AN ILLUSTRATED GUIDE

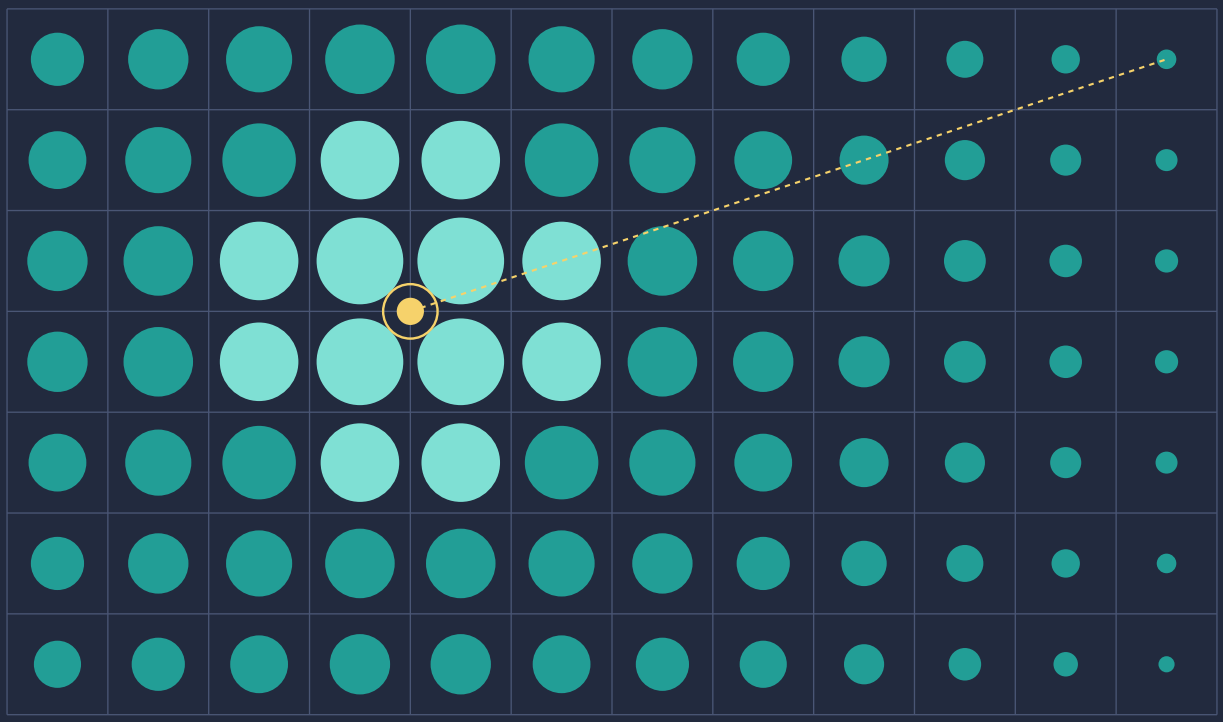
Parametric Design

A friendly guide to designing with relationships, rules and parameters

Plain English, clear pictures, real computed examples with real numbers, and a glossary, a quiz, a cheat sheet and an eight-week plan.

Sixteen short chapters · Quiz · Glossary · Eight-week plan

Prompted and directed by Mahmoud A.
Written and illustrated by Claude (Anthropic)



attractor (yellow) · each opening's size follows its distance to it

one rule, 84 panels

How to Use This Guide

Read it in order, or jump to what you need.

This guide explains **parametric design**, a way of designing in which you set up relationships and rules and let the computer draw the shapes. Change one number and the whole design updates. You need no coding and no maths beyond school level. Every idea comes with a picture.

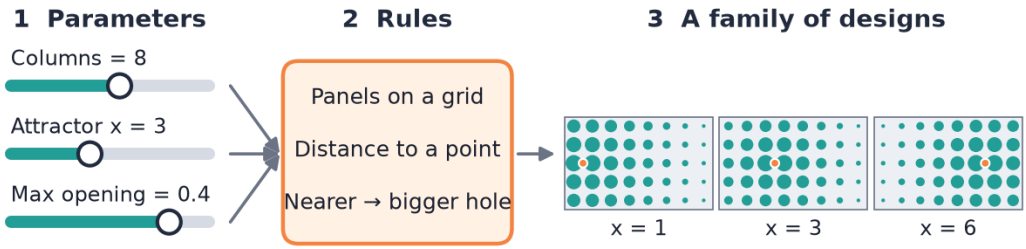


Figure 1. The whole guide in one picture. A few numbers feed a set of rules, and the rules produce a whole family of designs. Each thumbnail moves one point, the attractor, to a new x position.

Who it is for

- Curious people who keep hearing words like **parametric** or **attractor** and want to know what they mean.
- Designers, engineers and students who want to move from fixed shapes to flexible models.
- Anyone who needs to judge when a parametric approach is worth the effort, and when it is not.

How each chapter works

- A **banner** tells you the topic in a sentence.
- Plain explanation first, then a picture, then a small example.
- Boxes: **TIP** (a shortcut), **SAY IT LIKE THIS** (a phrase you can use out loud), **WATCH OUT** (a common trap) and **IN ONE LINE** (the chapter in a sentence).
- A few short code snippets in grey boxes show the same idea as text. You can skip them.
- At the end: myths and facts, a quiz, a glossary, a cheat sheet and further reading.

A note on the numbers

Every worked example is **real**. The Python script that builds this guide computes each staircase, opening, louvre, tiling, panel check and massing, then draws the figures and fills in the numbers from those results. The rules are **illustrative**. They show how the logic works and do not come from any real building code. Before using rules like these, check the ones that apply where you build.

Contents

Sixteen short chapters and a toolbox at the back.

1 **What “Parametric” Really Means**
Designing the rules instead of the shape.

2 **A Short History in Plain Words**
Old ideas, new tools.

3 **Families of Designs**
One model, many answers.

4 **Boxes and Wires**
How visual programming tools work.

5 **Numbers, Lists and Trees**
What actually flows along the wires.

6 **Geometry Building Blocks**
Points, vectors, planes, curves, surfaces and meshes.

7 **Transformations**
Move, rotate, scale and mirror, in the right order.

8 **Attractors and Fields**
Letting distance shape the design.

9 **Worked Example: Louvres**
A screen whose blade angles follow a rule.

10 **Patterns, Tiling and Panels**
From a smooth surface to buildable pieces.

11 **Conditions, Culling and Chance**
Making choices, and making randomness repeatable.

12 **Scaling Up: Rule-Based Massing**
The same thinking at building and city scale.

13 **From Model to Making**
Documentation and fabrication-ready data.

14 **Good Habits and Pitfalls**
Keeping a definition clean, and the traps to avoid.

15 **Tools and Where to Go Next**
Choosing a tool, and what to learn after this guide.

16 **Try It Yourself**
A short recipe, free tools and an eight-week plan.

At the back: Myths and Facts · Quick Quiz · Glossary · One-Page Cheat Sheet · Further Reading

1

What “Parametric” Really Means

Designing the rules instead of the shape.

Most drawings are fixed. A wall is 4 metres long because someone drew it 4 metres long. If the brief changes, someone redraws it. In parametric design you describe *how* things relate, and the shape follows from those relationships. Change one input and everything that depends on it updates by itself.

Three words to start with

- **Parameter.** An input you can change, usually a number. The height of a floor, the number of panels, the size of an opening.
- **Relationship.** A rule that links one thing to another. “Each riser is the floor-to-floor height divided by the number of risers.”
- **Constraint.** A limit the design must respect. “No riser may be taller than 180 mm.”

An everyday analogy

Think of a pancake recipe written per person: so many grams of flour, so much milk and one egg for each guest. The number of guests is the parameter, and every amount follows from it. If six friends turn up instead of four, you do not write a new recipe. You change one number. A parametric model is a recipe for a design.

A first worked example: a staircase

A staircase is a perfect first model, because its parts depend on each other. The script that builds this guide uses three rules:

1. The number of **risers** (the vertical part of each step) is the floor-to-floor height divided by 180 mm, rounded *up* to a whole number. Rounding up keeps every riser at or below the limit.
2. Each riser is the height divided by that number, so all the steps are equal.
3. The **going** (how far forward each step takes you, measured from one step edge to the next) follows a long-standing rule of thumb that two risers plus one going add up to about 630 mm.

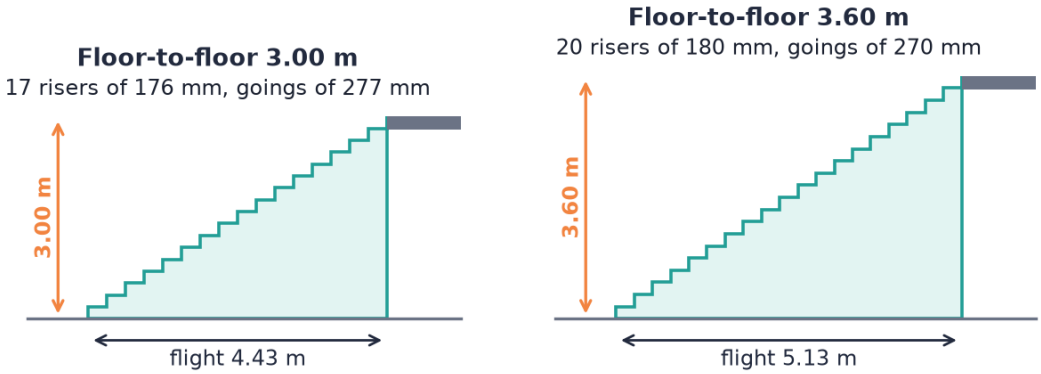


Figure 2. The same three rules drawn for two floor heights. Every number comes from the rules; nothing was drawn by hand.

For a floor-to-floor height of 3.00 m the rules give **17 risers** of 176 mm, goings of 277 mm and a flight 4.43 m long (its horizontal length: there is one going fewer than there are risers). Raise the height to 3.60 m and the same rules give **20 risers** of 180 mm, goings of 270 mm and a flight 5.13 m long. Nobody redrew anything. One number changed, and the table below shows two more heights.

Floor-to-floor height	Risers	Riser height	Going	Flight length
2.70 m	15	180 mm	270 mm	3.78 m
3.00 m	17	176 mm	277 mm	4.43 m
3.30 m	19	174 mm	283 mm	5.09 m
3.60 m	20	180 mm	270 mm	5.13 m

Notice that the flight does not grow smoothly. Going from 3.00 m to 3.30 m adds 2 risers and 0.65 m of flight, but going from 3.30 m to 3.60 m adds only 1 riser and 0.04 m, because every riser grows from 174 mm to 180 mm instead. Rules with whole numbers often behave in jumps like this, and a parametric model shows you exactly where the jumps happen.

WATCH OUT

Rules of thumb are a starting point. Real stairs must follow the building rules where they are built, which set their own limits for risers, goings and headroom.

SAY IT LIKE THIS

“A parametric model is a recipe for a design. I set the ingredients and the rules, and the computer works out every variation.”

IN ONE LINE

Parametric design means describing how the parts relate, so that changing one input updates everything that depends on it.

2 A Short History in Plain Words

Old ideas, new tools.



Figure 3. Five ways of designing with rules, from hand to screen. Each one links parts so that changing one changes the others.

Before computers

- Classical architecture used **proportion systems**. Many parts of a column were sized as multiples of one **module**, often its diameter. Change the module and the whole column rescales.
- Shipbuilders drew hull curves with flexible strips of wood called **splines**, held by heavy weights. Move a weight and the whole curve adjusts. The word lives on in today's software.
- **Antoni Gaudí** studied arches with hanging models of strings and small weights. Gravity pulls each string into a shape in pure tension. Turned upside down, it suggests an arch in pure compression.

On the screen

- Ivan Sutherland's **Sketchpad** (1963) let users set constraints such as “stay parallel”.
- Mechanical engineering adopted **feature-based modelling**. Edit a dimension and the part rebuilds from its history of steps.
- Architecture took up parametric tools such as GenerativeComponents and then **visual programming** tools such as Grasshopper for Rhino and Dynamo for Revit.
- Mark Burry and colleagues used parametric software for decades to study and continue Gaudí's Sagrada Família.

The architect Patrik Schumacher coined the word **parametricism** for a style of flowing, curved architecture. This guide is about the method, which works just as well for a plain box, a staircase or a street layout.

IN ONE LINE

Designing with rules is old; computers let us follow the rules faster and try many more versions.

3 Families of Designs

One model, many answers.

A fixed drawing is one design. A parametric model is a whole **family** of designs, and each set of input values picks out one member. The full set of possible results is called the **design space**. Below, a tower of 24 floors has two parameters. **Twist** turns each floor a little more than the one below. **Taper** shrinks the floors towards the top. Three values of each give nine members of the family.

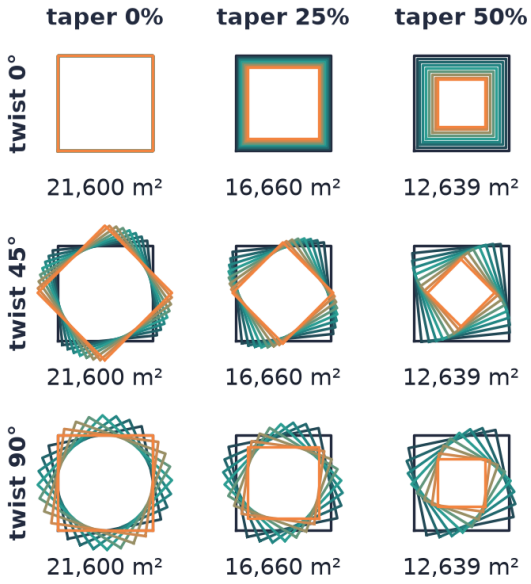


Figure 4. Nine towers from one model, seen from above (dark floors low, orange high), with floor areas.

Twisting changes the look but not the floor area, because turning a square does not change its size. Tapering until the top floor is half as wide as the base cuts the total from 21,600 m² to 12,639 m², so a slimmer top costs floor area.

Each parameter you can change is a **degree of freedom**. Two parameters with three values give 9 variants; five with ten values give 100,000. In an **associative** model, objects are built from other objects and remember how. Move a point and everything built on it follows.

TIP

Before you build, list the three to five parameters that matter most and their ranges. That list is your design space on paper.

IN ONE LINE

A parametric model is a design space; each set of inputs picks one design.

4 Boxes and Wires

How visual programming tools work.

Most parametric tools let you build a model without typing code. You place boxes on a canvas and connect them with wires. This is called **visual programming**. The finished box-and-wire drawing is often called a **definition**, a **graph** or a **script**.

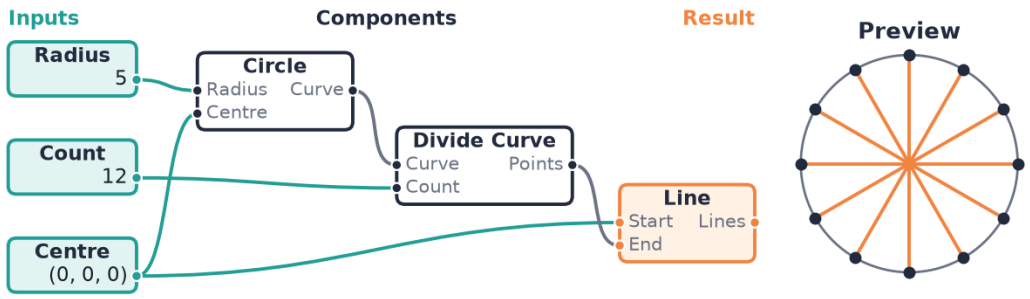


Figure 5. A small definition. Three inputs feed three components. The result is a circle of radius 5 divided into 12 points, each joined to the centre.

The parts of a graph

- **Components** (called **nodes** in some tools) are boxes that do one job: make a circle, divide a curve, draw a line.
- **Inputs** on the left of a component receive data. **Outputs** on the right send results on.
- **Wires** carry data from an output to an input.
- **Sliders** and **panels** are where you drag or type numbers. They are the parameters.
- The **preview** shows the result in the 3D view. When you are happy, you **bake** it, which copies the live result into the model as ordinary, fixed geometry.

How it updates

In most tools data flows from left to right. Drag the Count slider and the tool works out which components depend on it and recalculates only those. The circle does not depend on Count, so it is not rebuilt. This chain of dependencies is a **directed acyclic graph**. Directed means data flows one way. Acyclic means it never loops back on itself. Grasshopper (inside Rhino) and Dynamo (inside Revit) work this way, and Blender's Geometry Nodes also flows from left to right. Houdini follows the same idea, but its networks usually flow from top to bottom, and it has built-in blocks for repeating steps.

The same graph as text

Every graph can also be written as a short program. Here is the graph above in Python. You can skip it, but it shows that a graph is simply a recipe written with boxes.

```
from math import cos, sin, pi
radius = 5          # slider
count = 12            # slider
points = [(radius * cos(2 * pi * i / count),
           radius * sin(2 * pi * i / count)) for i in range(count)]
lines = [((0, 0), p) for p in points]   # spokes to the centre
```

Because a plain graph may not loop, a basic definition cannot say “repeat until good enough”. For that you need scripting or a solver. Chapter 15 points to both.

TIP

Read a graph like a sentence, from left to right: inputs, then logic, then results. If you cannot read your own graph this way, it is time to tidy it (Chapter 14).

SAY IT LIKE THIS

“The sliders are my parameters, the components are my rules, and the wires show what depends on what.”

IN ONE LINE

Visual programming builds a model from boxes that each do one job, joined by wires that carry data one way, usually from left to right.

5 Numbers, Lists and Trees

What actually flows along the wires.

A wire can carry one value or thousands. Understanding how tools organise many values is one of the biggest steps from beginner to confident user.

Numbers, domains, ranges and series

- A **number** is one value, such as a radius of 5.
- A **domain** (or interval) is the stretch of numbers between two ends, such as 0 to 10.
- In Grasshopper, a **range** divides a domain into equal steps. A range from 0 to 10 in 5 steps gives 6 numbers, because 5 steps need 6 fence posts. (Python's range() works differently: it takes a start, an end and a step, and leaves out the end.)
- A **series** starts somewhere and adds a fixed step a set number of times. Start 0, step 3, count 5 gives 0, 3, 6, 9, 12.

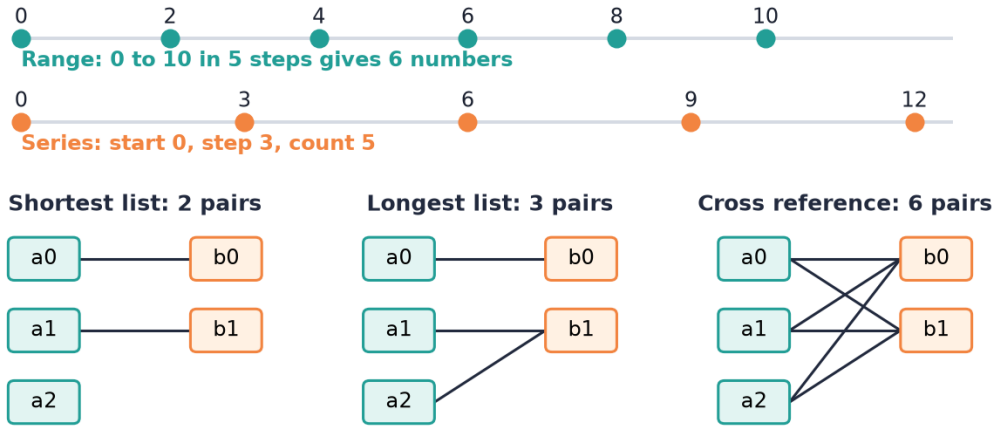


Figure 6. Top: a range and a series. Bottom: three ways to pair a list of three items with a list of two. The pair counts are worked out by the script.

Lists and list matching

A **list** is an ordered row of values. Each item has an **index**, its position in the list. Most tools count from 0, so the first item is item 0. When a component receives two lists of different lengths it must decide how to pair them. This is called **list matching**. *Shortest list* stops when the shorter list runs out. *Longest list* repeats the last item of the shorter list. *Cross reference* pairs everything with everything.

Data trees

Often you need lists of lists. A building has floors, and each floor has windows. Grasshopper stores this as a **data tree**: several lists, each with an address called a

path, such as {0}, {1} and {2}. Dynamo uses nested lists for the same job. Think of a filing cabinet. Each drawer is a path, and each drawer holds one list.

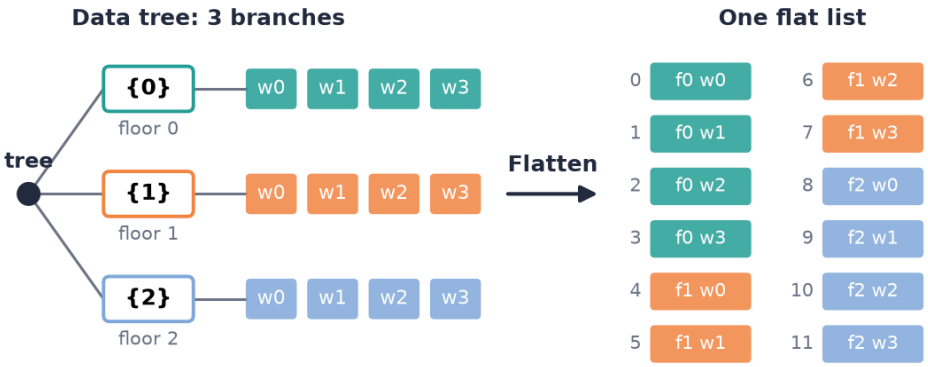


Figure 7. A data tree with three branches (one per floor) and four windows in each. Flattening keeps the order but forgets which floor each window was on.

Operation	What it does	Typical use
Flatten	Puts every item into one list	Count or sort everything together
Graft	Gives every item its own branch	Make one result per item, such as one line per point
Flip (transpose)	Swaps rows and columns	Turn floors-by-windows into windows-by-floors
Partition	Splits a list into chunks of n	Group points into rows of a grid
Shift	Moves every item along by n places	Pair each point with its neighbour

WATCH OUT

Most strange results in visual programs come from the shape of the data. When something looks wrong, first check how many items and branches each wire carries.

IN ONE LINE

Wires carry numbers, lists and trees of lists, and matching them correctly matters as much as the geometry.

6 Geometry Building Blocks

Points, vectors, planes, curves, surfaces and meshes.

Every parametric model is built from a small set of geometric parts. You do not need the maths behind them, but you do need to know what each one is and what it is good for.

Points, vectors and planes

- A **point** is a position, written as coordinates (x, y, z).
- A **vector** is a direction with a length, like an arrow that can be placed anywhere. Subtract one point from another and you get the vector between them. A vector of length 1 is a **unit vector**. It shows direction only.
- A **plane** is a flat sheet with an origin and two axes, X and Y. A third direction, the **normal**, points straight out of it. Planes tell objects where to sit and which way to face.

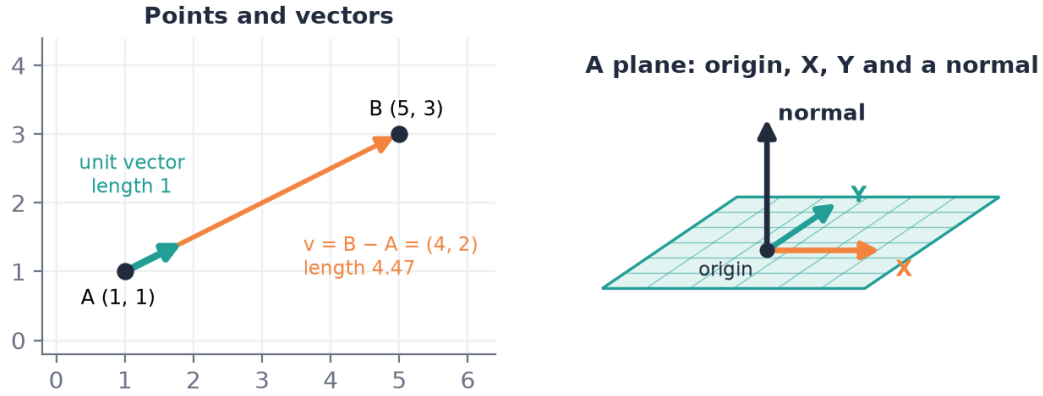


Figure 8. Left: the vector from A to B, and the unit vector pointing the same way. Right: a tilted plane with its two axes and its normal.

In the figure, the vector from A to B is (4, 2). Its length is $\sqrt{(4^2 + 2^2)} \approx 4.47$. Dividing by the length gives the unit vector (0.89, 0.45). Two more tools appear everywhere. The **dot product** of two unit vectors tells you how much they agree: 1 if they point the same way, 0 at right angles, −1 if opposite. The **cross product** gives a vector at right angles to both, which is how software finds normals.

Curves

A **line** runs straight between two points, and a **polyline** is a chain of straight pieces. Smooth curves are usually **NURBS** (non-uniform rational B-splines). A NURBS curve is pulled into shape by **control points**. Unlike the weights on a shipbuilder's spline, which sit on the curve, control points usually sit off it, like magnets pulling the curve towards them. Its **degree** sets how smooth it is: 1 gives a polyline, 3 the usual smooth curve. A **weight** makes a control point pull harder, and hidden **knots** set

where each point's pull starts and stops.

Every curve also has a **parameter** in a second, more mathematical sense: a number, usually called t, that runs along it, often from 0 at the start to 1 at the end. “Evaluate the curve at t = 0.5” means “give me the point at parameter 0.5”. That point is not always halfway along.

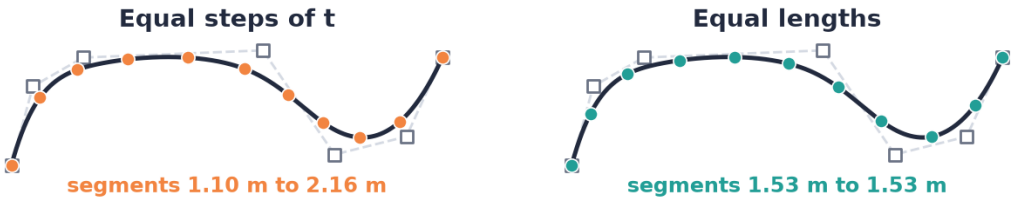


Figure 9. The same curve divided into 10 parts two ways. Dashed lines join the control points. The curve is 15.27 m long.

The script tested this. Equal steps of t give segments from 1.10 m to 2.16 m long. Dividing by length gives segments of 1.53 m each. Most tools offer both, so choose on purpose.

Surfaces and meshes

A **NURBS surface** is the same idea stretched in two directions, with two parameters called u and v. Stepping through u and v gives a grid of points on the surface, the starting point for most panelling. A **mesh** is a surface made of many small faces, usually triangles or quads (four-sided faces), joined at shared corners called **vertices**. Triangles are always flat; quads need not be (Chapter 10).

Object	Defined by	Good for
Point	Coordinates x, y, z	Positions, grid nodes, attractors
Vector	A direction and a length	Moving things, directions, normals
Plane	Origin, X axis, Y axis	Placing and orienting objects
NURBS curve	Control points, degree, weights, knots	Precise smooth edges and paths
NURBS surface	A grid of control points (u and v)	Smooth façades, roofs and products
Mesh	Vertices and faces	Analysis, rendering, 3D printing

TIP

When you divide a curve, ask for the count or the length you want, then check the real segment lengths. Curves are rarely as even as they look.

IN ONE LINE

Points, vectors and planes place things; curves, surfaces and meshes give them shape.

7 Transformations

Move, rotate, scale and mirror, in the right order.

A **transformation** changes where an object is, which way it faces or how big it is, without changing what it is. Five cover most daily work.

Transformation	Needs	Example
Move (translate)	A vector	Shift a panel 3 m along the façade
Rotate	An angle and an axis (a centre in 2D)	Turn a louvre blade by 30°
Scale	A factor and a centre	Make each floor 98% of the one below
Mirror	A line or plane	Make the left wing match the right
Orient	Two planes	Lay a panel onto a surface point

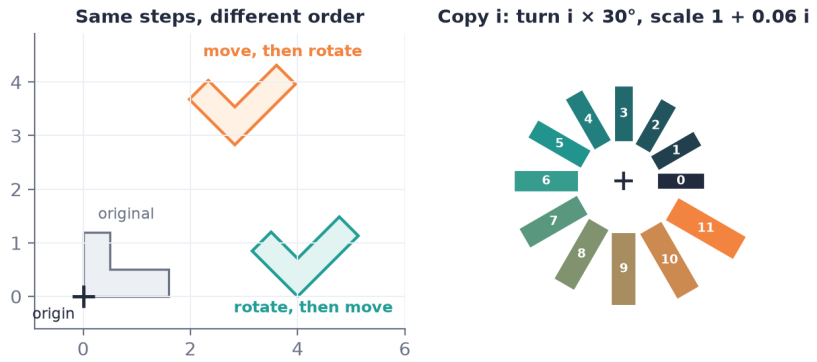


Figure 10. Left: one rotation and one move, in two orders. Right: an array driven by a rule.

Order matters, because rotation turns things about a centre, here the origin. Rotating by 45° and then moving 4 m puts the shape's centre at (4.14, 0.75). Moving first puts it at (2.97, 3.58). Software stores each step as a **transformation matrix**, a small grid of numbers, and chaining steps multiplies the matrices in order.

Rules can set the amounts. On the right, copy i turns by i × 30° and is scaled by 1 + 0.06 × i, so the last of the 12 copies is 1.66 times the size of the first.

WATCH OUT

Scaling about the world origin moves an object as well as resizing it. Scale about the object's own centre unless you mean otherwise.

IN ONE LINE

Move, turn, resize and mirror in the right order, and let rules set the amounts.

8 Attractors and Fields

Letting distance shape the design.

An **attractor** is a point, or a curve, that influences everything around it. For each object you measure its distance to the attractor and use that distance to set something: a size, an angle, a colour. It is one of the best-known parametric patterns, and a good one to learn first because it combines points, lists and simple maths.

The recipe

1. Divide a façade into a grid of panels and find each panel's centre point.
2. Measure the distance from each centre to the attractor point.
3. **Remap** the distances to a scale from 0 to 1. The nearest panel gets 0 and the farthest gets 1.
4. Turn that 0 to 1 value into an opening size, here from 440 mm next to the attractor down to 60 mm far away.

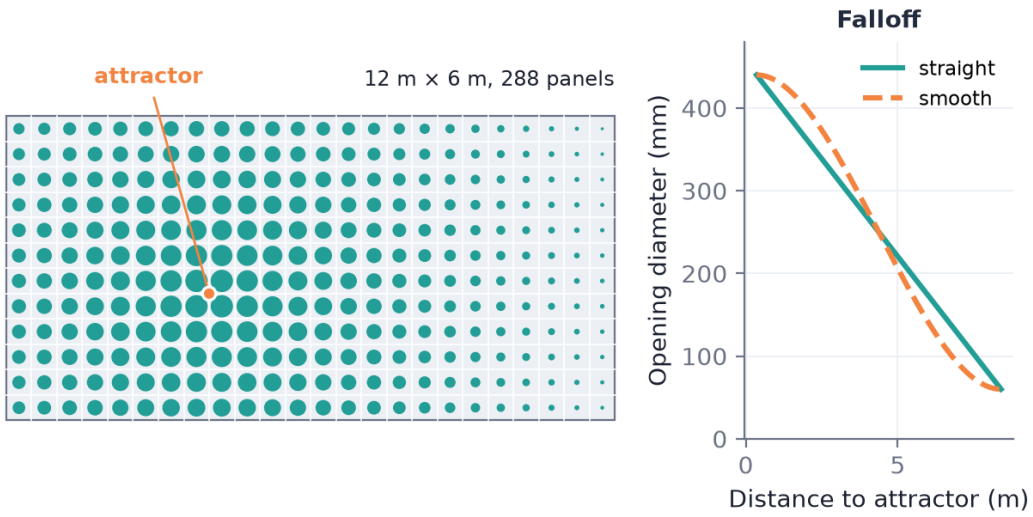


Figure 11. Left: each panel's opening follows its distance to one point. Right: the falloff curve that turns distance into size. The straight version drew the façade.

The script ran exactly this recipe on a 12 m × 6 m façade of 288 panels, each 500 mm square, with the attractor at (4, 2.5). Distances ranged from 0.35 m to 8.40 m, and the openings cover **26.1%** of the façade. Moving the attractor to (9, 4) changes the open area to **26.5%**, without touching any other part of the model.

Remapping in two lines

```
t = (d - d_min) / (d_max - d_min)              # 0 near, 1 far
size = size_max - t * (size_max - size_min)    # big near, small far
```

The shape of the falloff is a parameter too. The smooth curve in the figure keeps openings large for longer, then drops. With the smooth falloff and the same attractor, the open area becomes 29.0% instead of 26.1%.

From one point to a field

A **field** gives every point in space a value. With several attractors you can use the distance to the nearest one, or add their effects together. Curves can be attractors too, which suits a path or a river edge. You can even sample an image, so that a photo's brightness drives the opening sizes.

WATCH OUT

Attractor patterns are easy to make and easy to overuse. Ask what the attractor stands for: a view, a sun direction, an entrance, a structural need. If it stands for nothing, the pattern is only decoration.

SAY IT LIKE THIS

“Each panel measures its distance to the attractor, and that distance sets its opening size.”

IN ONE LINE

An attractor turns distance into a design decision, one panel at a time.

9 Worked Example: Louvres

A screen whose blade angles follow a rule.

A **louvre** is a screen of thin blades, or **fins**, that controls light, view and privacy. In a parametric louvre each blade's angle comes from a rule, so the whole screen can be tuned with a few numbers.

The set-up

The script models 15 vertical fins seen from above (in plan), 600 mm apart and 500 mm deep. At 0° a fin points straight out from the glass. At 90° it would lie flat along the façade. We measure one simple result, the share of the view straight out that stays open. Seen head-on, a fin turned by an angle θ blocks d × sin θ of each gap, so the open share is (s − d × sin θ) ÷ s, where s is the spacing and d the depth.

Two rules

- **Rule A, the ramp.** Angles rise evenly from 0° at the left end to 75° at the right. The left end stays open and the right end becomes private.
- **Rule B, the wave.** Angles follow a smooth bump: 0° at both ends and 75° in the middle.

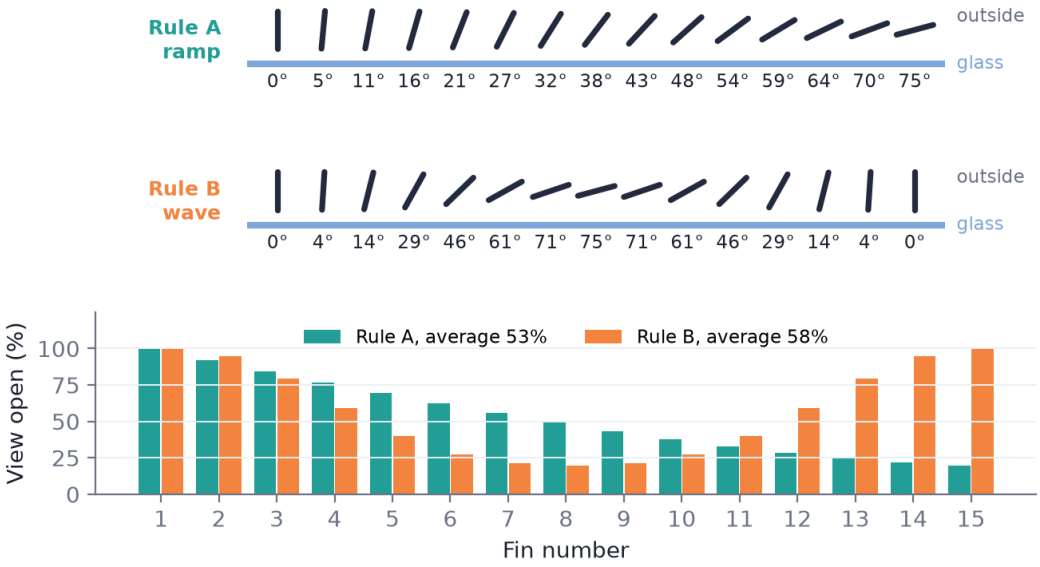


Figure 12. Top: the two screens in plan, with each fin's angle written under it. Bottom: the share of the straight-out view left open behind each fin.

The script's results show that Rule A leaves **53%** of the straight-out view open on average, Rule B **58%**. Change only the largest angle from 75° to 45° and Rule A opens up to 69%, Rule B to 71%. The rules stay the same; one parameter moved.

A second rule: spacing from the sun

Horizontal blades are often spaced with a simple rule. Picture a section cut through the façade. The sun's height in that section is called the **profile angle**. When the sun is higher than a chosen **cut-off angle**, no direct sun should get through. For flat blades of depth d, the largest spacing that achieves this is d × tan(cut-off angle). The table shows the script's results for blades 300 mm deep on a 3 m storey.

Cut-off angle	Largest spacing	Blades per 3 m of height
30°	173 mm	18
45°	300 mm	10
60°	520 mm	6

The counts assume a continuous screen, so the top blade of one storey is the bottom blade of the next. Lower cut-off angles block more sun but need more blades, which means more material and less view. The rule makes that trade-off visible in seconds.

TIP

Real sun angles change with the hour, the season and the direction a façade faces. Environmental plug-ins can compute them for your site and feed them straight into rules like these.

IN ONE LINE

Give each blade a rule instead of a fixed angle, then tune the whole screen with one or two numbers.

10

Patterns, Tiling and Panels

From a smooth surface to buildable pieces.

Buildings are made from pieces. **Tiling** covers an area with shapes that leave no gaps and do not overlap. **Panelisation** splits a surface into pieces that can be made, moved and installed. Both start with a pattern.

Three common tilings

- **Squares.** The simplest grid. Every cell is the same.
- **Hexagons.** They need less edge for the same area. The script computed the edge needed to enclose one square metre: 4.56 m with triangles, 4.00 m with squares and 3.72 m with hexagons.
- **Voronoi.** Scatter some seed points, then give each seed the area that is closer to it than to any other seed. The result looks natural, and every cell is different.

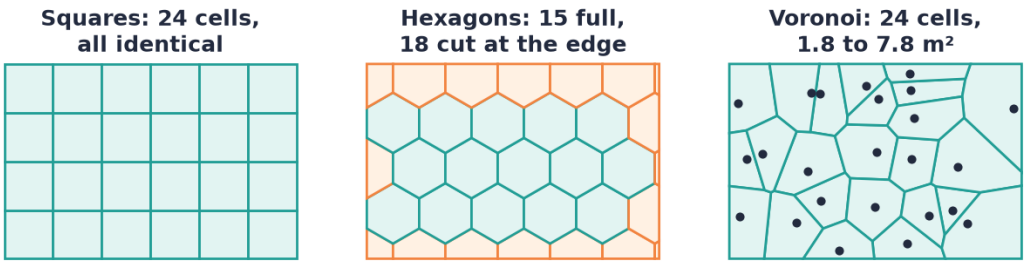


Figure 13. The same 12 m × 8 m area tiled three ways, with cells of about 4 m² each. Orange cells had to be cut at the edge.

The numbers tell a story. The square grid fits exactly, so all 24 pieces are identical. The hexagons give 15 full cells but also 18 cut ones, each a special piece. The Voronoi pattern from 24 seeds has cells from 1.8 m² to 7.8 m², so every piece is unique.

Flat panels on a curved surface

Flat sheets of glass or metal are much cheaper than curved ones. A four-sided panel is flat only if its four corners lie on one plane, and on a curved surface they usually do not. A simple check is to draw both diagonals of the panel and measure the gap between them. If the gap is zero, the panel is flat. That gap is called the **warp**.

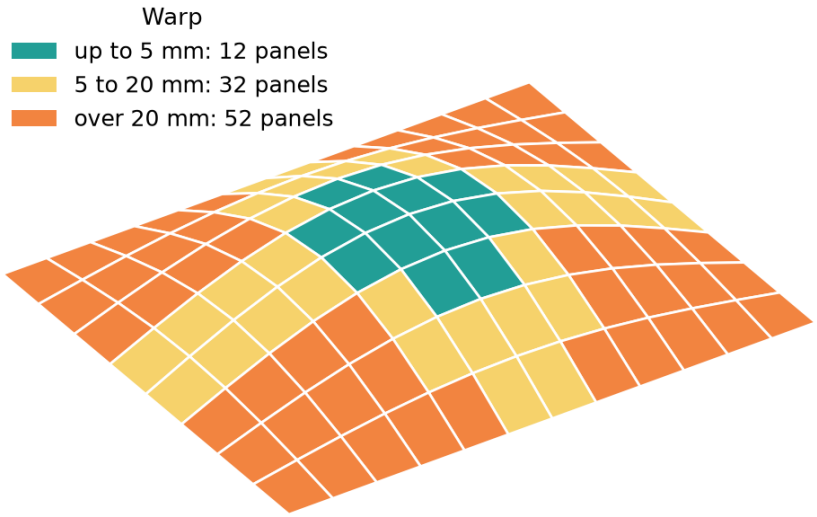


Figure 14. A cushion-shaped roof divided into 96 panels of 2 m and coloured by warp. Heights are drawn three times larger so the shape is easy to see.

The roof is 24 m × 16 m and rises 1.2 m in the middle. Panels in the middle are nearly flat, panels along the two centre lines stay fairly flat, and panels towards the corners twist most, up to 59 mm. How many pass depends on the tolerance you choose:

Tolerance	Panels that pass as flat	Panels to rethink
5 mm	12 of 96	84
10 mm	28 of 96	68
20 mm	44 of 96	52

Panels that fail can be split into two triangles, because three points always lie on one plane, or the grid can be redesigned. Adjusting a design so that it can be built economically is called **rationalisation**.

WATCH OUT

A pattern that looks good on screen can produce hundreds of unique parts. Count the different pieces early. Every unique piece costs time in drawing, cutting, labelling and installing.

IN ONE LINE

Choose a pattern, then check that its pieces can be made: flat enough, few enough types and sensible edges.

11 Conditions, Culling and Chance

Making choices, and making randomness repeatable.

So far every panel followed the same rule. Real designs need exceptions, such as a door here, a solid panel there or no openings next to a column. Conditional logic adds those choices.

If this, then that

A **condition** is a test that answers true or false, such as “is this opening smaller than 120 mm?” Asking it of every item gives a list of true and false values called a **mask**. Components then use the mask to choose or remove items. Removing items with a mask is called **culling**.

```
keep = [size_mm >= 120 for size_mm in sizes_mm]  # the mask
holes = [p for p, k in zip(panels, keep) if k]   # culling
```

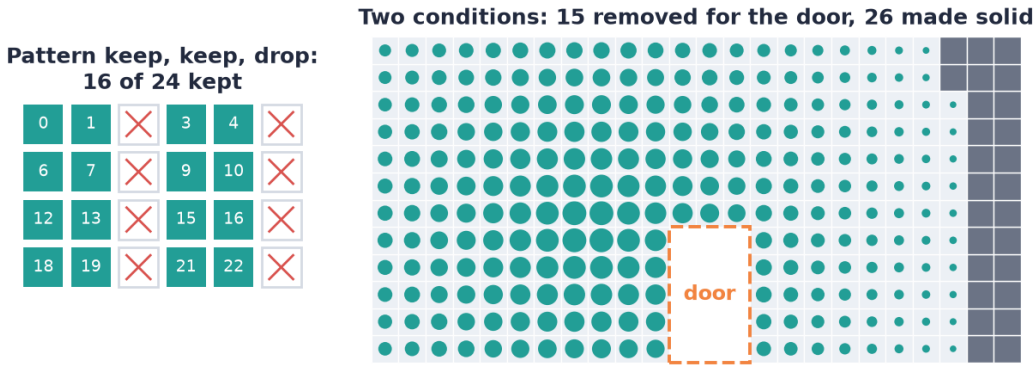


Figure 15. Left: a repeating cull pattern. Right: two conditions applied to the attractor façade from Chapter 8.

On the left, the pattern keep, keep, drop runs along 24 items and leaves 16. On the right, panels inside a door zone are removed (15 panels). Of the rest, any opening smaller than 120 mm becomes a solid panel (26 panels), because a hole that small is hard to make and does little. That leaves 247 panels with openings. Change the threshold and the counts update.

Randomness with a seed

A little randomness can make a pattern feel less mechanical. Computers make **pseudo-random** numbers. They look random but come from a formula that starts from a number called the **seed**. The same seed always gives the same numbers. A different seed gives a different but equally random result.

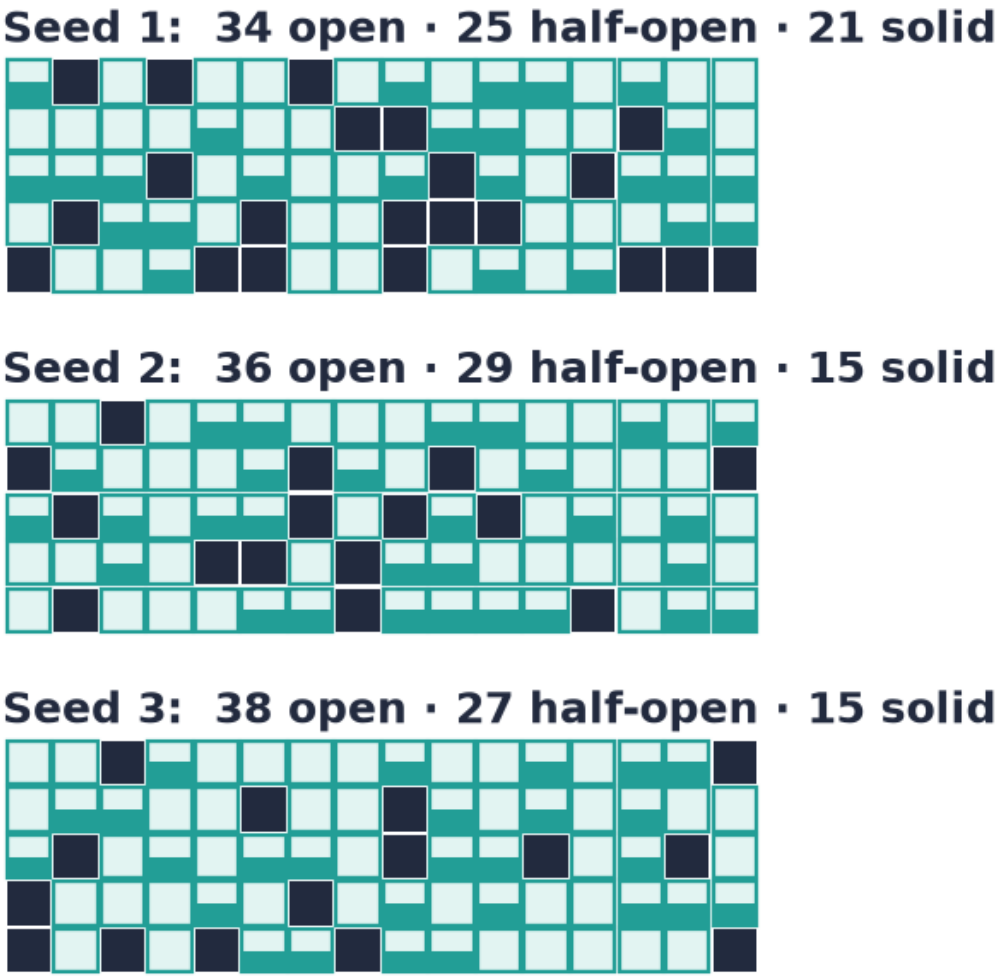


Figure 16. One rule, three seeds. Each panel is open, half-open or solid, chosen at random with chances of 50%, 30% and 20%.

Each façade has 80 panels. With a perfect 50/30/20 mix that would be 40, 24 and 16. Seed 1 gave 34, 25 and 21; seed 2 gave 36, 29 and 15; seed 3 gave 38, 27 and 15. The script also ran seed 1 a second time and checked that every panel matched.

TIP

Always set the seed and write it down. Then “the random one we liked” can be rebuilt exactly, by you or by anyone else.

WATCH OUT

Random is not the same as even. Small samples drift from the target mix, as the counts above show. If the exact mix matters, make a list with the exact counts and shuffle it instead.

IN ONE LINE

Conditions decide what to keep, and seeds make chance repeatable.

12 Scaling Up: Rule-Based Massing

The same thinking at building and city scale.

Parametric thinking works at every scale. A chair, a façade and a city block all have parameters, relationships and constraints. At urban scale, many rules come from planning: how far to stay back from the plot boundary, how high to build and how much floor area is allowed.

A block massing from rules

Massing means the overall shape and size of a building before any detail. The script builds a massing on a plot 40 m wide and 30 m deep with these illustrative rules:

- **Setbacks** (gaps kept clear inside the plot boundary): 5 m at the front, 3 m at each side, 4 m at the rear.
- Ground floor 4.5 m high, upper floors 3.2 m. Only whole floors that fit under the height limit count.
- Floors that start at 15 m or higher step back a further 3 m from the street.
- **Floor area ratio** (FAR), the total floor area divided by the plot area, may not exceed 4.

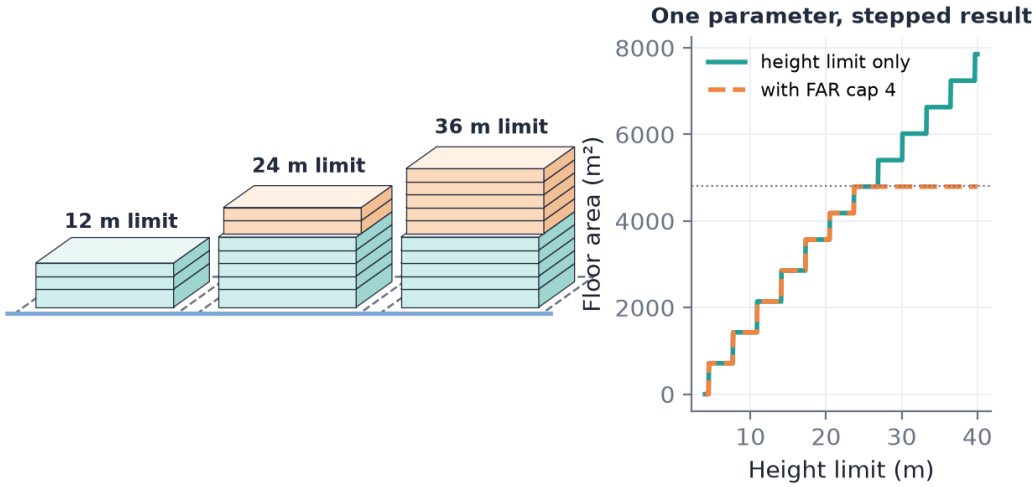


Figure 17. Left: the same rules with three height limits (orange floors are stepped back). Right: total floor area as the height limit rises, with and without the FAR cap.

Height limit	Floors	Floor area	FAR	Within FAR 4?
12 m	3	2,142 m²	1.785	yes
18 m	5	3,570 m²	2.975	yes
24 m	7	4,794 m²	3.995	yes
30 m	8	5,406 m²	4.505	no
36 m	10	6,630 m²	5.525	no

Watch what one parameter does. A full floor here is 714 m² and a stepped floor 612 m². Raising the height limit from 24 m to 30 m adds only one floor, because the next one would need 30.1 m. Higher limits add more floors but break the FAR cap. With the cap switched on, the building stops at 7 floors, so any height limit above 23.7 m adds nothing. Two constraints interact, and the model shows exactly where each one bites.

From one plot to a street

Apply the same rules to every plot in a street and you have an urban model. The script adds one condition. Plots narrower than 20 m build right up to their side boundaries, as terraced buildings often do. The height limit is 24 m and the FAR cap stays on.

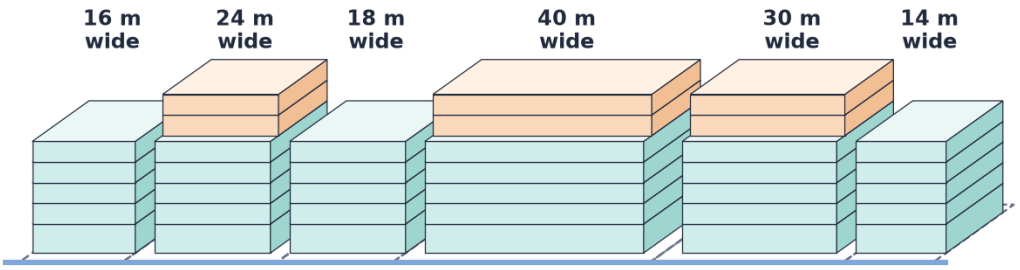


Figure 18. Six plots of different widths, one set of rules. The blue line is the street edge.

The street of 6 plots yields **15,756 m²** of floor area. The narrow plots lose their side gaps, so their floors are relatively larger, and the FAR cap trims the 16, 18 and 14 m plots from 7 floors to 5. Change the height limit, the stepback or the narrow-plot rule, and every building updates together.

WATCH OUT

A massing study shows what the rules allow. Whether the result is good still depends on light, wind, views, heritage and street life, and judging that is the designer's job.

IN ONE LINE

At urban scale, setbacks, height limits and area caps are parameters, and a massing model turns them into shapes and numbers.

13

From Model to Making

Documentation and fabrication-ready data.

A parametric model is only useful once it leaves the computer. That means drawings, schedules and files that a fabricator or builder can trust.

Bake, label, schedule

- **Bake** the final geometry into the normal model, or link it to a **building information model** (BIM), a 3D model that also stores data about every element.
- **Label** every piece with a unique ID that says where it goes, for example R06-C08 for row 6, column 8.
- **Schedule** the pieces. A schedule is a table listing each piece, its size and its type, ready for a spreadsheet.

Fewer types, nearly the same look

The attractor façade from Chapter 8 has 288 openings and 76 different diameters (to a tenth of a millimetre). Few workshops want that many hole sizes. The script rounds each diameter to the nearest 40 mm. That leaves **10 types**, and no opening changes by more than 20 mm.

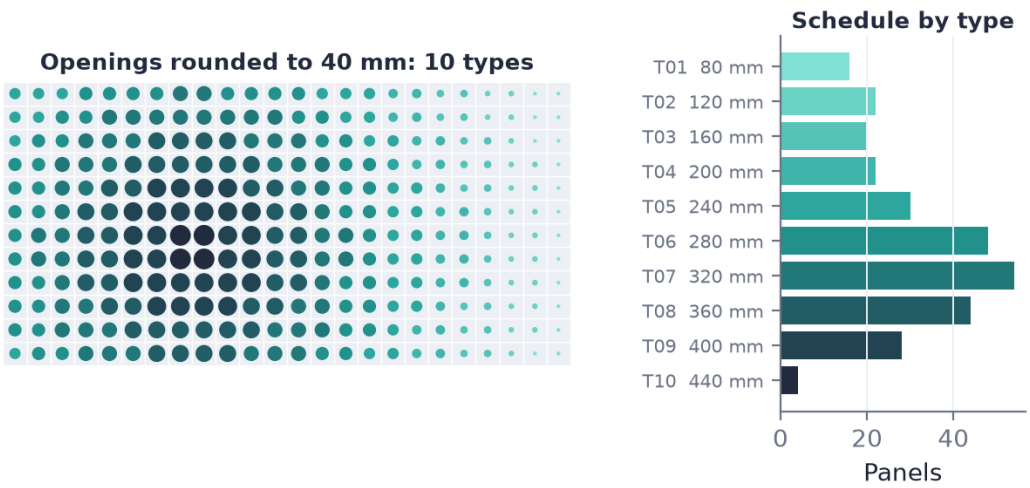


Figure 19. The façade after rounding to 40 mm steps, and the number of panels of each type. This chart is the start of a fabrication schedule.

Rounding step	Opening types	Largest change
10 mm	37	5 mm
20 mm	20	10 mm
40 mm	10	20 mm
80 mm	6	40 mm

Bigger steps mean fewer types but a coarser pattern. The table lets you choose with your eyes open. Here is a short excerpt of the schedule for five panels in row 6:

Panel ID	Exact diameter (mm)	Rounded (mm)	Type
R06-C06	396.5	400	T09
R06-C07	419.4	400	T09
R06-C08	440.0	440	T10
R06-C09	440.0	440	T10
R06-C10	419.4	400	T09

```
import csv
with open("panels.csv", "w", newline="") as f:
    out = csv.writer(f)
    out.writerow(["id", "diameter_mm", "type"])
    for p in panels:
        out.writerow([p.id, p.diameter_mm, p.type])
```

Formats and checks

- **2D cutting files** such as DXF for laser cutters, waterjets and CNC routers.
- **3D exchange** files such as STEP or IGES for NURBS geometry, and OBJ or STL for meshes.
- **Spreadsheets** (CSV) for schedules, and **IFC** for exchanging building models.
- **Checks** before sending: units, tolerances, material thickness, smallest hole, the narrowest strip of material left between holes, and sheet sizes.

TIP

Agree units, tolerances and file formats with the maker before you finish the model. It is far easier to build them into the rules than to fix the files afterwards.

IN ONE LINE

A good model ends in labelled pieces, clear schedules and clean files that a maker can trust.

14

Good Habits and Pitfalls

Keeping a definition clean, and the traps to avoid.

Definitions grow fast. A tidy one can be understood next month, by you and by others. A messy one becomes hard to change, which defeats the point of building it.

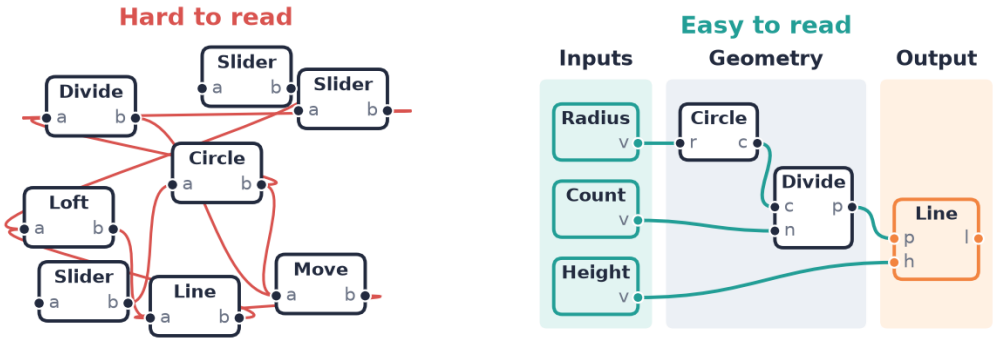


Figure 20. The same kind of logic laid out two ways. On the right, inputs sit on the left, wires flow one way and groups say what each part does.

Habits of clean definitions

- Put all inputs on the left in one labelled group. Give every slider a name and a unit.
- Group components by job (inputs, geometry, logic, output) and give each group a title and a colour.
- Keep wires short and flowing from left to right. Avoid wires that cross the whole canvas.
- Pack repeated logic into a **cluster** or **custom node**, a box that holds its own small graph.
- Add short notes that say *why* a step exists, as well as what it does.
- Test extreme values: the smallest and largest slider positions, zero, and a single item.
- Save versions with a one-line note of what changed.

Common pitfalls

Pitfall	What you see	What to do
Too many parameters	Slow, confusing, endless tweaking	Keep the few that matter and fix the rest
Data structure mismatch	Too many or too few objects, odd pairings	Check list lengths and tree paths at every step
Mixed units	Objects 1,000 times too big or small	Set units at the start and label every slider
Wrong tolerance	Gaps, failed intersections, broken joins	Use the tolerance the maker actually needs
Heavy computation	Sliders lag or the tool freezes	Lower the counts while designing; switch off unused parts
Fragile references	The definition breaks when a drawn object is deleted	Keep inputs inside the definition or on a clearly named layer
Pattern for its own sake	A busy design with no clear purpose	Tie every rule to a reason

WATCH OUT

A parametric model only explores the family you built. If the best idea lies outside it, no slider will find it. Sketch freely first, then build the model.

IN ONE LINE

Name things, group them, flow left to right, test the extremes and give every rule a reason.

15

Tools and Where to Go Next

Choosing a tool, and what to learn after this guide.

The ideas in this guide work in any parametric tool. The buttons differ, while the thinking stays the same.

Tool	Runs inside	Especially good for
Grasshopper	Rhino	Free-form architecture and product design, with a very large range of plug-ins
Dynamo	Revit and other Autodesk tools; also standalone	Automating building information models and linking geometry to building data
Houdini	Standalone	Large-scale procedural modelling, terrain, effects and simulation
Blender Geometry Nodes	Blender	Free, open-source procedural modelling, scattering and visualisation
Python scripting	Most of the above, or on its own	Loops, custom logic, large data and automation

Visual or text?

Visual programming is quick to start and easy to see. Text scripting handles loops, complex logic and large data more easily. Many people start visual and add small scripts when a graph gets clumsy. Most of these tools let you drop a Python box into a graph, so you can mix both.

Where to go next

Four directions build on these foundations.

- **Optimisation.** Let the computer search the design space for the best values, for example with evolutionary solvers such as Galapagos in Grasshopper.
- **Form-finding.** Let physics shape the design, like Gaudí's hanging models, with tools such as Kangaroo.
- **Analysis and AI.** Link the model to daylight, energy or structural analysis (for example Ladybug Tools or Karamba3D) and to machine learning.
- **Digital fabrication.** Send the model to laser cutters, CNC machines and robots.

> **IN ONE LINE**
> Pick one tool, learn its data structures well, and the ideas carry over to every other tool.

16

Try It Yourself

A short recipe, free tools and an eight-week plan.

The fastest way to learn is to build small things and break them on purpose. This recipe brings together ideas from several chapters.

Recipe: your first attractor façade

1. Draw a rectangle 12 units wide and 6 high. Make a grid of 24 by 12 cells and find each centre.
2. Place one attractor point and measure the distance from every centre to it.
3. Remap the distances to 0 to 1 and turn them into radii between 0.03 and 0.22.
4. Draw a circle at every centre, then drag the attractor and watch the pattern follow.
5. Cull circles with a radius below 0.06, then round the rest to a few sizes and count them.
6. Name your sliders, group your components and save the file.

Free and trial tools to practise with

Blender with Geometry Nodes is free and open source, and **Dynamo Sandbox** is a free, standalone Dynamo. **Houdini** and **Rhino** (which includes **Grasshopper**) offer learning or trial versions; check each vendor's current terms. **Python** with NumPy and Matplotlib is free and can rebuild every figure in this guide.

An eight-week plan

Week	Focus	Do this
1	Parameters and rules	Rebuild the staircase with three sliders.
2	Boxes and wires	Rebuild the spokes graph, then write it as code.
3	Lists and trees	Make a grid; practise matching and flattening.
4	Geometry	Divide a curve by count and by length.
5	Attractors	Build the attractor façade; try two attractors.
6	Patterns and panels	Tile with squares, hexagons and Voronoi cells.
7	Logic and chance	Cull by condition; randomise with a fixed seed.
8	Scale and output	Mass a plot by rules; export a schedule.

IN ONE LINE

Build small, break it on purpose, and keep every definition tidy enough to share.

Myths and Facts

Common misunderstandings, cleared up.

Myth	Fact
Parametric design means curvy, futuristic buildings.	Parametric is a method. It works just as well for a staircase, a plain office block or a street layout.
The computer designs it for you.	The computer follows your rules. Choosing the parameters, the rules and the best result is still design work.
You have to be a programmer.	Visual tools let you start without code. Clear logic matters far more than programming syntax.
More parameters make a better model.	Every parameter multiplies the design space and the effort. Good models have a few meaningful parameters.
Random patterns cannot be reproduced.	Pseudo-random numbers come from a seed. The same seed gives the same pattern every time.
Once a model is parametric, every change is free.	Changes inside the family are quick. Changes outside it, such as a new kind of geometry, can mean rebuilding the model.

Quick Quiz

Ten questions. Answers are on the next page.

1 What is a parameter?

2 In the staircase example, why is the number of risers rounded up?

3 How many numbers does a range from 0 to 10 in 5 steps produce?

4 Which way does data flow in a typical visual program?

5 Why can equal steps of t give uneven spacing on a curve?

6 What does remapping do in an attractor definition?

7 How can you check whether a four-sided panel is flat?

8 What is culling?

9 Why should you record the random seed?

10 In the massing example, why does raising the height limit from 24 m to 30 m add only one floor?

Quiz Answers

No peeking until you have tried.

1. An input you can change, usually a number, that other parts of the model depend on.
2. So that no riser is taller than the 180 mm limit.
3. 6: 0, 2, 4, 6, 8, 10.
4. One way: from inputs, through components, to results, usually from left to right. In most tools it may not loop back.
5. The parameter does not move along the curve at a constant speed. Divide by length for even spacing.
6. It rescales the distances to a 0 to 1 range so they can drive any size or angle.
7. Measure the gap between its two diagonals, the warp. Zero means flat.
8. Removing items from a list using a true and false mask.
9. The same seed always gives the same result, so it can be rebuilt.
10. Floors come in whole storeys, and the next one would need 30.1 m.

Glossary

Plain definitions of the words in this guide.

Term	Meaning
Attractor	A point or curve whose distance drives a size or angle.
Bake	Copy a live result into the model as fixed geometry.
Component	A box in a visual program that does one job (a node).
Constraint	A limit the design must respect, such as a height.
Culling	Removing items from a list with a true/false mask.
Curve parameter (t)	A number that runs along a curve, often from 0 to 1.
Data tree	Lists of lists, each stored at a path such as {0;2}.
Definition	A whole visual program of components and wires.
Design space	Every design a parametric model can produce.
Floor area ratio (FAR)	Total floor area divided by the plot area.
List matching	How a component pairs items from unequal lists.
Massing	The overall shape and size of a building.
Mesh	A surface of small faces (triangles or quads) joined at vertices.
NURBS	Smooth curves and surfaces shaped by control points.
Panelisation	Splitting a surface into pieces that can be built.
Parameter	An input you can change that the model depends on.
Plane	A flat reference with an origin, two axes and a normal.
Range	A stretch of numbers divided into equal steps.
Rationalisation	Adjusting a design so it is economical to build.
Remap	Rescale numbers to a new domain, often 0 to 1.
Seed	Start value for pseudo-random numbers, or a Voronoi cell's point.
Setback	A strip inside a plot boundary that must stay clear.
Vector	A direction with a length.
Voronoi pattern	Cells that each hold the area nearest one seed point.
Warp	How far a four-sided panel is from flat.

One-Page Cheat Sheet

Pin it next to your screen.

Idea	In short
Parametric design	Describe relationships; the shape follows. Change an input, everything updates.
Parameter / relationship / constraint	What you can change / how parts relate / what must never happen.
Design space	Every design the model can make. Few, meaningful parameters.
Graph	Components joined by wires. Data flows one way (usually left to right) and, in most tools, never loops.
Range vs series	A range divides a domain; a series counts in fixed steps.
List matching	Shortest, longest or cross reference. Always check the counts.
Data tree	Lists of lists at paths like {0}. Flatten, graft and flip to reshape.
Divide a curve	Equal steps of t can be uneven. Divide by length for even spacing.
Transformations	Move, rotate, scale, mirror, orient. Order matters.
Attractor	Distance, remap to 0 to 1, then size or angle.
Panels	Warp is the gap between diagonals. Count the unique types early.
Logic	A condition makes a mask; the mask culls or chooses.
Randomness	Always set and record the seed.
Output	Label, schedule, agree units and tolerances, export clean files.

Further Reading

Where the ideas come from.

Editions and publishers change over time; check the edition you have before citing.

- Woodbury, R. (2010). *Elements of Parametric Design*. Routledge.
- Tedeschi, A. (2014). *AAD Algorithms-Aided Design: Parametric Strategies Using Grasshopper*. Le Penseur.
- Payne, A. and Issa, R. (2009). *The Grasshopper Primer* (2nd ed.). Robert McNeel & Associates; later editions by Mode Lab.
- Issa, R. (2009). *Essential Mathematics for Computational Design*. Robert McNeel & Associates (later editions exist).
- Burry, M. (2011). *Scripting Cultures: Architectural Design and Programming*. Wiley.
- Schumacher, P. (2009). Parametricism: A New Global Style for Architecture and Urban Design. *Architectural Design*, 79(4), 14–23.
- Jabi, W. (2013). *Parametric Design for Architecture*. Laurence King.
- Pottmann, H., Asperl, A., Hofer, M. and Kilian, A. (2007). *Architectural Geometry*. Bentley Institute Press.
- Terzidis, K. (2006). *Algorithmic Architecture*. Architectural Press.
- Piegl, L. and Tiller, W. (1997). *The NURBS Book* (2nd ed.). Springer.
- Sutherland, I. (1963). *Sketchpad: A Man-Machine Graphical Communication System*. PhD thesis, MIT.
- Reas, C., McWilliams, C. and LUST (2010). *Form+Code in Design, Art, and Architecture*. Princeton Architectural Press.