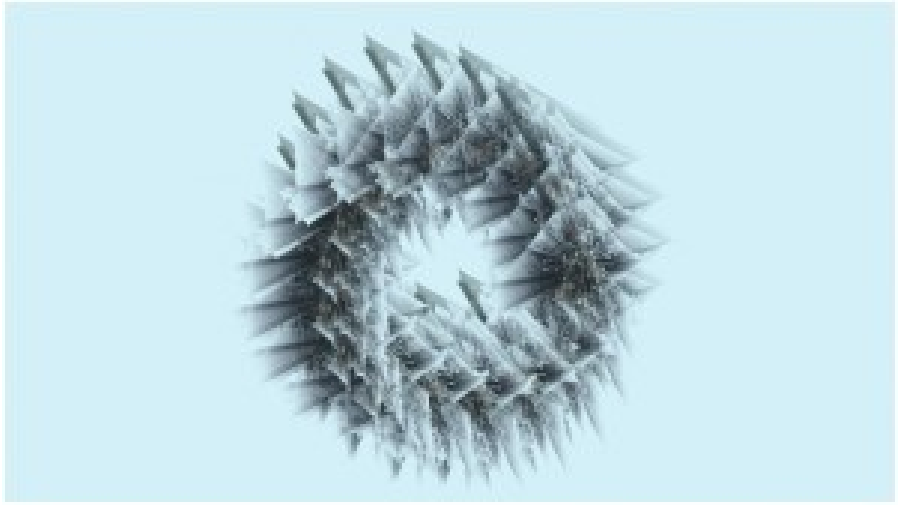
Procedural Generation – not just for games -

Procedural generation is probably most commonly known as a programming method used in games to create diverse environments. It's helpful in generating new worlds, terrains, characters, enemies and so-on for players to interact with, but it can also be used to great effect for practical purposes, and even to create music and art.



Procedurally generated art

I wanted to focus on how procedural generation (PG) has been used for city and building planning, and on how it's possible to combine it with simulations that test how effective the PG algorithm is at producing efficient layouts. By combining PG with a simulation, it's possible to create fully automated design and evaluation: the PG algorithm produces the maps or layouts, which the simulator tests automatically – no human interaction is required.

For a procedural generation algorithm to be as useful as possible in a practical application, you need to carefully define the parameters affecting its output, and what you are trying to measure when evaluating it. When designing the algorithm and simulator it's important to make the output as realistic as possible so that it can actually be applied to the

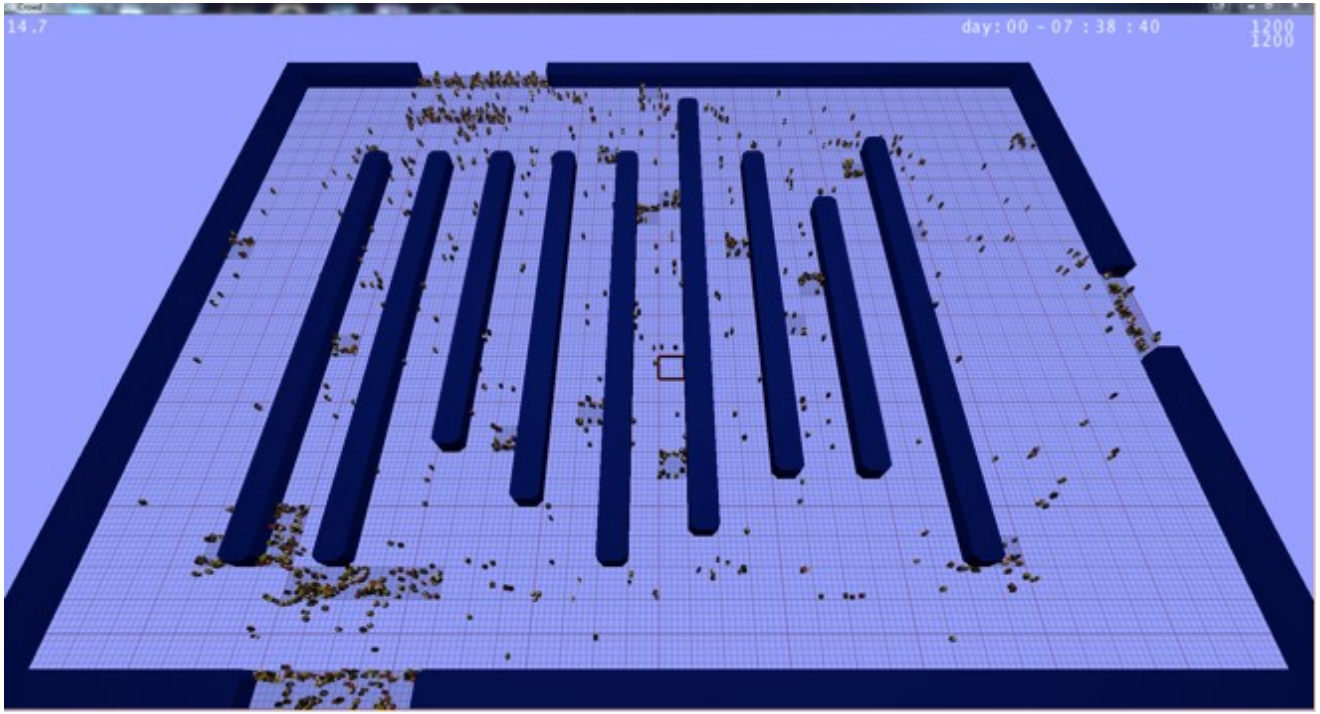
real-life problem you are trying to solve.

What might that mean in practice? Consider using PG to create road layouts for a city. Here, parameters might include the building density, and the width and straightness of roads. The quality of algorithm's output could be tested by measuring how many cars can enter and leave the city in a given time, or how dense any pockets of traffic are in the busiest areas. City planners could then use this data to see what combinations of junctions cause the most delays, or evaluate the difference made by having denser cities.

Catching a flight

During my final year at university my dissertation project was to create a procedural generation algorithm that designed airport terminal layouts. These would be evaluated by an existing crowd simulator, written by a post-grad student.

In my setup, the agents (people) in the crowd simulator were assigned different flight times and had a list of objectives that they needed to complete before they boarded their flight. If they had enough time after this, they would be randomly assigned additional, optional objectives. This meant that each agent would walk around many areas of the airport terminal, which is a more realistic representation of how people actually behave in airports.



Gatwick South Terminal simulation

The simulator originally used a single layout based on Gatwick Airport's South Terminal, so I could compare a real airport design to the output of my PG algorithm. The requirements of the algorithm included placing compulsory objectives such as entrances, departure gates and security checkpoints, as well as the optional stops such as coffee shops and toilets. The algorithm's parameters included whether to place walls in a uniform way (see the image, right) or to scatter them with little logic, and the information gathered by running the simulation was the throughput of people over the course of a day.

What did my project reveal? The gist of the results was that placing straight walls is more efficient than scattering them about. Perhaps that's not all that surprising, but while my straight-wall algorithms performed, on average, about as well as Gatwick's actual layout, their standard deviation was far higher. In other words, they were either a lot better or a lot worse than Gatwick.

One particular map realised a more than 70% higher throughput than the South Terminal's real life layout – potentially a very valuable improvement in efficiency and density. While I'm not suggesting Gatwick's owners rip everything up and start again, that's a good indication of how procedural generation can be a useful approach to solving real-life problems.