 1
  >)
difference {
  MakeWall(
    <
      widthRoom - 3 - widthDoorC,
      -1,
      -lengthPlatformD * 2
    >,
    <
      widthRoom - 3,
      0,
      lengthRoom / 2 - (heightDoorC - 5) + 2
    >)
  cylinder {
```

```
      <0, -10, 0> <0, 10, 0> 1
      scale <widthDoorC / 2, 1, 2>
      translate <
        widthRoom - 3 - widthDoorC / 2,
        0,
        lengthRoom / 2 - (heightDoorC - 5) + 2>
    }
  }
  MakeWall(
    <
      widthRoom - 3,
      -1,
      -lengthPlatformD * 2
    >,
    <
      widthRoom,
      0,
      lengthRoom / 2 + 1
    >)


  scale scaleBlock
}

#declare SideWallLeft = union {
  MakeWall(
    <
      widthRoom,
      0,
      0
    >,
    <
      widthRoom + 1,
      (lengthRoom - 2) / 2,
      19
    >)
  MakeWall(
    <
      widthRoom,
      0,
      19
    >,
    <
      widthRoom + 1,
      heightPlatformC,
      19 + widthDoorE
    >)
  MakeWall(
    <
      widthRoom,
      heightPlatformC + 1 + heightDoorE,
      19
    >,
    <
      widthRoom + 1,
      (lengthRoom - 2) / 2,
      19 + widthDoorE
    >)
  MakeWall(
    <
      widthRoom + 1,
      heightPlatformC + 1,
```

```
    18
  >,
  <
    widthRoom + 6,
    heightPlatformC + 1 + heightDoorE,
    19
  >)
MakeWall(
  <
    widthRoom + 1,
    heightPlatformC + 1,
    19 + widthDoorE
  >,
  <
    widthRoom + 6,
    heightPlatformC + 1 + heightDoorE,
    19 + widthDoorE + 1
  >)
MakeWall(
  <
    widthRoom + 17,
    heightPlatformC + 1,
    19 - 8
  >,
  <
    widthRoom + 18,
    heightPlatformC + 1 + heightDoorE,
    19 + widthDoorE + 8
  >)
MakeWall(
  <
    widthRoom,
    0,
    19 + widthDoorE
  >,
  <
    widthRoom + 1,
    (lengthRoom - 2) / 2,
    19 + widthDoorE + 6
  >)
MakeWall(
  <
    widthRoom,
    0,
    19 + widthDoorE + 6
  >,
  <
    widthRoom + 1,
    heightPlatformC + 1 - heightDoorG,
    19 + widthDoorE + 6 + widthDoorG
  >)
MakeWall(
  <
    widthRoom,
    heightPlatformC + 1 + heightDoorG,
    19 + widthDoorE + 6
  >,
  <
    widthRoom + 1,
    (lengthRoom - 2) / 2,
    19 + widthDoorE + 6 + widthDoorG
  >)
```

```
  MakeWall(
    <
      widthRoom + 1,
      heightPlatformC + 1 - heightDoorG,
      19 + widthDoorE + 5
    >,
    <
      widthRoom + 5,
      heightPlatformC + 1 + heightDoorG,
      19 + widthDoorE + 6
    >)
  MakeWall(
    <
      widthRoom + 1,
      heightPlatformC + 1 - heightDoorG,
      19 + widthDoorE + 6 + widthDoorG
    >,
    <
      widthRoom + 5,
      heightPlatformC + 1 + heightDoorG,
      19 + widthDoorE + 6 + widthDoorG + 1
    >)
  MakeWall(
    <
      widthRoom + 1,
      heightPlatformC + 1 - heightDoorG - 1,
      19 + widthDoorE + 6
    >,
    <
      widthRoom + 5,
      heightPlatformC + 1 - heightDoorG,
      19 + widthDoorE + 6 + widthDoorG
    >)
  MakeWall(
    <
      widthRoom,
      0,
      19 + widthDoorE + 6 + widthDoorG
    >,
    <
      widthRoom + 1,
      (lengthRoom - 2) / 2,
      (lengthRoom + 1) / 2
    >)
  scale scaleBlock
}

#declare SideWalls = union {
  object { SideWallRight }
  object { SideWallUp }
  object { SideWallDown }
  object { SideWallLeft }
}

// ------- Platforms definition -------

#declare PlatformA = union {
  MakeWall(
    <
      0,
      heightPlatformA,
      -20
```

```
  >,
  <
    widthPlatformA,
    heightPlatformA + 1,
    lengthPlatformA
  >)
MakeWall(
  <
    widthPlatformA - 4,
    heightPlatformA,
    lengthPlatformA
  >,
  <
    widthPlatformA,
    heightPlatformA + 1,
    lengthPlatformA + 2
  >)
MakeWall(
  <
    0,
    heightPlatformA,
    lengthPlatformA
  >,
  <
    3,
    heightPlatformA + 1,
    lengthPlatformA + 2
  >)
difference {
  MakeWall(
    <
      0,
      heightPlatformA,
      lengthPlatformA + 2
    >,
    <
      widthPlatformA,
      heightPlatformA + 1,
      lengthPlatformA + 4
    >)
  cylinder {
    <0, -10, 0> <0, 10, 0>, 1
    scale <1.5, 1, 1>
    translate <4.5, heightPlatformA, lengthPlatformA + 2>
  }
}
MakeWall(
  <
    0,
    heightPlatformA,
    lengthPlatformA + 4
  >,
  <
    widthPlatformA,
    heightPlatformA + 1,
    lengthPlatformA + 5
  >)
MakeWall(
  <
    0,
    heightPlatformA,
    lengthPlatformA + 5
```

```
    >,
    <
      widthPlatformB,
      heightPlatformA + 1,
      lengthPlatformA + 8
    >)
  MakeWall(
    <
      0,
      heightPlatformA,
      lengthPlatformA + 8
    >,
    <
      widthPlatformB - 1,
      heightPlatformA + 1,
      lengthPlatformA + 12
    >)
  MakeWall(
    <
      0,
      heightPlatformA,
      lengthPlatformA + 12
    >,
    <
      widthPlatformB,
      heightPlatformA + 1,
      lengthPlatformA + 16
    >)
  MakeWall(
    <
      0,
      heightPlatformA - widthStairsC / 2,
      lengthPlatformA - 1
    >,
    <
      widthStairsC,
      heightPlatformA,
      lengthPlatformA
    >)
  object {
    Block()
    scale <1, 1, 0.5>
    translate <
      widthPlatformB - 0.5,
      heightPlatformA + 0.5,
      lengthPlatformA + 8.25>
  }
  object {
    Block()
    scale <1, 1, 0.5>
    translate <
      widthPlatformB - 0.5,
      heightPlatformA + 0.5,
      lengthPlatformA + 11.75>
  }
  scale scaleBlock
}

#declare PlatformB = union {
  MakeWall(
    <
      0,
```

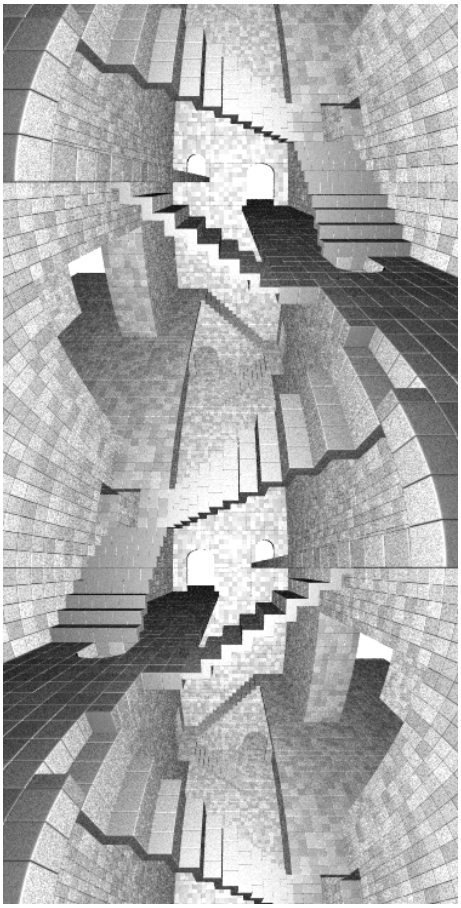
```
      0,
      heightPlatformB
    >,
    <
      widthPlatformB,
      lengthPlatformB,
      heightPlatformB + 1
    >)
  scale scaleBlock
}

#declare PlatformC = union {
  MakeWall(
    <
      widthRoom - widthPlatformC,
      heightPlatformC,
      -10
    >,
    <
      widthRoom + 30,
      heightPlatformC + 1,
      lengthPlatformC
    >)
  MakeWall(
    <
      widthRoom - widthPlatformC,
      heightPlatformC,
      lengthPlatformC
    >,
    <
      widthRoom - 3,
      heightPlatformC + 1,
      lengthPlatformC + widthStairsA
    >)
  MakeWall(
    <
      widthRoom - 3,
      heightPlatformC - 3,
      lengthPlatformC + widthStairsA
    >,
    <
      widthRoom,
      heightPlatformC,
      lengthPlatformC + widthStairsA + 1
    >)
  scale scaleBlock
}

#declare PlatformD = union {
  MakeWall(
    <
      -widthPlatformD,
      heightPlatformD,
      -lengthPlatformD
    >,
    <
      widthPlatformD,
      heightPlatformD + 1,
      lengthPlatformD
    >)
  scale scaleBlock
}
```

```
#declare Platforms = union {
  object { PlatformA }
  object { PlatformB }
  object { PlatformC }
  object { PlatformD }
}

// ------ Stairs definition ------

#declare StairsA = object {
  MakeStairs(
     widthStairsA,
     nbStairsA,
     posStairsA - y * 0.6,
     y * slopeUpStairsA * 1.03,
     z,
    -x * slopeFrontStairsA * 1.0)
  scale scaleBlock
}

#declare StairsB = object {
  MakeStairs(
    widthStairsB,
    nbStairsB,
    posStairsB + y * 0.25,
    y * slopeUpStairsB * 0.97,
    z,
    x * slopeFrontStairsB * 1.02)
  scale scaleBlock
}

#declare StairsC = object {
  MakeStairs(
    widthStairsC,
    nbStairsC,
    posStairsC,
    y * slopeUpStairsC,
    x,
    z * slopeFrontStairsC)
  scale scaleBlock
}

#declare StairsD = object {
  MakeStairs(
     widthStairsD,
     nbStairsD,
     posStairsD,
     y * slopeUpStairsD,
     z,
    -x * slopeFrontStairsD)
  scale scaleBlock
}

#declare StairsE = object {
  MakeStairs(
    widthStairsE,
    nbStairsE,
    posStairsE,
    y * slopeUpStairsE,
    z,
    x * slopeFrontStairsE)
```

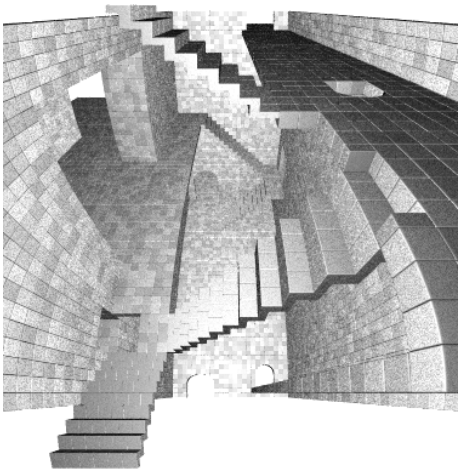
```
  scale scaleBlock
}

#declare Stairs = union {
  object { StairsA }
  object { StairsB }
  object { StairsC }
  object { StairsD }
  object { StairsE }
  texture {pigment {color rgb 1.0}}
}

// ------ House of Stairs definition ------

#declare HouseOfStairs = union {

  // Loop on the four quarters of the house
  #declare iQuarter = 0;
  #declare nbQuarter = 4;
  #while (iQuarter < nbQuarter)
    union {

      // Elements of one quarter
      object { SideWalls }
      object { Platforms }
      object { Stairs }

      // Rotate the quarter along the horizontal axis
      translate <0, -lengthRoom / 2, -lengthRoom / 2> * scaleBlock
      rotate x * 90.0 * iQuarter
      translate <0, lengthRoom / 2, lengthRoom / 2> * scaleBlock

      // Apply the symmetry to the odd quarters
      #if (iQuarter = 1 | iQuarter = 3)
        scale <-1.0, 1.0, 1.0>
        translate widthRoom * x * scaleBlock
      #end
    }
    #declare iQuarter = iQuarter + 1;
  #end
}

// ------ Camera ------

#declare posCamera = <widthRoom / 2, lengthRoom / 2, lengthRoom / 2> *
    scaleBlock;
#declare lookAt = <widthRoom / 2, 0, 0.25 * lengthRoom> * scaleBlock;

//#declare posCamera = (posStairsA - 3 + 3 * y) * scaleBlock;
//#declare lookAt = posStairsA * scaleBlock;

camera {
  cylinder 2
  angle 210
  location posCamera
  look_at lookAt
  right x * 3.8
  up y
}

// ------ Light ------
```

```
light_source {
  posCamera
  color rgb 1.0
}

// ------ Scene ------

object {
  HouseOfStairs
}

// ------ Frame ------

#declare QuarterFrame = union {
  #local r = 0.1;
  cylinder {
    <0, 0, 0>,
    <widthRoom, 0, 0>,
    r
  }
  cylinder {
    <0, 0, 0>,
    <0, lengthRoom / 2, 0>,
    r
  }
  cylinder {
    <0, 0, 0>,
    <0, 0, lengthRoom / 2>,
    r
  }
  cylinder {
    <widthRoom, 0, 0>,
    <widthRoom, lengthRoom / 2, 0>,
    r
  }
  cylinder {
    <widthRoom, 0, 0>,
    <widthRoom, 0, lengthRoom / 2>,
    r
  }
  cylinder {
    <0, lengthRoom / 2, 0>,
    <widthRoom, lengthRoom / 2, 0>,
    r
  }
  cylinder {
    <0, 0, lengthRoom / 2>,
    <widthRoom, 0, lengthRoom / 2>,
    r
  }
  cylinder {
    <0, lengthRoom / 2, 0>,
    <0, lengthRoom / 2, lengthRoom / 2>,
    r
  }
  cylinder {
    <0, 0, lengthRoom / 2>,
    <0, lengthRoom / 2, lengthRoom / 2>,
    r
  }
  cylinder {
    <widthRoom, lengthRoom / 2, 0>,
```

```
    <widthRoom, lengthRoom / 2, lengthRoom / 2>,
    r
  }
  cylinder {
    <widthRoom, 0, lengthRoom / 2>,
    <widthRoom, lengthRoom / 2, lengthRoom / 2>,
    r
  }
  texture {pigment {color rgb x}}
  scale scaleBlock
}

//object { QuarterFrame }
```



The single quarter which is repeated four time is shown below:



5 Light model

Now I have to solve another problem which is the lighting of the House.

Until now I was using one point light at the location of the camera. Instead I want to use the shadows in the lithography to search for the position of the lights. For example we see clearly that there is light coming from the outside of the doors in the front wall. I start from there.

blocks.inc:

```
// Seed for the randomization of blocks' texture and size

#declare seedBlock = seed(0);

// Roundness coefficient of the blocks

#declare blockRoundness = 0.075;

// Randomness coefficient for the block dimension

#declare blockRandomness = 0.025;

// Thickness coefficient of the stairs

#declare stairThickness = 1.0 / 2.0;

// Return a random value in range [1.0 - r / 2, 1.0 + r / 2]

#macro RandAroungOne(r)
  (1.0 + (0.5 - rand(seedBlock)) * r)
```

```
#end

// Return the texture of the blocks

#macro BlockTex()
  texture {
    pigment {
      color rgb (1.0 - blockRandomness) * RandAroungOne(0.25)
    }
    normal {
      bumps 0.25 scale .02
    }
    finish {
      ambient 0.0 diffuse 1.0
    }
  }
#end

// Return the scaling for the randomization of the size of the blocks

#macro BlockRandScale()
  0.5 * RandAroungOne(blockRandomness)
#end

// Return one block

#macro Block()
  superellipsoid {
    <blockRoundness, blockRoundness>
    BlockTex()
    scale BlockRandScale()
  }
#end

// Make the wall equivalent to box {posMin, posMax} made of blocks

#macro MakeWallSimple(
    posMin,
    posMax)
  box {
    posMin, posMax
    BlockTex()
    //texture { pigment {color rgb 1.0 } }
  }
#end

#macro MakeWall(
    posMin,
    posMax)
  union {
    #local wallX = posMin.x;
    #while (wallX < posMax.x)
      #local wallY = posMin.y;
      #while (wallY < posMax.y)
        #local wallZ = posMin.z;
        #while (wallZ < posMax.z)
          object {
            Block()
            translate <wallX + 0.5, wallY + 0.5, wallZ + 0.5>
          }
          #declare wallZ = wallZ + 1.0;
        #end
```

```
        #declare wallY = wallY + 1.0;
      #end
      #declare wallX = wallX + 1.0;
    #end
  }
#end

// Return one stair

#macro MakeStairsSimple(
    widthStair,
    nbStair,
    startPos,
    upVec,
    rightVec,
    frontVec)
  union {
    #local oneStairBottom =
      -0.5 * widthStair * rightVec;
    #local oneStairTop =
      0.5 * widthStair * rightVec + upVec + frontVec;
    #local iStair = 0;
    #while (iStair < nbStair)
      box {
        oneStairBottom
        oneStairTop
        translate (frontVec + upVec) * iStair
      }
      #declare iStair = iStair + 1;
    #end
    translate startPos
    BlockTex()
    //texture { pigment {color rgb 1.0 } }
  }
#end

#macro MakeStairs(
    widthStair,
    nbStair,
    startPos,
    upVec,
    rightVec,
    frontVec)
  union {
    #local oneStairBottom =
      -0.5 * widthStair * rightVec + (1.0 - stairThickness) * vnormalize(
          upVec);
    #local oneStairTop =
      0.5 * widthStair * rightVec + vnormalize(upVec) + frontVec;
    #local iStair = 0;
    #while (iStair < nbStair)
      #local jStair = 0;
      #while (jStair < widthStair)
        superellipsoid {
          <blockRoundness, blockRoundness>
          scale 0.5
          translate 0.5
          scale vnormalize(rightVec) + upVec * stairThickness + frontVec *
              (1.0 + blockRoundness)
          translate upVec * (1.0 - stairThickness)
          translate rightVec * (jStair - 0.5 * widthStair)
          translate (frontVec + upVec) * iStair
```

```
        }
        #if (iStair > 0)
          superellipsoid {
            <blockRoundness, blockRoundness>
            scale 0.5
            translate 0.5
            scale vnormalize(rightVec) + upVec * (1.0 + stairThickness) +
                frontVec * stairThickness
            translate -1.0 * upVec * stairThickness
            translate rightVec * (jStair - 0.5 * widthStair)
            translate (frontVec + upVec) * iStair
          }
        #end
        #declare jStair = jStair + 1;
      #end
      //box {
      //  oneStairBottom
      //  oneStairTop
      //  translate (frontVec + upVec) * iStair
      //}
      #declare iStair = iStair + 1;
    #end
    translate startPos
    BlockTex()
  }
#end
```

XL-51.pov:

```
// ------ Includes ------

// Include the macros relative to blocks

#include "blocks.inc"

// Rendering parameters

background { color rgb 1.0 }
#include "rad_def.inc"
global_settings {
   radiosity {
      Rad_Settings(Radiosity_Normal,off,off)
      error_bound 2.0
      count 200 //100
   }
}
#default {finish{ambient 0.}}
#declare nbQuarter = 4;
declare lightIntensity = 1.0 / (nbQuarter * 3);

// ------ Scene elements' dimensions ------

// Unit is one block size

// Scale of one block

#declare scaleBlock = <1.0, 0.5, 0.5>;

// Measured platforms' and stairs' dimensions

#declare widthStairsA = 4;
```

```
#declare widthStairsB = 4;
#declare nbStairsB = 8;
#declare slopeFrontStairsB = 1;

#declare widthStairsC = 6;
#declare nbStairsC = 9;
#declare slopeUpStairsC = 1;
#declare slopeFrontStairsC = 1;

#declare nbStairsD = 12;
#declare widthStairsD = 6;
#declare slopeUpStairsD = 1;
#declare slopeFrontStairsD = 1;

#declare nbStairsE = 15;
#declare widthStairsE = 6;
#declare slopeUpStairsE = 1;
#declare slopeFrontStairsE = 1;

#declare widthPlatformC = 6;

#declare widthPlatformD = 4;

#declare heightDoorA = 11;
#declare widthDoorA = 7;

#declare heightDoorB = 8;
#declare widthDoorB = 6;

#declare heightDoorC = 9;
#declare widthDoorC = 6;

#declare heightDoorD = 6;
#declare widthDoorD = 4;

#declare widthDoorE = 19;
#declare heightDoorE = 18;

#declare widthDoorG = 4;
#declare heightDoorG = 2;

// Empirically deducted or modified dimensions to correct
// inconsistencies

#declare lengthPlatformC = 44;
#declare nbStairsA = 12;

// Deducted platforms' and stairs' dimensions

#declare widthRoom = widthPlatformD + nbStairsE + widthPlatformC;

#declare heightPlatformD = nbStairsD;
#declare lengthPlatformD = widthStairsD + widthStairsE;

#declare posStairsD = <
  widthPlatformD + nbStairsD,
  0,
  widthStairsE + widthStairsD / 2
>;

#declare posStairsE = <
```

```
  widthPlatformD,
  nbStairsD + 1,
  widthStairsE / 2
>;

#declare heightPlatformC = nbStairsD + nbStairsE;

#declare heightGapPlatformBPlatformC = heightDoorE + 6;

#declare posStairsA = <
    widthRoom - widthPlatformC,
    heightPlatformC,
    lengthPlatformC + widthStairsA / 2
>;

#declare widthPlatformB = widthStairsC + 2;
#declare heightPlatformB = lengthPlatformC + widthStairsA + 3;
#declare lengthPlatformB = heightPlatformC + nbStairsC + widthStairsB + 2 +
    4;

#declare lengthRoom = heightPlatformB + heightPlatformC +
    heightGapPlatformBPlatformC + 2;

#declare posStairsC = <
    widthRoom - widthStairsC / 2,
    heightPlatformC,
    lengthPlatformC + widthStairsA
>;

#declare widthPlatformA = widthPlatformB + 2;
#declare lengthPlatformA = heightPlatformC + nbStairsC;
#declare heightPlatformA = lengthPlatformB;

#declare heightGapPlatformAPlatformB = 6;

#declare slopeUpStairsA = (heightPlatformA - heightPlatformC) / nbStairsA;
#declare slopeFrontStairsA = (widthRoom - widthPlatformA - widthPlatformC +
    3) / nbStairsA;

#declare posStairsB = <
    widthPlatformA,
    heightPlatformA,
    lengthPlatformA - widthStairsB / 2 + 5
>;

#declare slopeUpStairsB = heightGapPlatformAPlatformB / nbStairsB;

// ------- Side walls definition -------

#declare SideWallRight = union {
  MakeWall(
    <
      -1,
      0,
      0
    >,
    <
      0,
      (lengthRoom - 1) / 2,
      3
    >)
  MakeWall(
```

```
    <
      -1,
      0,
      3
    >,
    <
      0,
      heightPlatformD,
      3 + widthDoorB
    >)
  MakeWall(
    <
      -1,
      heightPlatformD + 1 + heightDoorB,
      3
    >,
    <
      0,
      (lengthRoom - 1) / 2,
      3 + widthDoorB
    >)
  MakeWall(
    <
      -1,
      0,
      3 + widthDoorB
    >,
    <
      0,
      (lengthRoom - 1) / 2,
      (lengthRoom + 1) / 2
    >)
  scale scaleBlock
}

#declare SideWallUp = union {
  difference {
    MakeWall(
      <
        -1,
        heightDoorA - 4,
        -1
      >,
      <
        widthRoom,
        (lengthRoom - 2) / 2 - 4,
        0
      >)
    cylinder {
      <0, 0, -10>, <0, 0, 10>, 1
      scale <widthDoorA / 2, 4, 1>
      translate <
        widthRoom - 2 - widthDoorA / 2,
        heightDoorA - 4,
        0>
    }
  }
  MakeWall(
    <
      -1,
      (lengthRoom - 2) / 2 - 4,
      -1
```

```
    >,
    <
      3,
      (lengthRoom - 2) / 2,
      0
    >)
  MakeWall(
    <
      3 + widthDoorC,
      (lengthRoom - 2) / 2 - 4,
      -1
    >,
    <
      widthRoom + 1,
      (lengthRoom - 2) / 2,
      0
    >)
  MakeWall(
    <
      0,
      0,
      -1
    >,
    <
      widthRoom - 2 - widthDoorA,
      heightDoorA - 4,
      0
    >)
  MakeWall(
    <
      widthRoom - 2,
      0,
      -1
    >,
    <
      widthRoom + 1,
      heightDoorA - 4,
      0
    >)
  scale scaleBlock
}

#declare SideWallDown = union {
  MakeWall(
    <
      -1,
      -1,
      -lengthPlatformD * 2
    >,
    <
      3,
      0,
      lengthRoom / 2 + 1
    >)
  difference {
    MakeWall(
      <
        3,
        -1,
        -lengthPlatformD * 2
      >,
      <
```

```
      3 + widthDoorD,
      0,
      lengthRoom / 2 - heightDoorD + 2
    >)
  cylinder {
    <0, -10, 0> <0, 10, 0> 1
    scale <widthDoorD / 2, 1, 2>
    translate <
      3 + widthDoorD / 2,
      0,
      lengthRoom / 2 - heightDoorD + 2>
  }
}
MakeWall(
  <
    3,
    -1,
    lengthRoom / 2
  >,
  <
    3 + widthDoorD,
    0,
    lengthRoom / 2 + 1
  >)
MakeWall(
  <
    3 + widthDoorD,
    -1,
    -lengthPlatformD * 2
  >,
  <
    widthRoom - 3 - widthDoorC,
    0,
    lengthRoom / 2 + 1
  >)
difference {
  MakeWall(
    <
      widthRoom - 3 - widthDoorC,
      -1,
      -lengthPlatformD * 2
    >,
    <
      widthRoom - 3,
      0,
      lengthRoom / 2 - (heightDoorC - 5) + 2
    >)
  cylinder {
    <0, -10, 0> <0, 10, 0> 1
    scale <widthDoorC / 2, 1, 2>
    translate <
      widthRoom - 3 - widthDoorC / 2,
      0,
      lengthRoom / 2 - (heightDoorC - 5) + 2>
  }
}
MakeWall(
  <
    widthRoom - 3,
    -1,
    -lengthPlatformD * 2
  >,
```

```
    <
      widthRoom + 1,
      0,
      lengthRoom / 2 + 1
    >)
  scale scaleBlock
}

#declare SideWallLeft = union {
  MakeWall(
    <
      widthRoom,
      0,
      0
    >,
    <
      widthRoom + 1,
      (lengthRoom - 1) / 2,
      19
    >)
  MakeWall(
    <
      widthRoom,
      0,
      19
    >,
    <
      widthRoom + 1,
      heightPlatformC,
      19 + widthDoorE
    >)
  MakeWall(
    <
      widthRoom,
      heightPlatformC + 1 + heightDoorE,
      19
    >,
    <
      widthRoom + 1,
      (lengthRoom - 1) / 2,
      19 + widthDoorE
    >)
  MakeWall(
    <
      widthRoom + 1,
      heightPlatformC + 1,
      18
    >,
    <
      widthRoom + 6,
      heightPlatformC + 1 + heightDoorE,
      19
    >)
  MakeWall(
    <
      widthRoom + 1,
      heightPlatformC + 1,
      19 + widthDoorE
    >,
    <
      widthRoom + 6,
      heightPlatformC + 1 + heightDoorE,
```

```
    19 + widthDoorE + 1
  >)
MakeWall(
  <
    widthRoom + 17,
    heightPlatformC + 1,
    19 - 8
  >,
  <
    widthRoom + 18,
    heightPlatformC + 1 + heightDoorE,
    19 + widthDoorE + 8
  >)
MakeWall(
  <
    widthRoom,
    0,
    19 + widthDoorE
  >,
  <
    widthRoom + 1,
    (lengthRoom - 1) / 2,
    19 + widthDoorE + 6
  >)
MakeWall(
  <
    widthRoom,
    0,
    19 + widthDoorE + 6
  >,
  <
    widthRoom + 1,
    heightPlatformC + 1 - heightDoorG,
    19 + widthDoorE + 6 + widthDoorG
  >)
MakeWall(
  <
    widthRoom,
    heightPlatformC + 1 + heightDoorG,
    19 + widthDoorE + 6
  >,
  <
    widthRoom + 1,
    (lengthRoom - 1) / 2,
    19 + widthDoorE + 6 + widthDoorG
  >)
MakeWall(
  <
    widthRoom + 1,
    heightPlatformC + 1 - heightDoorG,
    19 + widthDoorE + 5
  >,
  <
    widthRoom + 5,
    heightPlatformC + 1 + heightDoorG,
    19 + widthDoorE + 6
  >)
MakeWall(
  <
    widthRoom + 1,
    heightPlatformC + 1 - heightDoorG,
    19 + widthDoorE + 6 + widthDoorG
```

```
    >,
    <
      widthRoom + 5,
      heightPlatformC + 1 + heightDoorG,
      19 + widthDoorE + 6 + widthDoorG + 1
    >)
  MakeWall(
    <
      widthRoom + 1,
      heightPlatformC + 1 - heightDoorG - 1,
      19 + widthDoorE + 6
    >,
    <
      widthRoom + 5,
      heightPlatformC + 1 - heightDoorG,
      19 + widthDoorE + 6 + widthDoorG
    >)
  MakeWall(
    <
      widthRoom,
      0,
      19 + widthDoorE + 6 + widthDoorG
    >,
    <
      widthRoom + 1,
      (lengthRoom - 1) / 2,
      (lengthRoom + 1) / 2
    >)
  scale scaleBlock
}

#declare SideWalls = union {
  object { SideWallRight }
  object { SideWallUp }
  object { SideWallDown }
  object { SideWallLeft }
}

// ------- Platforms definition -------

#declare PlatformA = union {
  MakeWall(
    <
      0,
      heightPlatformA,
      -20
    >,
    <
      widthPlatformA,
      heightPlatformA + 1,
      lengthPlatformA
    >)
  MakeWall(
    <
      widthPlatformA - 4,
      heightPlatformA,
      lengthPlatformA
    >,
    <
      widthPlatformA,
      heightPlatformA + 1,
      lengthPlatformA + 2
```

```
  >)
MakeWall(
  <
    0,
    heightPlatformA,
    lengthPlatformA
  >,
  <
    3,
    heightPlatformA + 1,
    lengthPlatformA + 2
  >)
difference {
  MakeWall(
    <
      0,
      heightPlatformA,
      lengthPlatformA + 2
    >,
    <
      widthPlatformA,
      heightPlatformA + 1,
      lengthPlatformA + 4
    >)
  cylinder {
    <0, -10, 0> <0, 10, 0>, 1
    scale <1.5, 1, 1>
    translate <4.5, heightPlatformA, lengthPlatformA + 2>
  }
}
MakeWall(
  <
    0,
    heightPlatformA,
    lengthPlatformA + 4
  >,
  <
    widthPlatformA,
    heightPlatformA + 1,
    lengthPlatformA + 5
  >)
MakeWall(
  <
    0,
    heightPlatformA,
    lengthPlatformA + 5
  >,
  <
    widthPlatformB,
    heightPlatformA + 1,
    lengthPlatformA + 8
  >)
MakeWall(
  <
    0,
    heightPlatformA,
    lengthPlatformA + 8
  >,
  <
    widthPlatformB - 1,
    heightPlatformA + 1,
    lengthPlatformA + 12
```

```
    >)
  MakeWall(
    <
      0,
      heightPlatformA,
      lengthPlatformA + 12
    >,
    <
      widthPlatformB,
      heightPlatformA + 1,
      lengthPlatformA + 16
    >)
  MakeWall(
    <
      0,
      heightPlatformA - widthStairsC / 2,
      lengthPlatformA - 1
    >,
    <
      widthStairsC,
      heightPlatformA,
      lengthPlatformA
    >)
  object {
    Block()
    scale <1, 1, 0.5>
    translate <
      widthPlatformB - 0.5,
      heightPlatformA + 0.5,
      lengthPlatformA + 8.25>
  }
  object {
    Block()
    scale <1, 1, 0.5>
    translate <
      widthPlatformB - 0.5,
      heightPlatformA + 0.5,
      lengthPlatformA + 11.75>
  }
  scale scaleBlock
}

#declare PlatformB = union {
  MakeWall(
    <
      0,
      0,
      heightPlatformB
    >,
    <
      widthPlatformB,
      lengthPlatformB,
      heightPlatformB + 1
    >)
  scale scaleBlock
}

#declare PlatformC = union {
  MakeWall(
    <
      widthRoom - widthPlatformC,
      heightPlatformC,
```

```
      -10
    >,
    <
      widthRoom + 30,
      heightPlatformC + 1,
      lengthPlatformC
    >)
  MakeWall(
    <
      widthRoom - widthPlatformC,
      heightPlatformC,
      lengthPlatformC
    >,
    <
      widthRoom - 3,
      heightPlatformC + 1,
      lengthPlatformC + widthStairsA
    >)
  MakeWall(
    <
      widthRoom - 3,
      heightPlatformC - 3,
      lengthPlatformC + widthStairsA
    >,
    <
      widthRoom,
      heightPlatformC,
      lengthPlatformC + widthStairsA + 1
    >)
  scale scaleBlock
}

#declare PlatformD = union {
  MakeWall(
    <
      -widthPlatformD,
      heightPlatformD,
      -lengthPlatformD
    >,
    <
      widthPlatformD,
      heightPlatformD + 1,
      lengthPlatformD
    >)
  scale scaleBlock
}

#declare Platforms = union {
  object { PlatformA }
  object { PlatformB }
  object { PlatformC }
  object { PlatformD }
}

// ------ Stairs definition ------

#declare StairsA = object {
  MakeStairs(
     widthStairsA,
     nbStairsA,
     posStairsA - y * 0.6,
     y * slopeUpStairsA * 1.03,
```

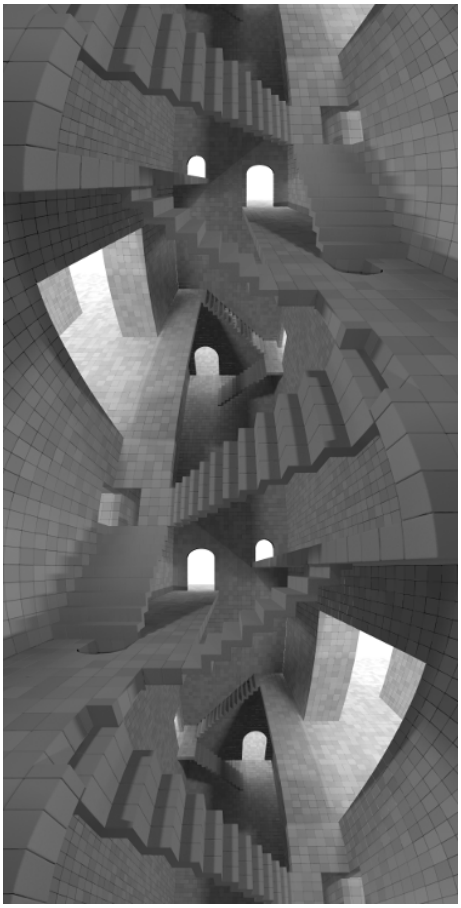
```
     z,
    -x * slopeFrontStairsA * 1.0)
  scale scaleBlock
}

#declare StairsB = object {
  MakeStairs(
    widthStairsB,
    nbStairsB,
    posStairsB + y * 0.25,
    y * slopeUpStairsB * 0.97,
    z,
    x * slopeFrontStairsB * 1.02)
  scale scaleBlock
}

#declare StairsC = object {
  MakeStairs(
    widthStairsC,
    nbStairsC,
    posStairsC,
    y * slopeUpStairsC,
    x,
    z * slopeFrontStairsC)
  scale scaleBlock
}

#declare StairsD = object {
  MakeStairs(
     widthStairsD,
     nbStairsD,
     posStairsD,
     y * slopeUpStairsD,
     z,
    -x * slopeFrontStairsD)
  scale scaleBlock
}

#declare StairsE = object {
  MakeStairs(
    widthStairsE,
    nbStairsE,
    posStairsE,
    y * slopeUpStairsE,
    z,
    x * slopeFrontStairsE)
  scale scaleBlock
}

#declare Stairs = union {
  object { StairsA }
  object { StairsB }
  object { StairsC }
  object { StairsD }
  object { StairsE }
  texture {pigment {color rgb 1.0}}
}

// ------ House of Stairs definition ------

#declare HouseOfStairs = union {
```

```
  // Loop on the four quarters of the house
  #declare iQuarter = 0;
  #while (iQuarter < nbQuarter)
    union {

      // Elements of one quarter
      object { SideWalls }
      object { Platforms }
      object { Stairs }

      // Rotate the quarter along the horizontal axis
      translate <0, -lengthRoom / 2, -lengthRoom / 2> * scaleBlock
      rotate x * 90.0 * iQuarter
      translate <0, lengthRoom / 2, lengthRoom / 2> * scaleBlock

      // Apply the symmetry to the odd quarters
      #if (iQuarter = 1 | iQuarter = 3)
        scale <-1.0, 1.0, 1.0>
        translate widthRoom * x * scaleBlock
      #end
    }
    #declare iQuarter = iQuarter + 1;
  #end
}

// ------ Camera ------

// Move the camera by epsilon to avoid aligning with the interstice between
    block
#declare posCamera = <widthRoom / 2, lengthRoom / 2, lengthRoom / 2> *
    scaleBlock + 0.1 * y + 0.1 * z;
#declare lookAt = <widthRoom / 2, 0, 0.25 * lengthRoom> * scaleBlock;

//#declare posCamera = (posStairsA - 3 + 3 * y) * scaleBlock;
//#declare lookAt = posStairsA * scaleBlock;

camera {
  cylinder 2
  angle 210
  location posCamera
  look_at lookAt
  right x * 3.8
  up y
}

// ------ Light ------

/*light_source {
  posCamera
  color rgb 1.0
}*/

#declare Lights = union {

  // Loop on the four quarters of the house
  #declare iQuarter = 0;
  #while (iQuarter < nbQuarter)

    // Lights
    light_source {
      <widthRoom - 2 - widthDoorA / 2 - 1, heightDoorA, -5>
      color rgb 1.0 * lightIntensity
```

```
  area_light <2, 0, 0> <0, 2, 0> 4, 4 adaptive 0 jitter

  // Rotate the quarter along the horizontal axis
  translate <0, -lengthRoom / 2, -lengthRoom / 2> * scaleBlock
  rotate x * 90.0 * iQuarter
  translate <0, lengthRoom / 2, lengthRoom / 2> * scaleBlock

  // Apply the symmetry to the odd quarters
  #if (iQuarter = 1 | iQuarter = 3)
    scale <-1.0, 1.0, 1.0>
    translate widthRoom * x * scaleBlock
  #end
  //scale scaleBlock
}

light_source {
  <-5, heightPlatformD + 1 + heightDoorB, 3 + widthDoorB / 2>
  color rgb 1.0 * lightIntensity
  area_light <0, 2, 0> <0, 0, 2> 4, 4 adaptive 0 jitter

  // Rotate the quarter along the horizontal axis
  translate <0, -lengthRoom / 2, -lengthRoom / 2> * scaleBlock
  rotate x * 90.0 * iQuarter
  translate <0, lengthRoom / 2, lengthRoom / 2> * scaleBlock

  // Apply the symmetry to the odd quarters
  #if (iQuarter = 1 | iQuarter = 3)
    scale <-1.0, 1.0, 1.0>
    translate widthRoom * x * scaleBlock
  #end
  //scale scaleBlock
}

light_source {
  <3 + widthDoorC / 2, heightPlatformA + 1 + heightDoorC, -5>
  color rgb 1.0 * lightIntensity
  area_light <2, 0, 0> <0, 2, 0> 4, 4 adaptive 0 jitter

  // Rotate the quarter along the horizontal axis
  translate <0, -lengthRoom / 2, -lengthRoom / 2> * scaleBlock
  rotate x * 90.0 * iQuarter
  translate <0, lengthRoom / 2, lengthRoom / 2> * scaleBlock

  // Apply the symmetry to the odd quarters
  #if (iQuarter = 1 | iQuarter = 3)
    scale <-1.0, 1.0, 1.0>
    translate widthRoom * x * scaleBlock
  #end
  //scale scaleBlock
}

light_source {
  <widthRoom + 2, heightPlatformC + 1 + heightDoorE - 1, 19 + widthDoorE
       / 2> * scaleBlock
  color rgb 1.0 * lightIntensity
  area_light <2, 0, 0> <0, 0, 2> 4, 4 adaptive 0 jitter

  // Rotate the quarter along the horizontal axis
  translate <0, -lengthRoom / 2, -lengthRoom / 2> * scaleBlock
  rotate x * 90.0 * iQuarter
  translate <0, lengthRoom / 2, lengthRoom / 2> * scaleBlock
```

```
      // Apply the symmetry to the odd quarters
      #if (iQuarter = 1 | iQuarter = 3)
        scale <-1.0, 1.0, 1.0>
        translate widthRoom * x * scaleBlock
      #end
      //scale scaleBlock
    }

    #declare iQuarter = iQuarter + 1;
  #end

  /*light_source {
    <widthRoom / 2, lengthRoom / 2, lengthRoom / 2>
    color rgb 0.1
  }*/

}

object { Lights }

// ------ Scene ------

object {
  HouseOfStairs
}

// ------ Frame ------

#declare QuarterFrame = union {
  #local r = 0.1;
  cylinder {
    <0, 0, 0>,
    <widthRoom, 0, 0>,
    r
  }
  cylinder {
    <0, 0, 0>,
    <0, lengthRoom / 2, 0>,
    r
  }
  cylinder {
    <0, 0, 0>,
    <0, 0, lengthRoom / 2>,
    r
  }
  cylinder {
    <widthRoom, 0, 0>,
    <widthRoom, lengthRoom / 2, 0>,
    r
  }
  cylinder {
    <widthRoom, 0, 0>,
    <widthRoom, 0, lengthRoom / 2>,
    r
  }
  cylinder {
    <0, lengthRoom / 2, 0>,
    <widthRoom, lengthRoom / 2, 0>,
    r
  }
  cylinder {
    <0, 0, lengthRoom / 2>,
```

```
    <widthRoom, 0, lengthRoom / 2>,
    r
  }
  cylinder {
    <0, lengthRoom / 2, 0>,
    <0, lengthRoom / 2, lengthRoom / 2>,
    r
  }
  cylinder {
    <0, 0, lengthRoom / 2>,
    <0, lengthRoom / 2, lengthRoom / 2>,
    r
  }
  cylinder {
    <widthRoom, lengthRoom / 2, 0>,
    <widthRoom, lengthRoom / 2, lengthRoom / 2>,
    r
  }
  cylinder {
    <widthRoom, 0, lengthRoom / 2>,
    <widthRoom, lengthRoom / 2, lengthRoom / 2>,
    r
  }
  texture {pigment {color rgb x}}
  scale scaleBlock
}

//object { QuarterFrame }
```



By adding lights behind the doors of the front walls and one in the door E on the side walls I get something pretty satisfying but too dark. There are also some problem due to radiosity on the stairs of the up wall, a problem in the texture of the round portion of the wall, and problems due to spaces between blocks (as I randomize slightly there shape for a more beautifull result.

The result is also quite different from the lithography but I give up on that as the lighting in Escher's work is definitely unrealistic. So I concentrate on the problems above and a solution that pleases me rather than replicates the lithography.

blocks.inc:

```
// Seed for the randomization of blocks' texture and size

#declare seedBlock = seed(0);
```

```
// Roundness coefficient of the blocks

#declare blockRoundness = 0.075;

// Randomness coefficient for the block dimension

#declare blockRandomness = 0.0; //0.025;

// Thickness coefficient of the stairs

#declare stairThickness = 1.0 / 2.0;

// Return a random value in range [1.0 - r / 2, 1.0 + r / 2]

#macro RandAroungOne(r)
  (1.0 + (0.5 - rand(seedBlock)) * r)
#end

// Return the texture of the blocks

#macro BlockTex()
  texture {
    pigment {
      color rgb (1.0 - blockRandomness) * RandAroungOne(0.25)
    }
    normal {
      bumps 0.25 scale .02
    }
    finish {
      ambient 0.0 diffuse 1.0
    }
  }
#end

// Return the scaling for the randomization of the size of the blocks

#macro BlockRandScale()
  0.5 * RandAroungOne(blockRandomness)
#end

// Return one block

#macro Block()
  superellipsoid {
    <blockRoundness, blockRoundness>
    BlockTex()
    scale BlockRandScale()
  }
#end

// Make the wall equivalent to box {posMin, posMax} made of blocks

#macro MakeWallSimple(
    posMin,
    posMax)
  box {
    posMin, posMax
    BlockTex()
    //texture { pigment {color rgb 1.0 } }
  }
#end
```

```
#macro MakeWall(
    posMin,
    posMax)
  union {
    #local wallX = posMin.x;
    #while (wallX < posMax.x)
      #local wallY = posMin.y;
      #while (wallY < posMax.y)
        #local wallZ = posMin.z;
        #while (wallZ < posMax.z)
          object {
            Block()
            translate <wallX + 0.5, wallY + 0.5, wallZ + 0.5>
          }
          #declare wallZ = wallZ + 1.0;
        #end
        #declare wallY = wallY + 1.0;
      #end
      #declare wallX = wallX + 1.0;
    #end
  }
#end

// Return one stair

#macro MakeStairsBox(
    widthStair,
    nbStair,
    startPos,
    upVec,
    rightVec,
    frontVec)
  union {
    #local oneStairBottom =
      -0.5 * widthStair * rightVec;
    #local oneStairTop =
      0.5 * widthStair * rightVec + upVec + frontVec;
    #local iStair = 0;
    #while (iStair < nbStair)
      box {
        oneStairBottom
        oneStairTop
        translate (frontVec + upVec) * iStair
      }
      #declare iStair = iStair + 1;
    #end
    translate startPos
    BlockTex()
    //texture { pigment {color rgb 1.0 } }
  }
#end

#macro MakeStairs(
    widthStair,
    nbStair,
    startPos,
    upVec,
    rightVec,
    frontVec)
  union {
    #local oneStairBottom =
```

```
      -0.5 * widthStair * rightVec + (1.0 - stairThickness) * vnormalize(
          upVec);
    #local oneStairTop =
      0.5 * widthStair * rightVec + vnormalize(upVec) + frontVec;
    #local iStair = 0;
    #while (iStair < nbStair)
      #local jStair = 0;
      #while (jStair < widthStair)
        superellipsoid {
          <blockRoundness, blockRoundness>
          scale 0.5
          translate 0.5
          scale vnormalize(rightVec) + upVec * stairThickness + frontVec *
              (1.0 + blockRoundness)
          translate upVec * (1.0 - stairThickness)
          translate rightVec * (jStair - 0.5 * widthStair)
          translate (frontVec + upVec) * iStair
        }
        #if (iStair > 0)
          superellipsoid {
            <blockRoundness, blockRoundness>
            scale 0.5
            translate 0.5
            scale vnormalize(rightVec) + upVec * (1.0 + stairThickness) +
                frontVec * stairThickness
            translate -1.0 * upVec * stairThickness
            translate rightVec * (jStair - 0.5 * widthStair)
            translate (frontVec + upVec) * iStair
          }
        #end
        #declare jStair = jStair + 1;
      #end
      //box {
      //  oneStairBottom
      //  oneStairTop
      //  translate (frontVec + upVec) * iStair
      //}
      #declare iStair = iStair + 1;
    #end
    translate startPos
    BlockTex()
  }
#end
```

XL-51.pov:

```
// ------ Includes ------

// Include the macros relative to blocks

#include "blocks.inc"

// Rendering parameters

background { color rgb 1.0 }
#include "rad_def.inc"
global_settings {
   radiosity {
      Rad_Settings(Radiosity_Normal,off,off)
      error_bound 2.0
      count 300 //200
```

```
   }
}
#default {finish{ambient 0.}}
#declare nbQuarter = 4;
declare lightIntensity = 1.0 / (nbQuarter * 3);

// ------ Scene elements' dimensions ------

// Unit is one block size

// Scale of one block

#declare scaleBlock = <1.0, 0.5, 0.5>;

// Measured platforms' and stairs' dimensions

#declare widthStairsA = 4;

#declare widthStairsB = 4;
#declare nbStairsB = 8;
#declare slopeFrontStairsB = 1;

#declare widthStairsC = 6;
#declare nbStairsC = 9;
#declare slopeUpStairsC = 1;
#declare slopeFrontStairsC = 1;

#declare nbStairsD = 12;
#declare widthStairsD = 6;
#declare slopeUpStairsD = 1;
#declare slopeFrontStairsD = 1;

#declare nbStairsE = 15;
#declare widthStairsE = 6;
#declare slopeUpStairsE = 1;
#declare slopeFrontStairsE = 1;

#declare widthPlatformC = 6;

#declare widthPlatformD = 4;

#declare heightDoorA = 11;
#declare widthDoorA = 7;

#declare heightDoorB = 8;
#declare widthDoorB = 6;

#declare heightDoorC = 9;
#declare widthDoorC = 6;

#declare heightDoorD = 6;
#declare widthDoorD = 4;

#declare widthDoorE = 19;
#declare heightDoorE = 18;

#declare widthDoorG = 4;
#declare heightDoorG = 2;

// Empirically deducted or modified dimensions to correct
// inconsistencies
```

```
#declare lengthPlatformC = 44;
#declare nbStairsA = 12;

// Deducted platforms' and stairs' dimensions

#declare widthRoom = widthPlatformD + nbStairsE + widthPlatformC;

#declare heightPlatformD = nbStairsD;
#declare lengthPlatformD = widthStairsD + widthStairsE;

#declare posStairsD = <
  widthPlatformD + nbStairsD,
  0,
  widthStairsE + widthStairsD / 2
>;

#declare posStairsE = <
  widthPlatformD,
  nbStairsD + 1,
  widthStairsE / 2
>;

#declare heightPlatformC = nbStairsD + nbStairsE;

#declare heightGapPlatformBPlatformC = heightDoorE + 6;

#declare posStairsA = <
    widthRoom - widthPlatformC,
    heightPlatformC,
    lengthPlatformC + widthStairsA / 2
>;

#declare widthPlatformB = widthStairsC + 2;
#declare heightPlatformB = lengthPlatformC + widthStairsA + 3;
#declare lengthPlatformB = heightPlatformC + nbStairsC + widthStairsB + 2 +
    4;

#declare lengthRoom = heightPlatformB + heightPlatformC +
    heightGapPlatformBPlatformC + 2;

#declare posStairsC = <
    widthRoom - widthStairsC / 2,
    heightPlatformC,
    lengthPlatformC + widthStairsA
>;

#declare widthPlatformA = widthPlatformB + 2;
#declare lengthPlatformA = heightPlatformC + nbStairsC;
#declare heightPlatformA = lengthPlatformB;

#declare heightGapPlatformAPlatformB = 6;

#declare slopeUpStairsA = (heightPlatformA - heightPlatformC) / nbStairsA;
#declare slopeFrontStairsA = (widthRoom - widthPlatformA - widthPlatformC +
    3) / nbStairsA;

#declare posStairsB = <
    widthPlatformA,
    heightPlatformA,
    lengthPlatformA - widthStairsB / 2 + 5
>;
```

```
#declare slopeUpStairsB = heightGapPlatformAPlatformB / nbStairsB;

// ------- Side walls definition -------

#declare SideWallRight = union {
  MakeWall(
    <
      -1,
      0,
      0
    >,
    <
      0,
      (lengthRoom - 1) / 2,
      3
    >)
  MakeWall(
    <
      -1,
      0,
      3
    >,
    <
      0,
      heightPlatformD,
      3 + widthDoorB
    >)
  MakeWall(
    <
      -1,
      heightPlatformD + 1 + heightDoorB,
      3
    >,
    <
      0,
      (lengthRoom - 1) / 2,
      3 + widthDoorB
    >)
  MakeWall(
    <
      -1,
      0,
      3 + widthDoorB
    >,
    <
      0,
      (lengthRoom - 1) / 2,
      (lengthRoom + 1) / 2
    >)
  scale scaleBlock
}

#declare SideWallUp = union {
  difference {
    MakeWall(
      <
        -1,
        heightDoorA - 4,
        -1
      >,
      <
        widthRoom + 1,
```

```
      (lengthRoom - 2) / 2 - 4,
      0
    >)
  cylinder {
    <0, 0, -10>, <0, 0, 10>, 1
    scale <widthDoorA / 2, 4, 1>
    translate <
      widthRoom - 2 - widthDoorA / 2,
      heightDoorA - 4,
      0>
    BlockTex()
  }
}
MakeWall(
  <
    -1,
    (lengthRoom - 2) / 2 - 4,
    -1
  >,
  <
    3,
    (lengthRoom - 2) / 2,
    0
  >)
MakeWall(
  <
    3 + widthDoorC,
    (lengthRoom - 2) / 2 - 4,
    -1
  >,
  <
    widthRoom + 1,
    (lengthRoom - 2) / 2,
    0
  >)
MakeWall(
  <
    0,
    0,
    -1
  >,
  <
    widthRoom - 2 - widthDoorA,
    heightDoorA - 4,
    0
  >)
MakeWall(
  <
    widthRoom - 2,
    0,
    -1
  >,
  <
    widthRoom + 1,
    heightDoorA - 4,
    0
  >)
scale scaleBlock
}

#declare SideWallDown = union {
  MakeWall(
```

```
  <
    -1,
    -1,
    -lengthPlatformD * 2
  >,
  <
    3,
    0,
    lengthRoom / 2 + 1
  >)
difference {
  MakeWall(
    <
      3,
      -1,
      -lengthPlatformD * 2
    >,
    <
      3 + widthDoorD,
      0,
      lengthRoom / 2 - heightDoorD + 2
    >)
  cylinder {
    <0, -10, 0> <0, 10, 0> 1
    scale <widthDoorD / 2, 1, 2>
    translate <
      3 + widthDoorD / 2,
      0,
      lengthRoom / 2 - heightDoorD + 2>
    BlockTex()
  }
}
MakeWall(
  <
    3,
    -1,
    lengthRoom / 2
  >,
  <
    3 + widthDoorD,
    0,
    lengthRoom / 2 + 1
  >)
MakeWall(
  <
    3 + widthDoorD,
    -1,
    -lengthPlatformD * 2
  >,
  <
    widthRoom - 3 - widthDoorC,
    0,
    lengthRoom / 2 + 1
  >)
difference {
  MakeWall(
    <
      widthRoom - 3 - widthDoorC,
      -1,
      -lengthPlatformD * 2
    >,
    <
```

```
        widthRoom - 3,
        0,
        lengthRoom / 2 - (heightDoorC - 5) + 2
      >)
    cylinder {
      <0, -10, 0> <0, 10, 0> 1
      scale <widthDoorC / 2, 1, 2>
      translate <
        widthRoom - 3 - widthDoorC / 2,
        0,
        lengthRoom / 2 - (heightDoorC - 5) + 2>
      BlockTex()
    }
  }
  MakeWall(
    <
      widthRoom - 3,
      -1,
      -lengthPlatformD * 2
    >,
    <
      widthRoom + 1,
      0,
      lengthRoom / 2 + 1
    >)
  scale scaleBlock
}

#declare SideWallLeft = union {
  MakeWall(
    <
      widthRoom,
      0,
      0
    >,
    <
      widthRoom + 1,
      (lengthRoom - 1) / 2,
      19
    >)
  MakeWall(
    <
      widthRoom,
      0,
      19
    >,
    <
      widthRoom + 1,
      heightPlatformC,
      19 + widthDoorE
    >)
  MakeWall(
    <
      widthRoom,
      heightPlatformC + 1 + heightDoorE,
      19
    >,
    <
      widthRoom + 1,
      (lengthRoom - 1) / 2,
      19 + widthDoorE
    >)
```

```
MakeWall(
  <
    widthRoom + 1,
    heightPlatformC + 1,
    18
  >,
  <
    widthRoom + 6,
    heightPlatformC + 1 + heightDoorE,
    19
  >)
MakeWall(
  <
    widthRoom + 1,
    heightPlatformC + 1,
    19 + widthDoorE
  >,
  <
    widthRoom + 6,
    heightPlatformC + 1 + heightDoorE,
    19 + widthDoorE + 1
  >)
MakeWall(
  <
    widthRoom + 17,
    heightPlatformC + 1,
    19 - 8
  >,
  <
    widthRoom + 18,
    heightPlatformC + 1 + heightDoorE,
    19 + widthDoorE + 8
  >)
MakeWall(
  <
    widthRoom,
    0,
    19 + widthDoorE
  >,
  <
    widthRoom + 1,
    (lengthRoom - 1) / 2,
    19 + widthDoorE + 6
  >)
MakeWall(
  <
    widthRoom,
    0,
    19 + widthDoorE + 6
  >,
  <
    widthRoom + 1,
    heightPlatformC + 1 - heightDoorG,
    19 + widthDoorE + 6 + widthDoorG
  >)
MakeWall(
  <
    widthRoom,
    heightPlatformC + 1 + heightDoorG,
    19 + widthDoorE + 6
  >,
  <
```

```
      widthRoom + 1,
      (lengthRoom - 1) / 2,
      19 + widthDoorE + 6 + widthDoorG
    >)
  MakeWall(
    <
      widthRoom + 1,
      heightPlatformC + 1 - heightDoorG,
      19 + widthDoorE + 5
    >,
    <
      widthRoom + 5,
      heightPlatformC + 1 + heightDoorG,
      19 + widthDoorE + 6
    >)
  MakeWall(
    <
      widthRoom + 1,
      heightPlatformC + 1 - heightDoorG,
      19 + widthDoorE + 6 + widthDoorG
    >,
    <
      widthRoom + 5,
      heightPlatformC + 1 + heightDoorG,
      19 + widthDoorE + 6 + widthDoorG + 1
    >)
  MakeWall(
    <
      widthRoom + 1,
      heightPlatformC + 1 - heightDoorG - 1,
      19 + widthDoorE + 6 - 1
    >,
    <
      widthRoom + 5,
      heightPlatformC + 1 - heightDoorG,
      19 + widthDoorE + 6 + widthDoorG + 1
    >)
  MakeWall(
    <
      widthRoom,
      0,
      19 + widthDoorE + 6 + widthDoorG
    >,
    <
      widthRoom + 1,
      (lengthRoom - 1) / 2,
      (lengthRoom + 1) / 2
    >)
  scale scaleBlock
}

#declare SideWalls = union {
  object { SideWallRight }
  object { SideWallUp }
  object { SideWallDown }
  object { SideWallLeft }
}

// ------- Platforms definition -------

#declare PlatformA = union {
  MakeWall(
```

```
  <
    0,
    heightPlatformA,
    -20
  >,
  <
    widthPlatformA,
    heightPlatformA + 1,
    lengthPlatformA
  >)
MakeWall(
  <
    widthPlatformA - 4,
    heightPlatformA,
    lengthPlatformA
  >,
  <
    widthPlatformA,
    heightPlatformA + 1,
    lengthPlatformA + 2
  >)
MakeWall(
  <
    0,
    heightPlatformA,
    lengthPlatformA
  >,
  <
    3,
    heightPlatformA + 1,
    lengthPlatformA + 2
  >)
difference {
  MakeWall(
    <
      0,
      heightPlatformA,
      lengthPlatformA + 2
    >,
    <
      widthPlatformA,
      heightPlatformA + 1,
      lengthPlatformA + 4
    >)
  cylinder {
    <0, -10, 0> <0, 10, 0>, 1
    scale <1.5, 1, 1>
    translate <4.5, heightPlatformA, lengthPlatformA + 2>
    BlockTex()
  }
}
MakeWall(
  <
    0,
    heightPlatformA,
    lengthPlatformA + 4
  >,
  <
    widthPlatformA,
    heightPlatformA + 1,
    lengthPlatformA + 5
  >)
```

```
  MakeWall(
    <
      0,
      heightPlatformA,
      lengthPlatformA + 5
    >,
    <
      widthPlatformB,
      heightPlatformA + 1,
      lengthPlatformA + 8
    >)
  MakeWall(
    <
      0,
      heightPlatformA,
      lengthPlatformA + 8
    >,
    <
      widthPlatformB - 1,
      heightPlatformA + 1,
      lengthPlatformA + 12
    >)
  MakeWall(
    <
      0,
      heightPlatformA,
      lengthPlatformA + 12
    >,
    <
      widthPlatformB,
      heightPlatformA + 1,
      lengthPlatformA + 16
    >)
  MakeWall(
    <
      0,
      heightPlatformA - widthStairsC / 2,
      lengthPlatformA - 1
    >,
    <
      widthStairsC,
      heightPlatformA,
      lengthPlatformA
    >)
  object {
    Block()
    scale <1, 1, 0.5>
    translate <
      widthPlatformB - 0.5,
      heightPlatformA + 0.5,
      lengthPlatformA + 8.25>
  }
  object {
    Block()
    scale <1, 1, 0.5>
    translate <
      widthPlatformB - 0.5,
      heightPlatformA + 0.5,
      lengthPlatformA + 11.75>
  }
  scale scaleBlock
}
```

```
#declare PlatformB = union {
  MakeWall(
    <
      0,
      0,
      heightPlatformB
    >,
    <
      widthPlatformB,
      lengthPlatformB,
      heightPlatformB + 1
    >)
  scale scaleBlock
}

#declare PlatformC = union {
  MakeWall(
    <
      widthRoom - widthPlatformC,
      heightPlatformC,
      -10
    >,
    <
      widthRoom + 30,
      heightPlatformC + 1,
      lengthPlatformC
    >)
  MakeWall(
    <
      widthRoom - widthPlatformC,
      heightPlatformC,
      lengthPlatformC
    >,
    <
      widthRoom - 3,
      heightPlatformC + 1,
      lengthPlatformC + widthStairsA
    >)
  MakeWall(
    <
      widthRoom - 3,
      heightPlatformC - 3,
      lengthPlatformC + widthStairsA
    >,
    <
      widthRoom,
      heightPlatformC,
      lengthPlatformC + widthStairsA + 1
    >)
  scale scaleBlock
}

#declare PlatformD = union {
  MakeWall(
    <
      -widthPlatformD,
      heightPlatformD,
      -lengthPlatformD
    >,
    <
      widthPlatformD,
```

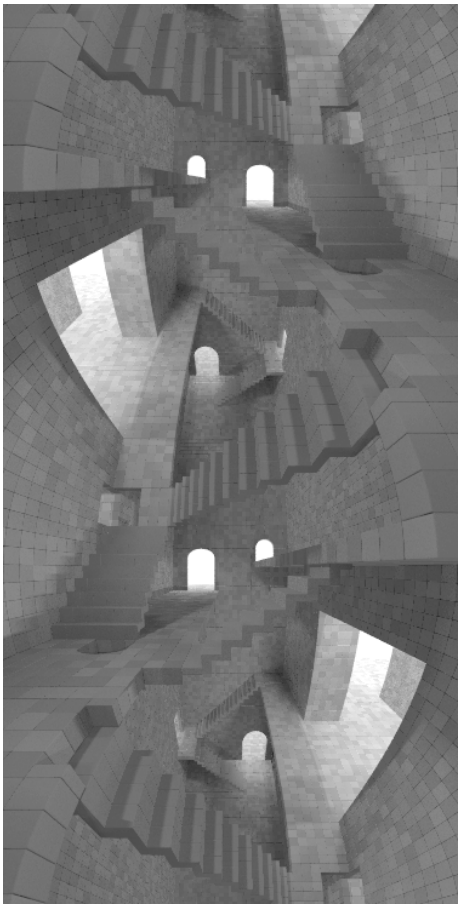
```
      heightPlatformD + 1,
      lengthPlatformD
    >)
  scale scaleBlock
}

#declare Platforms = union {
  object { PlatformA }
  object { PlatformB }
  object { PlatformC }
  object { PlatformD }
}

// ------ Stairs definition ------

#declare StairsA = object {
  MakeStairs(
     widthStairsA,
     nbStairsA,
     posStairsA - y * 0.6,
     y * slopeUpStairsA * 1.03,
     z,
    -x * slopeFrontStairsA * 1.0)
  scale scaleBlock
}

#declare StairsB = object {
  MakeStairs(
    widthStairsB,
    nbStairsB,
    posStairsB + y * 0.25,
    y * slopeUpStairsB * 0.97,
    z,
    x * slopeFrontStairsB * 1.02)
  scale scaleBlock
}

#declare StairsC = object {
  MakeStairs(
    widthStairsC,
    nbStairsC,
    posStairsC,
    y * slopeUpStairsC,
    x,
    z * slopeFrontStairsC)
  scale scaleBlock
}

#declare StairsD = object {
  MakeStairs(
     widthStairsD,
     nbStairsD,
     posStairsD,
     y * slopeUpStairsD,
     z,
    -x * slopeFrontStairsD)
  scale scaleBlock
}

#declare StairsE = object {
  MakeStairs(
    widthStairsE,
```

```
    nbStairsE,
    posStairsE,
    y * slopeUpStairsE,
    z,
    x * slopeFrontStairsE)
  scale scaleBlock
}

#declare Stairs = union {
  object { StairsA }
  object { StairsB }
  object { StairsC }
  object { StairsD }
  object { StairsE }
  texture {pigment {color rgb 1.0}}
}

// ------ House of Stairs definition ------

#declare HouseOfStairs = union {

  // Loop on the four quarters of the house
  #declare iQuarter = 0;
  #while (iQuarter < nbQuarter)
    union {

      // Elements of one quarter
      object { SideWalls }
      object { Platforms }
      object { Stairs }

      // Rotate the quarter along the horizontal axis
      translate <0, -lengthRoom / 2, -lengthRoom / 2> * scaleBlock
      rotate x * 90.0 * iQuarter
      translate <0, lengthRoom / 2, lengthRoom / 2> * scaleBlock

      // Apply the symmetry to the odd quarters
      #if (iQuarter = 1 | iQuarter = 3)
        scale <-1.0, 1.0, 1.0>
        translate widthRoom * x * scaleBlock
      #end
    }
    #declare iQuarter = iQuarter + 1;
  #end
}

// ------ Camera ------

// Move the camera by epsilon to avoid aligning with the interstice between
    block
#declare posCamera = <widthRoom / 2, lengthRoom / 2, lengthRoom / 2> *
    scaleBlock + 0.1 * y + 0.1 * z;
#declare lookAt = <widthRoom / 2, 0, 0.25 * lengthRoom> * scaleBlock;

//#declare posCamera = (posStairsA - 3 + 3 * y) * scaleBlock;
//#declare lookAt = posStairsA * scaleBlock;

camera {
  cylinder 2
  angle 210
  location posCamera
  look_at lookAt
```

```
  right x * 3.8
  up y
}

// ------ Light ------

/*light_source {
  posCamera
  color rgb 1.0
}*/

#declare Lights = union {

  // Loop on the four quarters of the house
  #declare iQuarter = 0;
  #while (iQuarter < nbQuarter)

    // Lights
    light_source {
      <widthRoom - 2 - widthDoorA / 2 - 1, heightDoorA, -5>
      color rgb 1.0 * lightIntensity
      area_light <2, 0, 0> <0, 2, 0> 4, 4 adaptive 0 jitter

      // Rotate the quarter along the horizontal axis
      translate <0, -lengthRoom / 2, -lengthRoom / 2> * scaleBlock
      rotate x * 90.0 * iQuarter
      translate <0, lengthRoom / 2, lengthRoom / 2> * scaleBlock

      // Apply the symmetry to the odd quarters
      #if (iQuarter = 1 | iQuarter = 3)
        scale <-1.0, 1.0, 1.0>
        translate widthRoom * x * scaleBlock
      #end
    }

    light_source {
      <-5, heightPlatformD + 1 + heightDoorB, 3 + widthDoorB / 2>
      color rgb 1.0 * lightIntensity
      area_light <0, 2, 0> <0, 0, 2> 4, 4 adaptive 0 jitter

      // Rotate the quarter along the horizontal axis
      translate <0, -lengthRoom / 2, -lengthRoom / 2> * scaleBlock
      rotate x * 90.0 * iQuarter
      translate <0, lengthRoom / 2, lengthRoom / 2> * scaleBlock

      // Apply the symmetry to the odd quarters
      #if (iQuarter = 1 | iQuarter = 3)
        scale <-1.0, 1.0, 1.0>
        translate widthRoom * x * scaleBlock
      #end
    }

    light_source {
      <3 + widthDoorC / 2, heightPlatformA + 1 + heightDoorC, -5>
      color rgb 1.0 * lightIntensity
      area_light <2, 0, 0> <0, 2, 0> 4, 4 adaptive 0 jitter

      // Rotate the quarter along the horizontal axis
      translate <0, -lengthRoom / 2, -lengthRoom / 2> * scaleBlock
      rotate x * 90.0 * iQuarter
      translate <0, lengthRoom / 2, lengthRoom / 2> * scaleBlock
```

```
      // Apply the symmetry to the odd quarters
      #if (iQuarter = 1 | iQuarter = 3)
        scale <-1.0, 1.0, 1.0>
        translate widthRoom * x * scaleBlock
      #end
    }

    light_source {
      <widthRoom + 2, heightPlatformC + 1 + heightDoorE - 1, 19 + widthDoorE
          / 2> * scaleBlock
      color rgb 1.0 * lightIntensity
      area_light <2, 0, 0> <0, 0, 2> 4, 4 adaptive 0 jitter

      // Rotate the quarter along the horizontal axis
      translate <0, -lengthRoom / 2, -lengthRoom / 2> * scaleBlock
      rotate x * 90.0 * iQuarter
      translate <0, lengthRoom / 2, lengthRoom / 2> * scaleBlock

      // Apply the symmetry to the odd quarters
      #if (iQuarter = 1 | iQuarter = 3)
        scale <-1.0, 1.0, 1.0>
        translate widthRoom * x * scaleBlock
      #end
    }

    light_source {
      <widthRoom / 2, lengthRoom / 4, lengthRoom / 4> * scaleBlock
      color rgb 0.3 * lightIntensity
      area_light <2, 0, 0> <0, 0, 2> 4, 4 adaptive 0 jitter

      // Rotate the quarter along the horizontal axis
      translate <0, -lengthRoom / 2, -lengthRoom / 2> * scaleBlock
      rotate x * 90.0 * iQuarter
      translate <0, lengthRoom / 2, lengthRoom / 2> * scaleBlock

      // Apply the symmetry to the odd quarters
      #if (iQuarter = 1 | iQuarter = 3)
        scale <-1.0, 1.0, 1.0>
        translate widthRoom * x * scaleBlock
      #end
    }

    #declare iQuarter = iQuarter + 1;
  #end

  /*light_source {
    <widthRoom / 2, lengthRoom / 2, lengthRoom / 2>
    color rgb 0.1
  }*/

}

object { Lights }

// ------ Scene ------

object {
  HouseOfStairs
}

// ------ Frame ------
```

```
#declare QuarterFrame = union {
  #local r = 0.1;
  cylinder {
    <0, 0, 0>,
    <widthRoom, 0, 0>,
    r
  }
  cylinder {
    <0, 0, 0>,
    <0, lengthRoom / 2, 0>,
    r
  }
  cylinder {
    <0, 0, 0>,
    <0, 0, lengthRoom / 2>,
    r
  }
  cylinder {
    <widthRoom, 0, 0>,
    <widthRoom, lengthRoom / 2, 0>,
    r
  }
  cylinder {
    <widthRoom, 0, 0>,
    <widthRoom, 0, lengthRoom / 2>,
    r
  }
  cylinder {
    <0, lengthRoom / 2, 0>,
    <widthRoom, lengthRoom / 2, 0>,
    r
  }
  cylinder {
    <0, 0, lengthRoom / 2>,
    <widthRoom, 0, lengthRoom / 2>,
    r
  }
  cylinder {
    <0, lengthRoom / 2, 0>,
    <0, lengthRoom / 2, lengthRoom / 2>,
    r
  }
  cylinder {
    <0, 0, lengthRoom / 2>,
    <0, lengthRoom / 2, lengthRoom / 2>,
    r
  }
  cylinder {
    <widthRoom, lengthRoom / 2, 0>,
    <widthRoom, lengthRoom / 2, lengthRoom / 2>,
    r
  }
  cylinder {
    <widthRoom, 0, lengthRoom / 2>,
    <widthRoom, lengthRoom / 2, lengthRoom / 2>,
    r
  }
  texture {pigment {color rgb x}}
  scale scaleBlock
}

//object { QuarterFrame }
```



Adding one more light with low intensity in the center of the duplicated quarter, removing the randomness of the block size and solving the texture problem in the rounded door brings me to a satisfying solution.

The remaining problem is a lack of contrast in the image. I think it comes from the texture of the blocks, then consider the light model ok and move on finalizing the texture.

6 Block texture model