AN ILLUSTRATED GUIDE

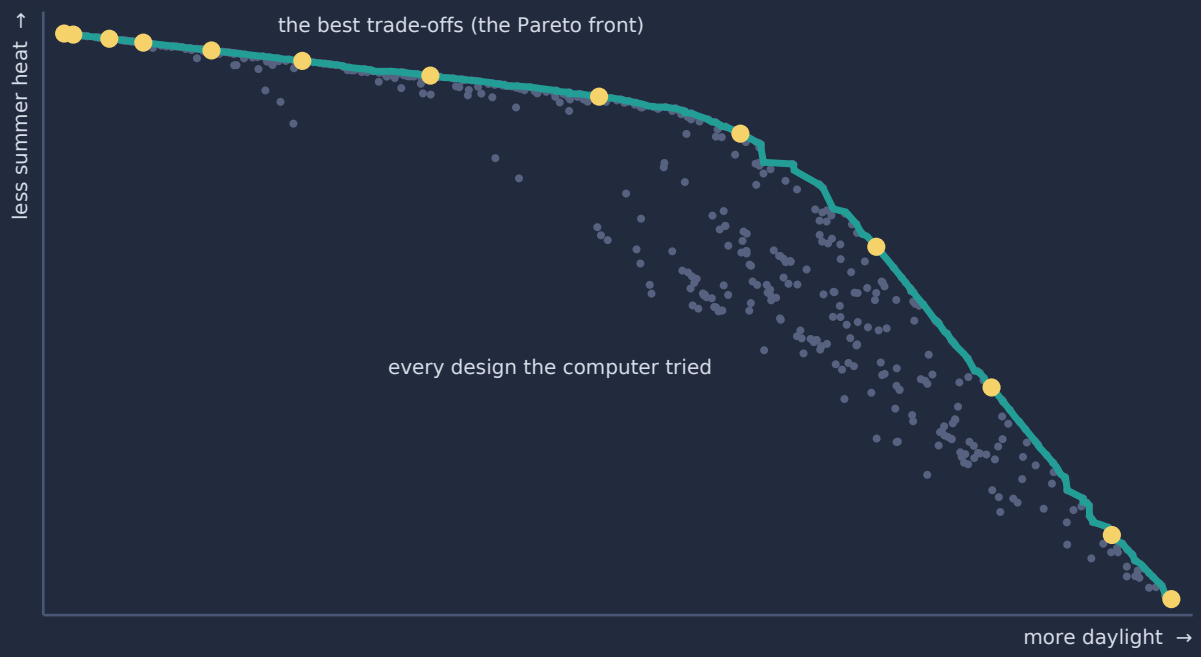
Advanced Digital and Parametric Design

A friendly guide to design that searches, simulates, learns and gets made

Optimisation, simulation, form-finding, AI and fabrication in plain words, with real computed examples, a glossary, a quiz and an eight-week plan.

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

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



How to Use This Guide

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

This guide covers the step after basic parametric modelling. Once a design is described by rules and adjustable numbers, the computer can **search** thousands of versions, **simulate** how each one performs, **learn** quick shortcuts, **find shapes** from physical forces, and send the chosen design to a machine that **makes** it. Every idea starts with a picture and plain words, and no coding is needed.

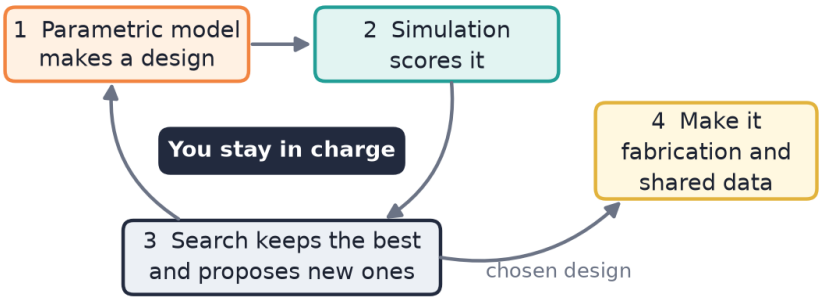


Figure 1. The loop at the heart of this guide. The computer generates, scores and searches; you set the question and make the choice.

Who it is for

- Designers, architects, engineers and students who have built a simple parametric model.
- Anyone who hears words like **optimisation**, **Pareto front** or **surrogate model** and wants a clear picture of each.
- Managers and clients who need to judge what these methods can promise.

How each chapter works

- A **banner** gives the topic in one sentence. Then come plain words, a picture and 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).
- At the back: myths and facts, a quiz with answers, a glossary, a cheat sheet and further reading. Chapter 14 has a recipe, a tool list and an eight-week plan.

Where the numbers come from

The worked examples are **real computations**. The Python script that built this guide also ran every search, simulation and model shown, and every number in the worked examples and figures comes from that run. Random choices use fixed seeds, so the numbers repeat exactly; only timings vary between computers. The models are small toys, much simpler than professional tools, and the text says what each one leaves out.

Contents

Fourteen short chapters and a toolbox at the back.

1 **From Drawing to Searching**
Let the computer try thousands of versions for you.

2 **Objectives, Constraints and Fitness**
Telling the computer what “good” means.

3 **One Goal and the Genetic Algorithm**
Single-objective search, explained with a cooking contest.

4 **Other Ways to Search**
Gradients, annealing and models, in brief.

5 **Many Goals: The Pareto Front**
When improving one thing makes another worse.

6 **Worked Example: A Window With Two Goals**
A real multi-objective genetic algorithm, run for this guide.

7 **Simulation in the Loop**
Performance-driven design with sun, light, energy, structure and wind.

8 **Surrogate Models: Instant Feedback**
Teach a fast stand-in to imitate a slow simulation.

9 **Form-Finding: Let Forces Draw**
Hanging chains, dynamic relaxation and soap films.

10 **Rules That Grow Designs**
Generative design, shape grammars and agents.

11 **AI Images and 3D: Promise and Limits**
What diffusion models do, and what they cannot do.

12 **Designing for Making and Sharing**
Machines, open formats and version control.

13 **Keeping the Designer in Charge**
Pitfalls, honest limits and good habits.

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

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

1 From Drawing to Searching

Let the computer try thousands of versions for you.

In a parametric model, a design is described by rules and a few adjustable numbers called **parameters**. Change a parameter and the model redraws itself. Visual programming tools such as Grasshopper let you build these rules by wiring boxes together on a canvas. If this is new to you, a beginner guide to parametric design is a good first step.

Once a design is a set of numbers, every combination of those numbers is a possible design. The collection of all of them is called the **design space**. Drawing by hand visits this space one point at a time. A computer can visit thousands of points, check each one and report what it found. That is the move from drawing to searching.

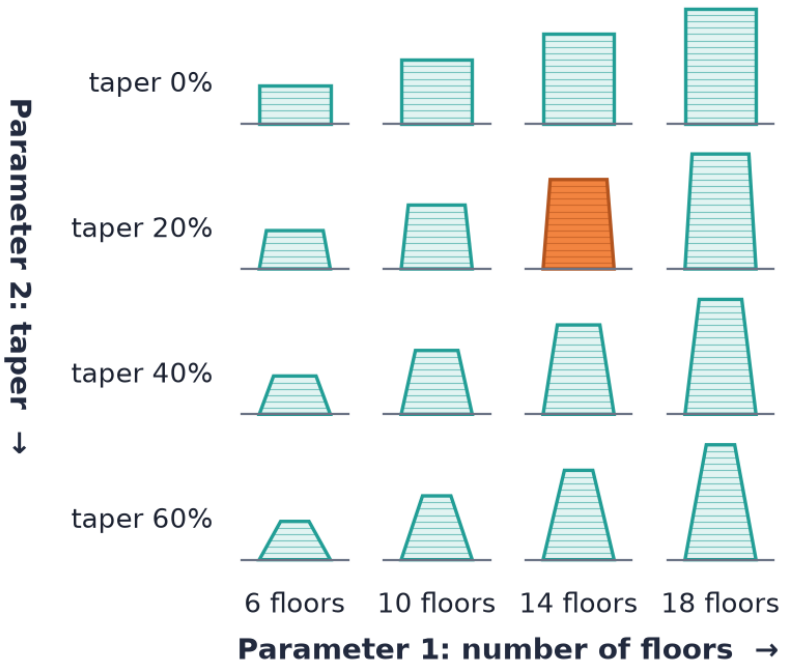


Figure 2. A tiny design space with two parameters. Every cell is one design. The orange tower is one point in this space: 14 floors with a 20% taper.

How big is a design space?

It grows very fast. Three parameters with ten values each give 10 × 10 × 10 = 1,000 designs. Six parameters give a million. A parameter that can take any value in a range, such as a depth anywhere between 0 and 1.5 metres, gives endless designs. So the computer cannot try everything either. It needs a plan for where to look.

Three ways to sample

Sampling means choosing which designs to try. A **grid** tries evenly spaced values. It is easy to understand, but it grows very fast as you add parameters. **Random** sampling picks points by chance, which can leave gaps and clumps. **Latin hypercube sampling** is a tidy middle way. It splits the range of every parameter into equal slices and puts exactly one sample in each slice, so every part of every range gets covered.

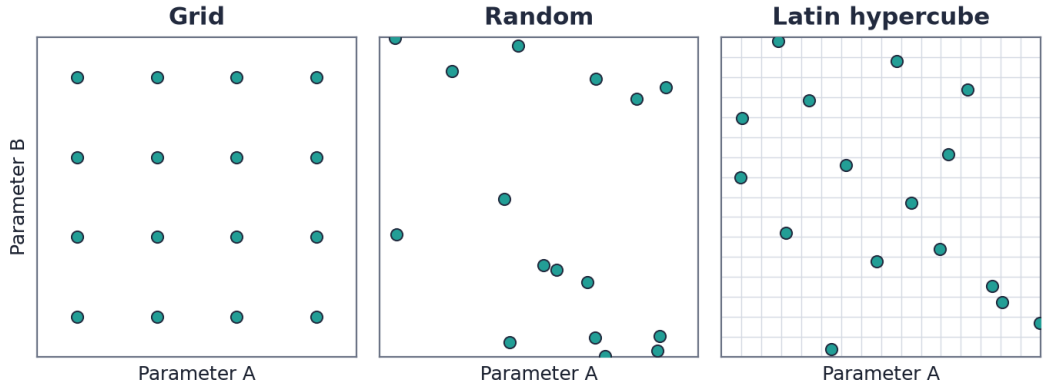


Figure 3. Sixteen samples, three ways. In the Latin hypercube, each of the 16 columns and each of the 16 rows holds exactly one point.

Exploring or optimising?

There are two reasons to search. **Design space exploration** means looking at many options to learn how the parameters behave. Which ones matter? Which ones fight each other? **Optimisation** means hunting for the best design according to a measure you define. Exploration builds understanding and optimisation gives answers. Good projects usually do both, in that order.

TIP

Before optimising anything, run a quick exploration of 50 to 100 samples and plot the results. You will often spot a broken rule or a useless parameter before you spend hours on a search.

IN ONE LINE

A parametric model turns a design into numbers, and numbers can be searched.

2 Objectives, Constraints and Fitness

Telling the computer what “good” means.

A computer cannot search for “a better building”. It needs numbers. Three words describe how we turn wishes into numbers.

- **Objective**: a quantity you want as high or as low as possible, such as more daylight, less energy or less steel.
- **Constraint**: a rule a design must meet to count at all, such as a height limit, a minimum floor area or a budget.
- **Fitness**: the single score the search uses to rank designs. Often it is just the objective. Sometimes it is the objective minus a penalty for breaking a constraint.

Objective	How it might be measured	Better when
Daylight	Share of floor area with enough daylight for most of the year	Higher
Energy use	Heating and cooling energy per year, from a simulation	Lower
Structural weight	Total mass of steel or timber needed	Lower
Views	Share of windows that can see a park, the sea or the skyline	Higher
Floor area	Usable area that fits within the planning rules	Higher

The fitness landscape

Picture every design as a spot on a map. The two map directions are two parameters, and the height of the ground is the fitness. This imaginary terrain is called the **fitness landscape**. Searching for the best design means looking for the highest peak. A landscape with one smooth hill is easy. Real landscapes often have many hills, ridges and flat plains.

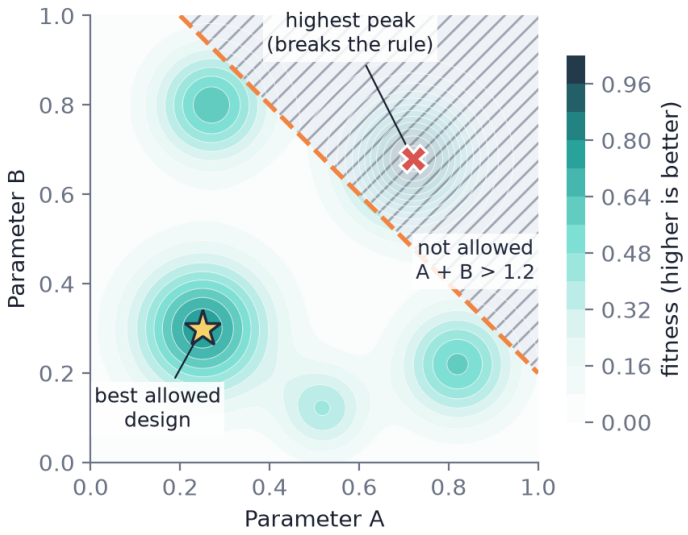


Figure 4. A made-up fitness landscape; darker means fitter. The grey zone breaks a constraint (A + B must not exceed 1.2), so the best allowed design sits on a lower hill. Computed for this guide.

In this example the highest point has a fitness of 1.00, but it lies in the forbidden zone. The best allowed design scores 0.78, on a different hill. So a constraint can move the answer to a completely different place on the map.

Hard and soft constraints

There are two ways to handle a constraint. The strict way throws out every design that breaks it. The gentler way keeps such designs but lowers their fitness by a **penalty** that grows with the size of the breach. Penalties let the search pass through slightly wrong designs on the way to good ones, which helps when the allowed zone is small or oddly shaped. A rule that must never be broken in the final design is a **hard constraint**. A rule you are willing to break at a cost, such as a preferred budget, is a **soft constraint**.

Combining objectives

If you care about two things, you can add them into one score with weights, such as 0.7 × daylight + 0.3 × view. This is called a **weighted sum**. It is quick, but the weights are a value judgement made before you have seen any options. Chapter 5 shows another way.

WATCH OUT

Objectives with different units need scaling before you add them. Adding kilowatt-hours to square metres gives a number, but it means nothing. Scale each objective to a range of 0 to 1 first, and write down the weights you chose and why.

IN ONE LINE

Objectives say what to improve, constraints say what is allowed, and fitness turns both into one score.

3 One Goal and the Genetic Algorithm

Single-objective search, explained with a cooking contest.

Single-objective optimisation means searching for the design with the best value of one number. In the landscape picture, it means finding the highest peak. In practice you cannot see the landscape. Each look costs one evaluation, and with a real simulation one look can take minutes. A good search method finds a high peak with as few looks as possible.

Peaks come in two kinds. The **global optimum** is the highest point anywhere. A **local optimum** is a hilltop that is higher than its neighbours but lower than the global one. A search that only ever walks uphill stops at the first hilltop it reaches, and that may be a local one.

A cooking contest

A **genetic algorithm** (GA) borrows the idea of evolution. Imagine a cooking contest with twenty cooks. Each cook follows a recipe, which is just a list of amounts: flour, sugar, baking time. Judges taste every dish and give it a score. The best recipes are kept. Pairs of good recipes are mixed, taking the flour from one and the sugar and time from the other. Now and then a cook changes one amount a little, just to see. Then the next round begins. After many rounds the recipes are far better than at the start, even though nobody ever worked out the chemistry of baking.

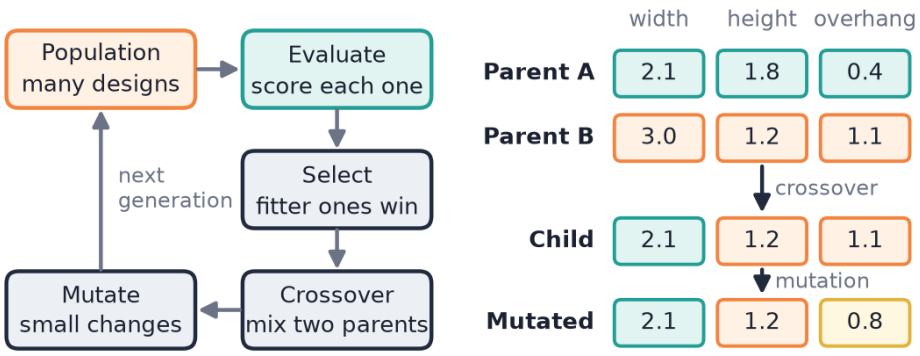


Figure 5. Left: the loop a genetic algorithm repeats. Right: two parent designs, each written as three numbers, make a child, and one value is then nudged.

Evolution word	Design meaning	In the cooking contest
Gene, genome	One parameter; the full list of parameters	One amount; the whole recipe
Population	The designs in the current round	This round's twenty recipes
Fitness	The score from your objective	The judges' score
Selection	Fitter designs become parents more often	The best recipes are kept
Crossover	A child takes some values from each parent	Mix two recipes
Mutation	A small random change to one value	Change one amount a little
Generation	One full round of the loop	One round of the contest

Watching it work

We ran a small GA on the made-up landscape from Chapter 2, this time without the constraint. It used 24 designs per generation. Its crossover blends the two parents' numbers instead of swapping them, a common choice when parameters are continuous. At the start the designs are scattered at random and the best scores 0.702. By generation 3 the best design is within 1% of the true peak of 1.000, and by generation 12 most of the population has gathered on that peak, with a best of 1.000.

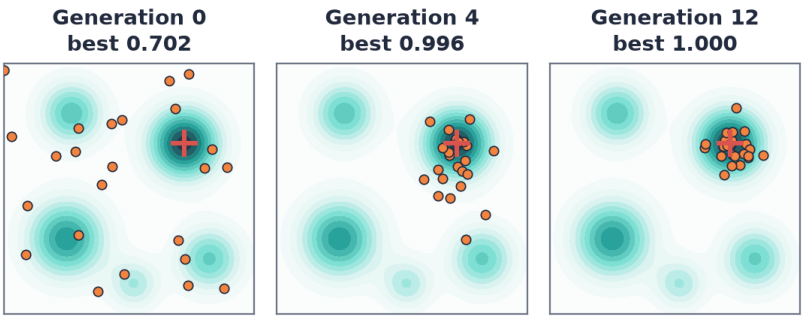


Figure 6. One run of a genetic algorithm on the made-up landscape, computed for this guide. Orange dots are designs; the red cross marks the global peak.

Because the population is spread across the map, some designs land on the slopes of other hills, and this helps the search escape local peaks. There is still no guarantee, so it is good practice to run the search more than once from different random starts. A population of a few dozen designs is a common starting point. In Grasshopper, the built-in **Galapagos** component runs this kind of single-objective evolutionary search on any number your model produces.

IN ONE LINE

A genetic algorithm improves a population of designs by keeping the fittest, mixing them and adding small random changes.

4 Other Ways to Search

Gradients, annealing and models, in brief.

Genetic algorithms are popular in design, but they are one tool among many. Here are the other main families, from the simplest upwards.

Random search and hill climbing

Random search tries designs by chance and keeps the best. **Hill climbing** starts somewhere, tries a small step and keeps it only if it is an improvement. Hill climbing is quick, but it stops at the first hilltop.

Gradient-based methods

A **gradient** is the slope of the landscape. It says which way is uphill and how steep. Gradient-based methods walk straight up the slope. When the landscape is smooth and the slope is cheap to compute, they are very fast and can handle thousands of parameters, which is why engineers use them to size structural members. They need a smooth landscape, and they stop at the nearest peak.

Simulated annealing

Simulated annealing copies the way hot metal is cooled slowly so that its atoms can settle into a calm, orderly arrangement. The search wanders over the landscape. Early on, while the “temperature” is high, it sometimes accepts a step downhill, which lets it walk off a small hill. As the temperature falls it becomes pickier, until it only climbs. Galapagos offers an annealing solver next to its evolutionary one.

Model-based methods

Model-based optimisation (one well-known kind is called Bayesian optimisation) builds a quick stand-in model of the landscape from the designs tried so far, and uses it to choose the most promising design to try next. Each step takes more thinking but far fewer slow simulations, so it suits problems where every evaluation takes minutes or hours. Chapter 8 explains these stand-in models.

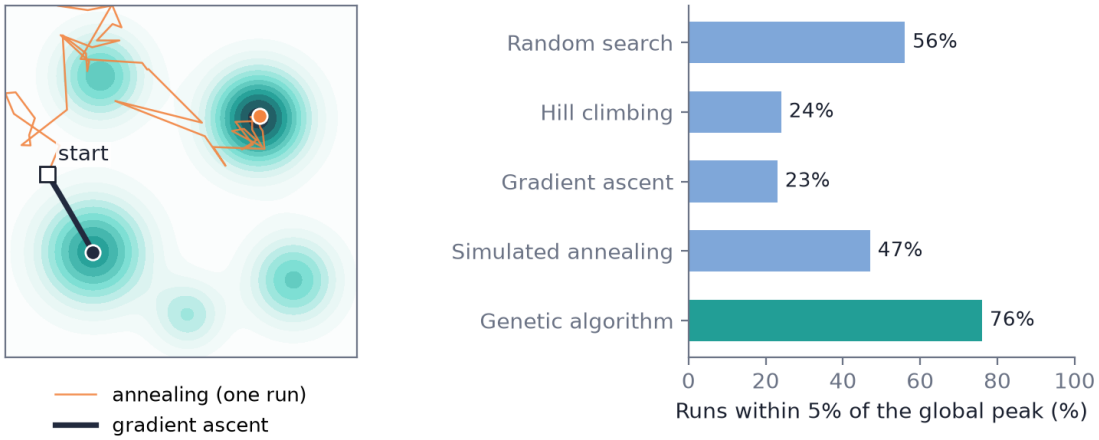


Figure 7. Left: from the same start, gradient ascent (navy) stops on a nearby local peak, while one annealing run (orange) wanders off and finds the global peak. We chose an annealing run that escaped. Right: how often each method came within 5% of the global peak in 100 runs of 300 evaluations each. Computed for this guide.

With a budget of 300 evaluations per run, we counted a run as a success if its best design came within 5% of the global peak's height, which on this landscape means finding the right hill. The genetic algorithm succeeded in 76% of 100 runs, simulated annealing in 47%, hill climbing in 24% and gradient ascent in 23%. Plain random search did surprisingly well, at 56%, because 300 random tries cover a small two-parameter square quite densely. With ten or twenty parameters, random tries spread far too thin and the smarter methods pull ahead.

Method	Good when	Weak when
Gradient-based	The landscape is smooth and slopes are available	The landscape is bumpy or has on/off choices
Simulated annealing	The landscape is bumpy and there is one goal	Each simulation is very slow
Genetic algorithm	The landscape is bumpy, parameters are mixed, or there are several goals	Each simulation is very slow
Model-based	Each simulation is slow and there are not too many parameters	There are very many parameters

WATCH OUT

Averaged over every possible problem, no search method does better than any other. This is a proven result, known as the “no free lunch” theorem. Real design problems are not random, so some methods do suit some problems better, and the only way to know is to try. Try a simple method first, compare it with a smarter one, and keep whichever works for your problem.

IN ONE LINE

Choose the search method to suit the landscape and the cost of each evaluation.

5 Many Goals: The Pareto Front

When improving one thing makes another worse.

Most design problems have goals that pull against each other, such as daylight and summer heat. When goals conflict, the answer is a set of best compromises.

Imagine choosing a flat with low rent and a short commute. If flat A is cheaper than flat B and also closer to work, then A **dominates** B, because it is at least as good on every goal and better on at least one. Cross off every dominated flat, and the ones left form the **Pareto front**, named after the economist Vilfredo Pareto. Along the front, saving money always costs travel time.

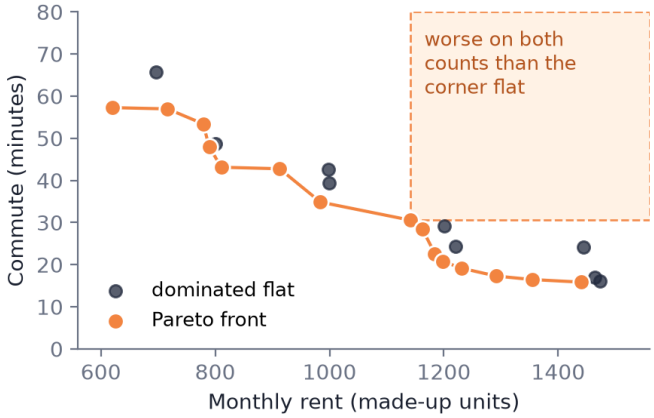


Figure 8. Made-up flats; lower is better on both axes. Orange flats form the Pareto front. Every flat inside the shaded box is worse on both counts than the flat at its corner.

Searching for the whole front

Adding the goals into one weighted score (Chapter 2) means picking the weights before seeing any options, and compromises that sit in a dent of the front can never win. **Multi-objective optimisation** searches for the whole front instead. A widely used method, **NSGA-II** (Non-dominated Sorting Genetic Algorithm II), is a genetic algorithm that sorts designs into layers, with the front itself as the first, and prefers parents from better layers. Within a layer it prefers designs in sparse parts of the front, which keeps the front evenly spread. In Grasshopper, **Wallacei** runs NSGA-II, and **Octopus** runs related multi-objective methods.

SAY IT LIKE THIS

“None of these designs beats the others on both goals. The front shows what each extra unit of daylight costs in heat. Now we choose.”

IN ONE LINE

On the Pareto front, improving one goal always makes another worse.

6 Worked Example: A Window With Two Goals

A real multi-objective genetic algorithm, run for this guide.

Let us put Chapters 3 to 5 to work. We design one window in a south-facing wall, at a latitude of 45° north. We want plenty of daylight, but little summer sun on the glass, because that sun turns into heat indoors. The computer controls three parameters.

Item	Setting
Wall	3.6 m wide and 3.0 m high; the top of the window is fixed at 2.7 m
Parameters	Window width 0.6 to 3.4 m; window height 0.4 to 2.4 m; depth of a horizontal overhang just above the window, 0 to 1.5 m
Objective 1 (higher is better)	**Daylight score**: the glass area, discounted for the part of the sky that the overhang hides from each point of the glass (m^2)
Objective 2 (lower is better)	**Summer sun**: sunlight on the glass, June to August, hour by hour, clear skies (kWh). Direct sun counts only where the overhang's shadow misses the glass. Sky light on open ground is taken as 13% of the direct beam, and the glass receives the share of it that its view of the sky allows.
Search	NSGA-II style genetic algorithm, 40 designs per generation, 30 generations

What happened

The search evaluated 1,240 designs. As a check, we also evaluated a brute-force grid of 15,625 designs (25 values per parameter). A common score for a front is the area between the front and a fixed reference point (here, zero daylight and slightly more than the worst summer sun), called the **hypervolume**. A bigger area means a better front. The random first generation alone reached 69% of the grid front's hypervolume. The GA's front reached 80% after 5 generations and 101% after 30. It can pass 100% because the GA can land between grid steps. So the GA matched the brute force with only 8% of the work.

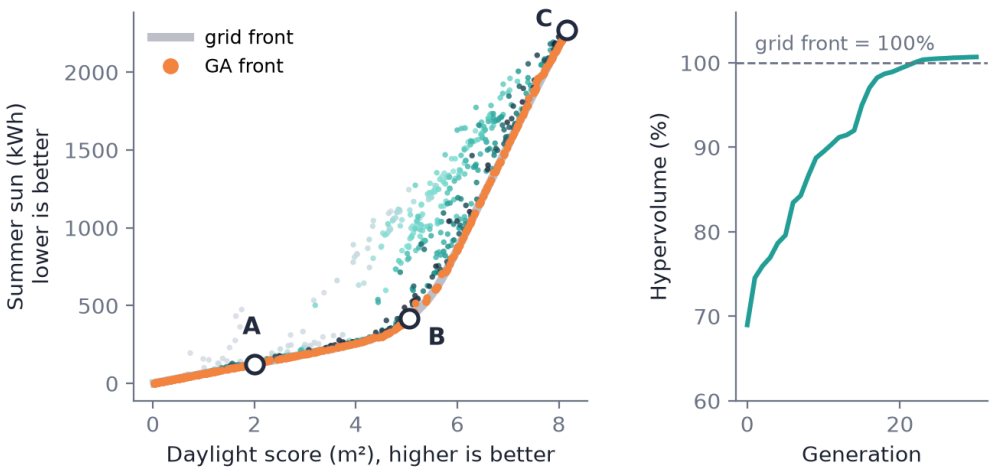


Figure 9. Left: every design the search tried (light grey early, dark late), the GA's final Pareto front (orange) and the front of the brute-force grid (wide grey line). Right: the GA front's hypervolume as a share of the grid front's. Computed for this guide.

Design	Window (width × height)	Overhang	Daylight score	Summer sun
A	2.32 × 1.66 m	1.18 m	2.00 m²	126 kWh
B	3.40 × 2.40 m	1.20 m	5.05 m²	418 kWh
C	3.40 × 2.40 m	0.00 m	8.16 m²	2,273 kWh

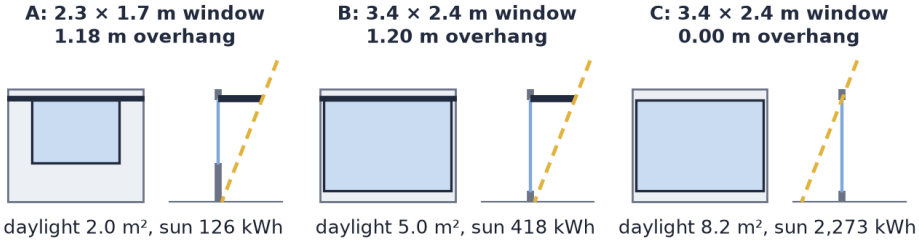


Figure 10. Designs A, B and C in elevation and section (outside is to the right). The dashed yellow line is the noon sun on 21 June, which the overhangs of A and B keep off the glass.

Reading the front

The front bends sharply at design B, a point called the **knee**. Up to the knee, the overhang blocks most of the direct summer sun. For every design on the front up to B, direct sun is at most 24% of the summer sun on the glass. Past the knee, more daylight means a shallower overhang, and direct sun pours in. Design C has a 62% higher daylight score than B but 5.4 times the summer sun, 77% of it direct. Remember that this is a toy model. It ignores glare, winter heat loss, reflected light, cloudy days and blinds, and real projects use validated simulation tools (Chapter 7).

IN ONE LINE

A multi-objective search maps the whole trade-off, and the knee of the front is a good place to start looking.

7

Simulation in the Loop

Performance-driven design with sun, light, energy, structure and wind.

Performance-driven design lets predicted performance help shape the form. The parametric model sends geometry to a **simulation**, a program that predicts how the design will behave in the real world. The results come back as numbers for an optimiser, or as colours painted on the model so that the designer can see them while changing parameters. This is called **simulation in the loop**.

Topic	Question it answers	What the simulation does	Example engines
Sun and shadow	Where and when does direct sun fall?	Traces sun positions through the year	Sun-path geometry
Solar radiation	How much solar energy reaches each surface?	Adds up sun and sky over a year from a weather file	Radiance
Daylight	Is there enough useful daylight inside? Any glare?	Traces light through the rooms	Radiance
Energy	How much heating and cooling over a year?	Balances heat flows hour by hour	EnergyPlus, OpenStudio
Structure	Will it stand? How much does it bend? How heavy is it?	Finite element analysis: splits the structure into many small pieces and solves them together	Karamba3D and others
Wind and airflow	Is it comfortable outside? Does air move through?	Computational fluid dynamics (CFD)	OpenFOAM

Ladybug Tools

Ladybug Tools is a free, open-source collection of plugins that connects parametric models, in Grasshopper and other tools, to well-tested simulation engines. **Ladybug** reads weather files and draws sun paths, wind roses and radiation maps. **Honeybee** builds models for daylight (with Radiance) and energy (with EnergyPlus and OpenStudio). **Dragonfly** works at the scale of districts and cities, and **Butterfly** sets up airflow studies with OpenFOAM. The designer keeps working in the parametric model while serious engines do the physics.

Most of these studies start from a **weather file**, a table of hourly temperature, sun, wind and humidity for a typical year at one location. The EPW (EnergyPlus Weather) format is a common one.

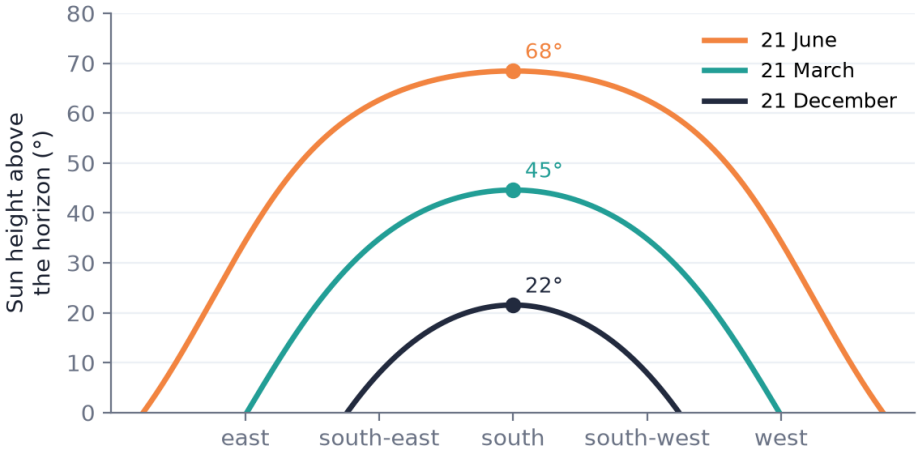


Figure 11. The sun's path across the sky at 45° north on three days, computed for this guide from the standard formulas for the sun's position.

At 45° north, the noon sun stands 68° above the horizon in late June but only 22° in late December. That is why a horizontal overhang can shade a south-facing window in summer and still let the low winter sun in. The window example in Chapter 6 used exactly this sun model, hour by hour.

TIP

Use quick, coarse settings while searching, and check the final few designs with careful settings. Coarse settings are often good enough to rank designs, as long as the winners are checked properly.

WATCH OUT

A simulation is only as good as its inputs, such as the weather file, the materials, how people use the building, and the settings. Chapter 13 returns to this.

IN ONE LINE

Simulation in the loop lets physics score every design, so performance can shape the form from the start.

8 Surrogate Models: Instant Feedback

Teach a fast stand-in to imitate a slow simulation.

Optimisation needs many evaluations. If one daylight or airflow simulation takes ten minutes, a thousand designs take about a week. A **surrogate model** (also called a metamodel) is a quick formula that imitates the slow simulation. You run the slow simulation on a few well-chosen designs and fit the surrogate to those results. From then on, the surrogate answers in a fraction of a second.

Fitting a surrogate is a simple kind of **machine learning**, which means learning a pattern from examples. Common surrogates include **polynomials** (smooth curved surfaces, also called response surfaces), **radial basis functions**, **Gaussian processes** (also called kriging), which also report how unsure each prediction is, and **neural networks** for large data sets.

Worked example

Our slow simulation is the summer-sun model from Chapter 6, written as a plain loop over every hour from June to August. We fix the window width at 2.4 m and vary the window height and the overhang depth. We ran the slow model on 30 designs chosen by Latin hypercube sampling and fitted a cubic polynomial (a formula with 10 terms) to the results. Then we tested it on 625 new designs it had never seen.

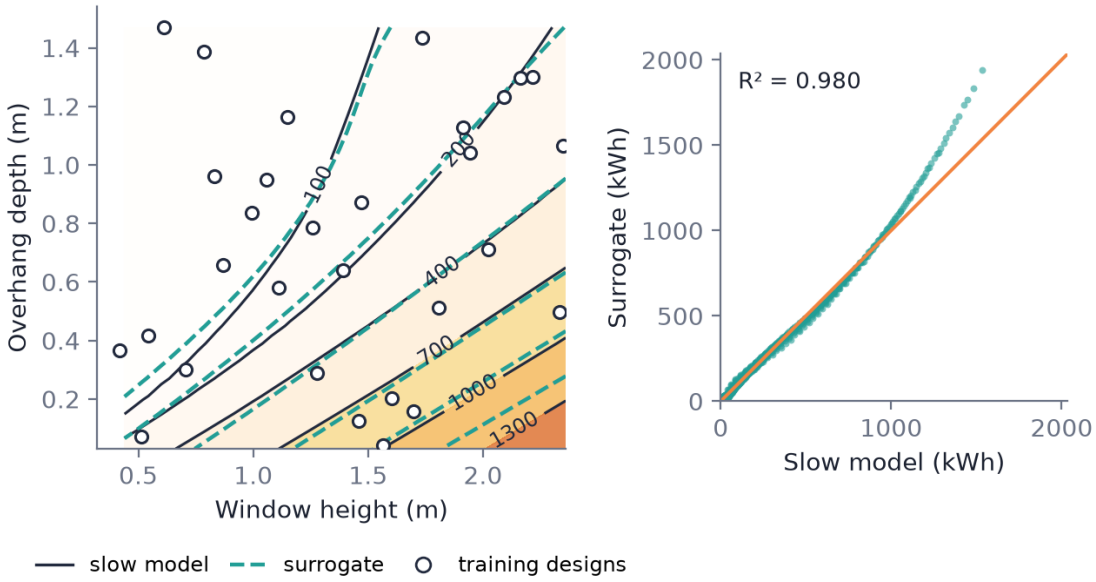


Figure 12. Left: lines of equal summer sun (kWh) from the slow model (navy) and the surrogate (teal, dashed), with the 30 training designs. Right: surrogate against slow model for 625 test designs; perfect predictions would sit on the orange line. Computed for this guide.

Across the test designs the average result was 315 kWh. The surrogate's mean error was 21 kWh and its median error only 11 kWh. Its largest error, 404 kWh, came in one

corner of the map, for a 2.4 m tall window with almost no overhang, where the result is highest and changes fastest. On the computer that built this guide, one run of the slow loop took about 0.88 milliseconds and one surrogate prediction about 10.5 microseconds, roughly 84 times faster. Real simulations often take seconds to minutes, so the saving there is much larger.

Training designs	Mean error, quadratic (6 terms)	Mean error, cubic (10 terms)
10	40 kWh	274 kWh
20	34 kWh	26 kWh
30	33 kWh	25 kWh
60	32 kWh	23 kWh

The table shows the mean error on the same test designs, averaged over 20 different random sets of training designs. The single set of 30 designs used above happened to do a little better than average. More training data helps, and a more flexible formula helps once there is enough data to support it. With only 10 designs, the cubic has as many terms as data points, so it passes through every one and swings wildly in between. Some error remains because the true result bends sharply where the overhang starts to block all direct sun, and smooth formulas struggle with sharp bends.

Where surrogates help

- Instant colours on the model while you drag a slider.
- Fast optimisation: search the surrogate, then check the winners with the real simulation.
- Model-based optimisers (Chapter 4) build and update a surrogate as they go.

WATCH OUT

A surrogate can only be trusted inside the range of designs it learned from. Outside that range it can be badly wrong without any warning. Chapter 13 shows an example.

SAY IT LIKE THIS

“The surrogate is for steering. The final numbers come from the full simulation.”

IN ONE LINE

A surrogate model trades a little accuracy for a huge gain in speed.

9 Form-Finding: Let Forces Draw

Hanging chains, dynamic relaxation and soap films.

Usually we draw a shape and then check whether it can stand. **Form-finding** works the other way round. We set up the materials, supports and loads, and let the forces find the shape. A hanging chain is the classic example. A chain can only pull, so it settles into a curve where every link is in pure tension. Turn that curve upside down and you get an arch in which every part is in pure compression, which suits stone, brick and concrete. Robert Hooke described this idea in the 17th century. Antoni Gaudí used hanging models of strings and small weights to shape vaults and towers, and Heinz Isler hung wet cloth, let it stiffen and turned it over to find shapes for thin concrete shells.

The curve of a hanging chain is called a **catenary**. It looks like a parabola, but it is a slightly different curve.

Dynamic relaxation

Computers find these shapes with **dynamic relaxation**. The model is a set of points, called **nodes**, joined by springs. Loads such as gravity pull on the nodes. In each small time step, every node speeds up a little in the direction of the total force on it and then moves, like a weight on a spring. Some friction, called **damping**, makes the swinging die away. When the forces on every node cancel out, the shape has been found. In Grasshopper, **Kangaroo** does this kind of live physics.

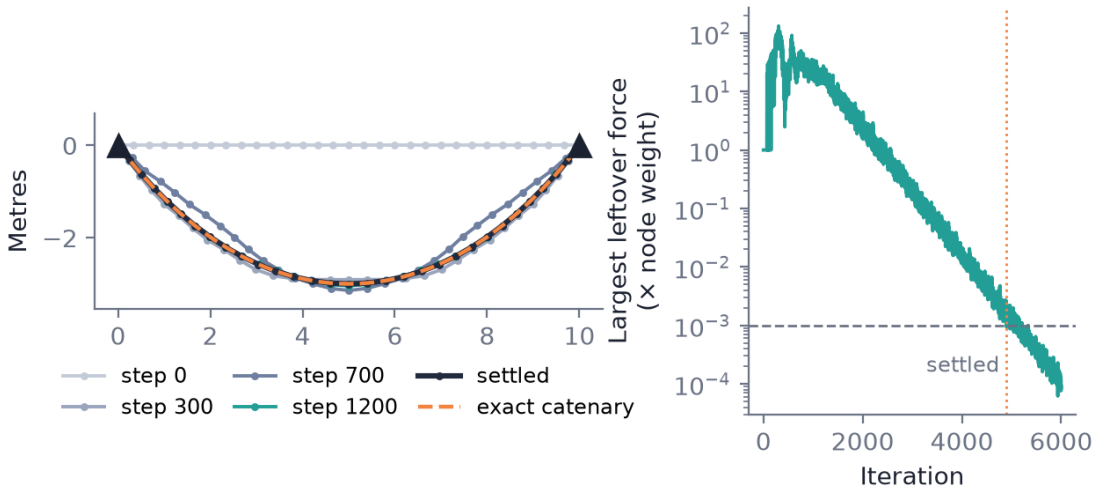


Figure 13. Left: a chain released from a straight line swings down and settles onto the exact catenary. Right: the largest leftover force on any node, step by step. Computed by dynamic relaxation for this guide.

For this picture we hung a chain of 30 links, 12 m long, between supports 10 m apart. By step 4,900 the leftover force on every node was below one thousandth of a node's weight. The chain stretched slightly under its own weight, to 12.08 m. Compared with an exact catenary of that length, the largest gap was 2.0 mm, on a sag of 2.99 m.

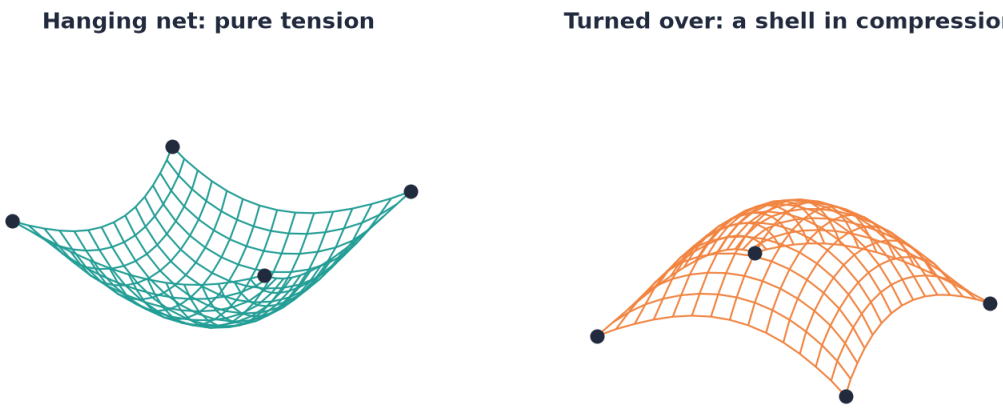


Figure 14. A square net of 14 × 14 cells hung from its four corners (left), then turned over (right). Upside down, it is the shape of a thin shell standing on four points. Computed by dynamic relaxation for this guide.

Minimal surfaces

Dip a wire loop in soapy water and the film settles into a shape that no small change could make smaller in area. Such shapes are called **minimal surfaces**. A soap film has the same tension everywhere, which is just what a fabric roof or tent needs. Frei Otto used soap films and hanging nets in his studies of lightweight roofs and tents. Software finds these shapes with the same relaxation idea, using springs that pull the nodes towards a smaller area.

WATCH OUT

Form-finding gives a good shape for one load case, usually the structure's own weight. Wind, snow, people and uneven loads still need a full engineering analysis.

IN ONE LINE

Form-finding lets forces draw the shape, and the computer does it with springs, small steps and patience.

10 Rules That Grow Designs

Generative design, shape grammars and agents.

Generative design means writing rules or goals and letting the computer produce many designs from them. The term is used in two ways. Sometimes it means rule-based generation, where the rules themselves build the form. In some commercial tools it means goal-driven generation, where software creates many options and ranks them against your objectives, much as in the optimisation chapters. This chapter is about the rule-based kind.

Shape grammars

A **shape grammar** is a set of rules of the form “if you see this shape, you may replace it with that shape”. George Stiny and James Gips introduced the idea in 1972. Like the grammar of a language, a few rules can produce a huge variety of results that still share a family look. Researchers have written grammars that capture the style of particular architects. **Split grammars**, which cut boxes into floors and bays, can generate large numbers of building models quickly. A related idea, **L-systems**, grows branching, plant-like shapes by rewriting strings of symbols.

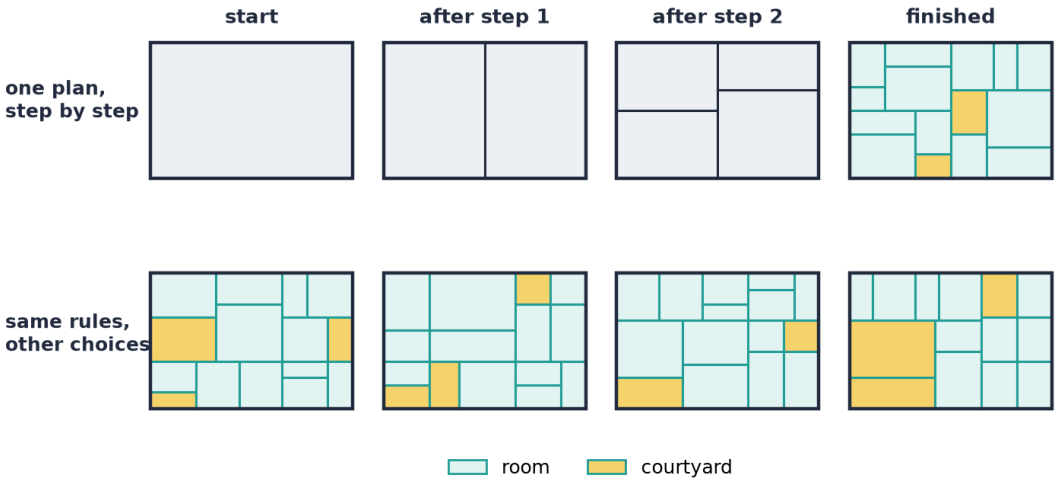


Figure 15. A two-rule grammar, computed for this guide. Rule 1, applied for four rounds, splits each block larger than 40 m² across its long side at 35%, 50% or 65%. Rule 2 then turns each block into a room or, one time in five, a courtyard.

Agents and emergence

An **agent-based model** fills a space with many simple actors, called **agents**, each following a few local rules. No agent knows the big picture, yet patterns appear across the whole group. This is called **emergence**. Flocking birds are the classic example. Craig Reynolds showed in 1987 that three rules (keep a little apart, match your neighbours' direction, stay close to the group) are enough to produce lifelike flocks on screen.

Below is a small agent model run for this guide. 40 walkers cross a lawn between five doors, for 7,000 time steps and 3,460 finished trips. Each walker heads for its door, but prefers ground that others have already trodden. Nobody plans the paths. Even so, shared trails form and merge into a few main routes, like the paths worn into real lawns.

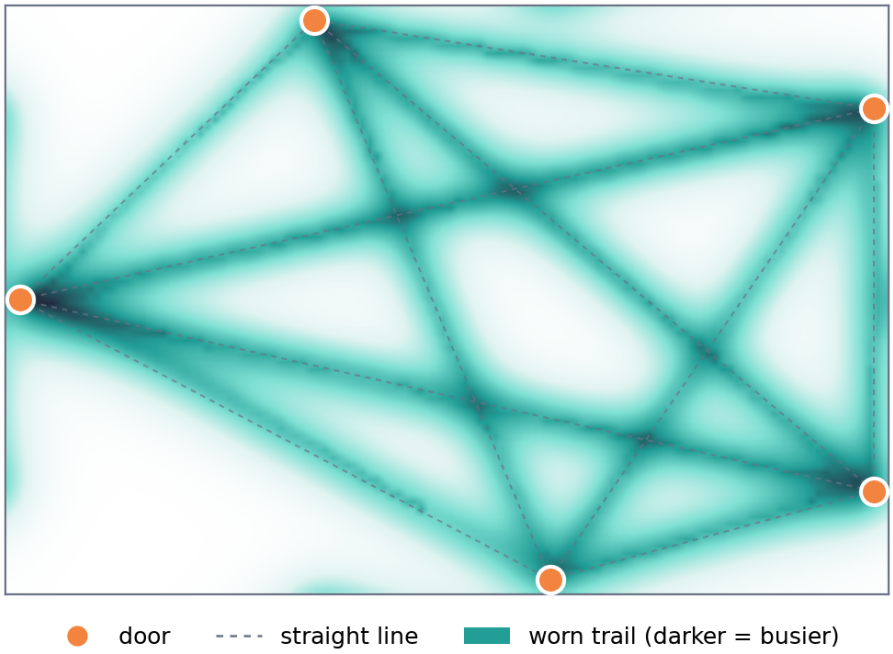


Figure 16. Trails worn by simulated walkers, computed for this guide. Dashed lines show the straight routes between doors. The walkers share and bend their routes instead.

Designers use agents to study crowds and evacuation, to grow branching structures, to lay out paths, and to explore forms that would be hard to draw directly.

WATCH OUT

Emergent results can be fascinating and hard to control, and a small change in the rules can change the outcome a lot. Keep rules simple, fix the random seed while testing, and run several seeds before drawing conclusions.

IN ONE LINE

Generative design writes the rules and lets the forms grow, from grammars that rewrite shapes to agents whose small choices add up.

11 AI Images and 3D: Promise and Limits

What diffusion models do, and what they cannot do.

Since the early 2020s, AI tools have been able to turn a line of text into a convincing image in seconds. Many of these tools use **diffusion models**. Designers now use them for quick mood images, and newer tools try to do the same for 3D models.

How a diffusion model works

During training, the model sees a huge collection of images paired with captions. Each image is covered with random noise, step by step, until nothing is left but static. The model learns to undo one small step of noise at a time, using the caption as a hint. To make a new image, the tool starts from pure random noise and runs the learned clean-up step many times, steered by your text, which is called the **prompt**. A picture slowly appears out of the static.

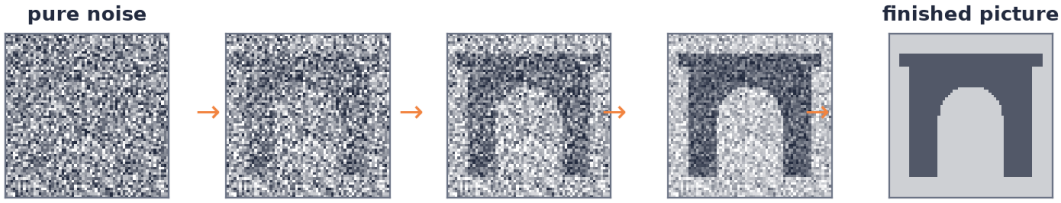


Figure 17. Read from left to right. A trained model removes noise one small step at a time until a picture appears. Here we only made the noisy versions, by adding noise to a simple drawing of an arch; no model was trained.

Good uses and poor uses

Good uses	Poor uses
Mood boards and early atmosphere studies	Plans, sections or anything that must be measured
Quick variations of material, light and season on a sketch or massing image	Structural, technical or safety decisions
Starting a conversation with a client or a team	Proof that a design works or meets the rules
Testing many visual directions in an afternoon	Matching views of the same building from several angles

Honest limits

- **No scale.** The image has no dimensions. Stairs, doors and floors may not fit a human body.
- **No physics.** Nothing has been checked for structure, daylight, energy or fire safety.
- **Invented detail.** The model fills gaps with things that look plausible but may be impossible to build.
- **Bias.** Results lean towards the kinds of images that were most common in the training data.
- **3D is younger.** Tools that make 3D models from text or images tend to produce rough meshes that need cleaning before use. Separate methods, such as photogrammetry and newer neural techniques, rebuild real places from many photographs and are useful for capturing existing sites.

Authorship and rights

Image models learn from millions or even billions of pictures, many made by artists and photographers who were never asked. Laws on copyright and AI differ between countries and are still changing, and who counts as the author of an AI-assisted image is unsettled. Good practice is to check the tool's licence, say clearly when an image is AI-assisted, keep a note of the tool and prompt you used, and never present a generated image as a tested design.

> **SAY IT LIKE THIS**
> “This image was made with an AI tool to explore the mood. Nothing in it has been measured or checked yet.”

> **IN ONE LINE**
> Diffusion models turn noise into convincing pictures, which makes them good for mood and poor for proof.

12

Designing for Making and Sharing

Machines, open formats and version control.

Digital fabrication means making parts with machines driven directly by digital data. The model that was optimised can also produce the machine's instructions, an idea often called **file to factory**. This makes complex, one-off parts far more practical, as long as the design respects how the machine and the material work.

Process	How it works	Rules of thumb for the designer
CNC (computer numerical control) milling	A spinning cutter carves material away (subtractive)	Inside corners cannot be sharper than the cutter. Deep pockets need long tools that can bend.
Laser and water-jet cutting	A beam or jet cuts flat sheets	Design in flat parts, nest them closely to save sheet, and allow for the width of the cut.
3D printing	Material is added layer by layer (additive)	Overhangs that lean more than about 45° from vertical usually need support, and parts are weaker across the layers.
Bending and forming	Sheets, tubes or timber are bent to shape	Each material has a smallest bend radius, and springback (the material springing partly back after bending) changes the final shape.
Robotic assembly	A robot arm places, glues, nails or welds parts	The robot must reach every part, in order, without collisions.

Designing for making

Work to the sizes of real stock material. Allow **tolerances**, small gaps that absorb the difference between model and reality. Plan joints and the order of assembly early, label parts so they can be found on site, and reduce the number of different parts, which is called **rationalisation**.

Worked example: how many different panels?

Below is a curved roof, 24 m by 12 m with a rise of 3 m, divided into 128 four-sided panels. We grouped panels whose four edges and two diagonals all match within a tolerance. Thanks to the roof's symmetry, even at 0.1 mm there are only 32 different panels. If the details can absorb 20 mm, for example with adjustable brackets, the fabricator needs 23 panel types; at 50 mm, which few joints can take, only 12. These counts treat mirror-image panels as one type, which only works if a panel can be used either way round. Warped panels usually cannot, and counting mirror twins as different parts gives 63 types at 0.1 mm and 40 at 20 mm.

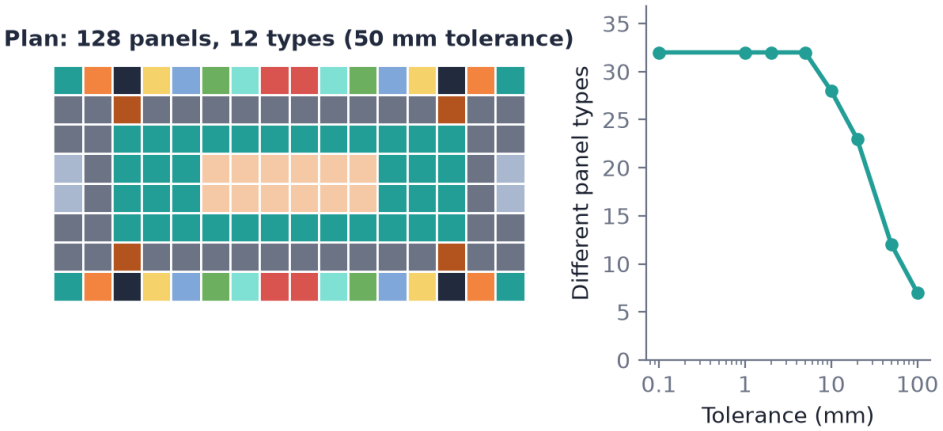


Figure 18. Left: the roof in plan, coloured by panel type at a 50 mm tolerance. Right: how the number of panel types falls as the tolerance grows. Computed for this guide.

Open formats and version control

An **open format** is a file format whose rules are published, so any software can read and write it. **IFC** (Industry Foundation Classes) is the open standard for building information, maintained by buildingSMART. It describes a building as objects with meaning, such as a wall with a fire rating, so people using different software can share one model. Always check what arrives at the other end.

Format	Good for	Notes
IFC	Building information	Open standard (ISO 16739)
OBJ, glTF	Meshes for viewing and the web	glTF is compact and web-friendly
STL	3D printing	Triangles only, with no units
CSV, JSON	Parameters and results	Plain text, easy to compare and version

Version control records every change to a set of files, with who made it and why, so you can return to any earlier state. **Git** is the most widely used system for code and works best with text files, such as scripts, JSON parameters and CSV results. Make results **reproducible** by saving the random seed, the software versions and the parameter file for every run. This guide does the same, so running its script again gives the same numbers. And never keep your only copy of a design in a format that just one program can open.

> **IN ONE LINE**
> Design for the machine and the material, and keep the script, parameters and data in open, versioned form.

13 Keeping the Designer in Charge

Pitfalls, honest limits and good habits.

Optimisers and simulations are powerful, but they only answer the question they are given. The designer decides what the question is, which model answers it and which result gets built. The pitfalls below are common, and each one has a simple defence.

1. Over-optimising a bad objective

If the objective leaves out something important, the optimiser will happily sacrifice it. In Chapter 6, a search for daylight alone would pick design C, the full window with no overhang. It has the highest daylight score, 8.2 m², but receives 2,273 kWh of summer sun, 5.4 times as much as design B at the knee. The optimiser did what it was asked, and the question simply left out heat. A related warning, usually called Goodhart's law, says that when a measure becomes a target it tends to stop being a good measure.

2. Garbage in, garbage out

A simulation fed with the wrong weather file, the wrong materials or careless settings gives confident but wrong numbers. The optimiser then finds the best design for a building that does not exist. Check the inputs, test the model on a simple case you can work out by hand, and compare with a known result where you can.

3. Trusting a surrogate outside its range

Below, a quadratic surrogate was trained on overhangs from 0 to 0.5 m. Inside that range its largest error was only 3 kWh. At an overhang of 1.5 m the real model gives 151 kWh, but the surrogate predicts −372 kWh, which is impossible, because sunlight cannot be negative.

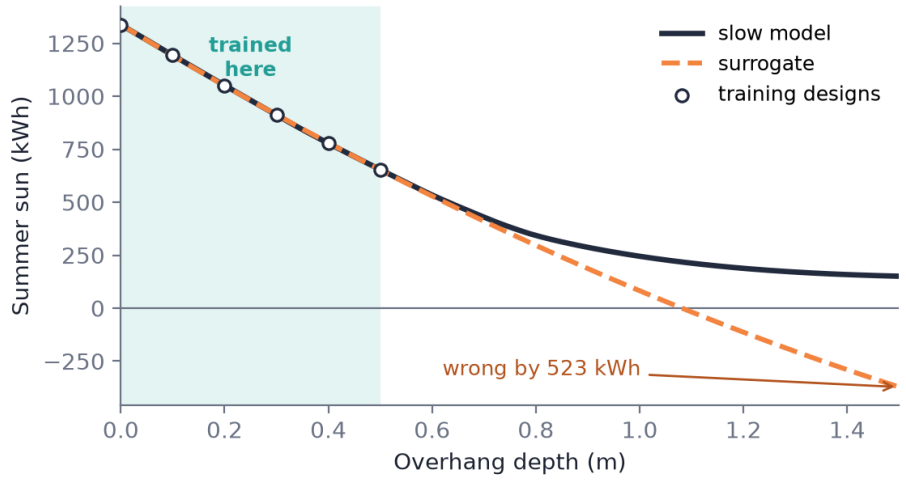


Figure 19. A surrogate trained on a narrow range (shaded) and then used outside it. Computed for this guide.

4. Other traps

- **Local peaks and luck.** One run of a GA or of annealing can stop on a local peak. Run it several times with different seeds.
- **Too many objectives.** With many goals almost every design sits on the front. Group the goals, or turn some of them into constraints.
- **What is not measured.** Beauty, meaning, context and how a place feels are hard to put into numbers. Keep them in the conversation.
- **False precision.** Simulations print many decimal places, but real buildings rarely behave that precisely. Round your numbers when you report them.

Good habits

- Explore before you optimise. Then study the whole front, including the designs near the winner.
- Use the search interactively. Pause it, steer it, add a constraint and run again.
- Sanity-check results with a rough hand calculation.
- Write down the question, the model, the settings and the seed for every run.

> **SAY IT LIKE THIS**
> “The optimiser found the best answer to the question we asked. Let's make sure we asked the right question.”

> **IN ONE LINE**
> The computer searches, simulates and suggests; the designer frames the question and makes the choice.

14

Try It Yourself

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

A recipe for your first study

1. Pick one small question, such as the shape of a sunshade or the massing of a small block.
2. Build a parametric model with two to four parameters, each with a sensible range.
3. Write down one or two objectives and any constraints, with units.
4. Connect a fast simulation, such as sun hours or radiation, and test it on a design you can check by hand.
5. Explore with 50 to 100 Latin hypercube samples and plot the results.
6. Optimise: Galapagos for one goal, Wallacei or Octopus for two or three, or a Python library.
7. Study the front, pick three designs (both ends and the knee) and check them with careful settings.
8. Save the script, parameters, seed and results in open formats, under version control.

Tools

Tool	What it does	Notes
Grasshopper	Visual programming for parametric models	Comes with Rhino, which is commercial
Galapagos	Single-objective evolutionary and annealing solvers	Built into Grasshopper
Wallacei, Octopus	Multi-objective evolutionary search, with tools to study the results	Grasshopper plugins
Opossum	Model-based optimisation for slow simulations	Grasshopper plugin
Kangaroo	Live physics and form-finding	Grasshopper plugin
Karamba3D	Structural analysis inside the parametric model	Grasshopper plugin
Ladybug Tools	Climate, sun, daylight, energy and airflow studies	Free and open source
Python with NumPy, SciPy, pymoo, DEAP and scikit-learn	Scripting, optimisation and surrogate models	Free and open source

Tool	What it does	Notes
Houdini	Procedural modelling with nodes, strong for generative work	Commercial, with a free learning edition
Blender Geometry Nodes	Node-based procedural modelling	Free and open source
Dynamo	Visual programming linked to Revit	Open-source core
IfcOpenShell, Speckle	Reading and writing IFC; sharing data between tools	Free and open source
COMPAS	A Python framework for architecture, structures and fabrication	Free and open source

An eight-week plan

Week	Focus	Do this
1	Parametric refresher	Rebuild a simple model with three parameters. Draw a 4 × 4 grid of its design space.
2	Sampling and exploration	Run 100 Latin hypercube samples. Plot each parameter against a simple measure.
3	One objective	Define a fitness with a penalty for one constraint. Run Galapagos or a short Python GA three times.
4	Many objectives	Add a second, conflicting objective. Find the Pareto front and its knee.
5	Simulation in the loop	Connect sun hours or radiation with Ladybug. Compare coarse and careful settings on three designs.
6	Surrogates	Fit a polynomial to 20 to 30 simulation runs. Test it on new designs inside and outside its range.
7	Form and rules	Hang a chain and a net in Kangaroo or Python. Write a two-rule grammar.
8	Make and share	Rationalise a set of panels, export open formats (IFC, STL, JSON) and put the project under version control.

IN ONE LINE

Start small, keep the loop honest, and add one new layer each week.

Myths and Facts

Common misunderstandings, cleared up.

Myth	Fact
Optimisation finds the best building.	It finds the best answer to the question you encoded, within the model you built. Change the objective, the constraints or the model, and the “best” changes too.
A genetic algorithm always finds the global optimum.	There is no guarantee. GAs are good at broad searches of bumpy landscapes, but a run can still settle on a local peak. Run several seeds.
More objectives give a better design.	With many objectives almost every design ends up on the Pareto front, and choosing gets harder. Two or three well-chosen objectives plus constraints are usually clearer.
A surrogate is as good as the simulation.	It is a quick imitation. It can be close inside the range it learned from and badly wrong outside it. Check the winners with the real simulation.
Form-finding makes structural engineering unnecessary.	It gives an efficient shape for one load case. Other loads, connections, buckling and safety still need a full analysis by an engineer.
AI image generators can design buildings.	They make convincing pictures from patterns in their training data. They do not measure, calculate or check anything.

Quick Quiz

Ten questions. Answers are on the next page.

1 What is a design space?

2 What is the difference between an objective and a constraint?

3 In a genetic algorithm, what do crossover and mutation do?

4 Why can hill climbing miss the best design?

5 Design A has more daylight and less summer sun than design B. What do we say?

6 What is the Pareto front?

7 What is a surrogate model, and when is it worth using?

8 What shape does a hanging chain take, and why does it matter for arches?

9 Name two limits of AI image generators for design.

10 Why save design data in open formats such as IFC?

Quiz Answers

No peeking until you have tried.

1. All the designs a parametric model can produce, one for each combination of parameter values.
2. An objective is something to make as high or as low as possible. A constraint is a rule a design must meet to count at all.
3. Crossover makes a child design by mixing values from two parents. Mutation makes a small random change to one value.
4. It only accepts uphill steps, so it stops at the first hilltop it reaches, which may be a local peak.
5. A dominates B, so B can be dropped whatever your priorities are.
6. The set of designs that no other design dominates. You cannot improve one goal without losing on another.
7. A quick formula that imitates a slow simulation. It is worth it when you need many evaluations and each simulation is slow.
8. A catenary. Turned upside down, it gives an arch that works in pure compression.
9. Any two of: no scale or dimensions, no physics or checks, invented detail, bias towards common images, views that do not match.
10. So that any software can read it, others can check and reuse it, and the work is not locked into one program.

Glossary

Plain definitions of the words in this guide.

Term	Meaning
Agent-based model	Many simple actors (agents), each following local rules.
Catenary	The curve of a hanging chain; flipped, an arch in pure compression.
Constraint	A rule a design must meet to count at all.
Crossover	Making a child design from the values of two parents.
Design space	All the designs a parametric model can produce.
Diffusion model	An AI model that makes images by removing noise step by step.
Dominance	A dominates B if it is better on one goal and no worse on any other.
Dynamic relaxation	Nodes swing under forces, with damping, until they balance.
Emergence	A pattern across a group that no single member planned.
Fitness	The single score a search uses to rank designs.
Form-finding	Letting forces and materials find the shape.
Genetic algorithm	A search that uses selection, crossover and mutation.
Hypervolume	The area between a Pareto front and a reference point.
IFC	Industry Foundation Classes, an open standard for building data.
Latin hypercube	Sampling with one point in each slice of every range.
Local optimum	A hilltop that is not the highest peak.
Minimal surface	A surface whose area no small change can reduce, like a soap film.
Mutation	A small random change to one parameter.
Objective	A quantity to make as high or as low as possible.
Pareto front	The designs that no other design dominates.
Rationalisation	Simplifying a design so it needs fewer different parts.
Shape grammar	Rules that replace one shape with another to make designs.
Simulated annealing	A search that accepts some worse steps early on.
Surrogate model	A fast formula that imitates a slow simulation.
Version control	A record of every change to files, so you can go back.

One-Page Cheat Sheet

Pin it next to your screen.

Idea	In short
Design space	Every combination of parameter values. Search it instead of drawing one design.
Sampling	Grid, random or Latin hypercube. Explore before you optimise.
Objective and constraint	What to improve, and what must be true. Write both down with units.
Fitness	One score for ranking: the objective, lowered by a penalty for each broken rule.
Genetic algorithm	Select, cross over, mutate, repeat. Good on bumpy landscapes.
Other searches	Gradients for smooth landscapes, annealing for bumpy ones, model-based for slow simulations.
Pareto front	The best trade-offs. Look for the knee, then choose.
Simulation in the loop	Geometry → simulation → score → search. Check the inputs.
Surrogate model	A fast imitation of a slow simulation. Trust it only inside its range.
Form-finding	Hang it, flip it. Dynamic relaxation and minimal surfaces.
Generative rules	Grammars rewrite shapes; agents create emergent patterns.
AI images	Good for mood, poor for proof. Record the tool and the prompt.
Making and data	Design for the machine, use fewer part types, keep open formats under version control.
The designer	Ask the right question, check by hand, record seeds and settings.

Further Reading

Where the ideas come from.

Classic sources for the ideas in this guide.

- Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press. The founding book on genetic algorithms.
- Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley. A classic, readable introduction.
- Deb, K., Pratap, A., Agarwal, S. and Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2), 182–197.
- Deb, K. (2001). *Multi-Objective Optimization Using Evolutionary Algorithms*. Wiley.
- Kirkpatrick, S., Gelatt, C. D. and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220(4598), 671–680.
- Wolpert, D. H. and Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1), 67–82.
- McKay, M. D., Beckman, R. J. and Conover, W. J. (1979). A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21(2), 239–245.
- Forrester, A., Sóbester, A. and Keane, A. (2008). *Engineering Design via Surrogate Modelling: A Practical Guide*. Wiley.
- Otto, F. and Rasch, B. (1995). *Finding Form: Towards an Architecture of the Minimal*. Edition Axel Menges.
- Stiny, G. and Gips, J. (1972). Shape grammars and the generative specification of painting and sculpture. In C. V. Freiman (ed.), *Information Processing 71*, 1460–1465. North-Holland.
- Prusinkiewicz, P. and Lindenmayer, A. (1990). *The Algorithmic Beauty of Plants*. Springer.
- Reynolds, C. W. (1987). Flocks, herds and schools: a distributed behavioral model. *Computer Graphics* (SIGGRAPH '87), 21(4), 25–34.
- Kolarevic, B. (ed.) (2003). *Architecture in the Digital Age: Design and Manufacturing*. Spon Press.
- Menges, A. and Ahlquist, S. (eds.) (2011). *Computational Design Thinking*. Wiley.
- Ho, J., Jain, A. and Abbeel, P. (2020). Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems* 33, 6840–6851.