



PROCEDURAL CONTENT CREATION

MAKING SOMETHING OUT OF NOTHING

PROCEDURAL CONTENT CREATION IS NOTHING NEW. IT'S A CONCEPT THAT'S BEEN AROUND SINCE the beginning of video games. Even though nowadays hand-crafted content in games is king, we still rely on procedural content quite a bit for heightfield terrains, water surfaces, or particle effects. And, of course, games like *SPORE* rejuvenated the concept and took it a step further by procedurally creating animations and textures.

This month's column delves into the use of procedural content creation in games, and specifically how it was used and what I learned during the development of my game *FLOWER GARDEN* for the iPhone, which uses procedurally-created flowers in pots as its central focus.

WHY PROCEDURAL CONTENT

» The main reason to create content procedurally is that it can be cheap. Really cheap. Generating new content is a matter of turning knobs and moving sliders. At that point, our imagination is the limit of what we can do, and there's very little work involved in creating each asset.

Before any content can be generated though, the code and tools to create it need to be in place. This means that procedural content creation has a steep initial cost, but it flattens out to almost nothing once that technology is in place. In contrast, hand-created content has a much lower initial cost (in the form of exporters and asset pipeline) but a higher-per asset creation cost (see Figure 1). This creates an economy of scale, where procedural content creation is much cheaper for large amounts of content.

Procedural content can also be a very effective form of compression. Just a few bytes or Kilobytes of information are enough to generate intricate, high-resolution textures, or a full galaxy of alien worlds. This is particularly beneficial for downloadable games that need to keep asset size to a minimum. The flip side to this is that generating procedural content on the fly can be a performance-intensive step, so it's a tradeoff between asset size and CPU time. Fortunately, these days we have plenty of idle cores around that can be put to good use. As a plus, procedural generation can usually be parallelized easily, or at least done concurrently with level loading.

Games that use procedural content can also generate nearly-infinite variations of content on the fly to avoid repeating the same asset multiple times. A game with procedural vegetation can generate thousands of variations of the same plants, which will make the landscape look much more natural. Variation can also be used to adapt to the particular situation in the game: time of day, player's past actions, etc. That would all be much more difficult with pre-generated content.

In *FLOWER GARDEN*, the main reason I decided to create flowers procedurally was to keep content creation costs to a minimum. The game centers around growing flowers from seeds, and the final game shipped with 20 different seed types. I later added another 10 bonus downloadable seeds (with more coming). Modeling 30–50 different flowers, plus all the stages of growth would have been very expensive and difficult to iterate with an artist. Creating a totally new seed type from scratch procedurally takes about 10–5 minutes, and is literally a matter of moving some sliders around tweaking DNA parameters.

The other reason I decided to use procedural content creation was to allow for cross-pollination of flowers to create hybrids. This is something that would be almost impossible with pre-generated flowers, but would have been very easy to implement with procedural generation. Unfortunately, this was a feature I put on hold half-way through development, so it was not implemented in the released version. It still remains as an option for a future update though.

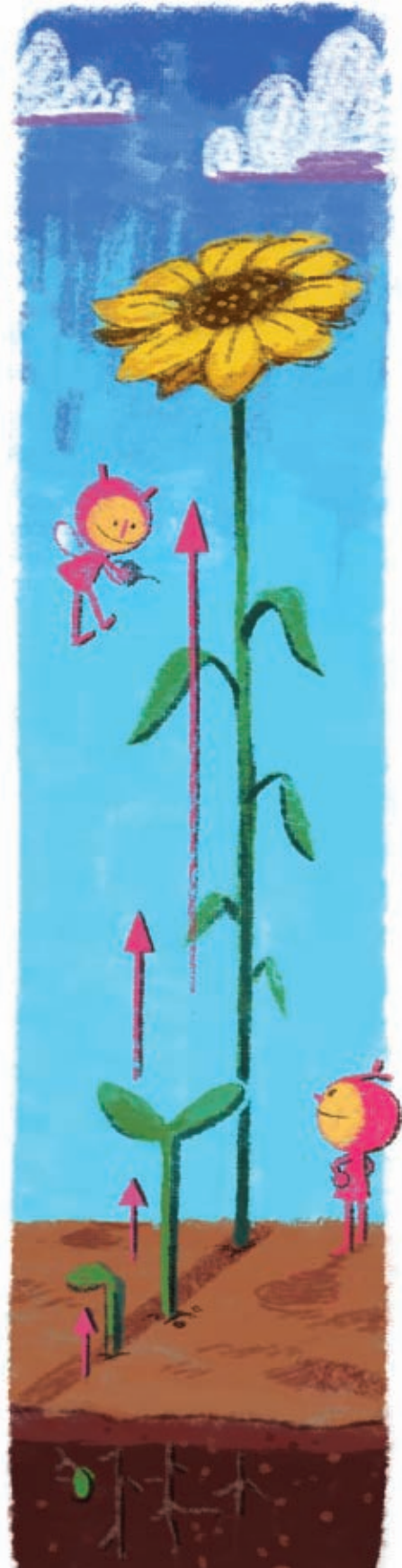


ILLUSTRATION BY MATT BRALY

CONTINUED ON PAGE 42



CONTINUED FROM PAGE 41

Even so, each flower is slightly different than all the others. When a seed is generated, its parameters are tweaked slightly from the reference DNA, so no two flowers are exactly alike.

Content size wasn't a deciding factor, but it helps that the data necessary to fully describe a flower and how it grows weighs in at under 3KB. And that's just the raw size of the binary structure, without any attempts to compress it further.

TYPES OF PROCEDURAL CONTENT

>> Almost any type of content in a game can be generated procedurally. We usually think of meshes, because that's one of the easiest types of content to create—for example, creating a road mesh from a spline laid out in the tool. But it can really be applied to any type of content: textures, animations, and even sounds or speech.

Another important distinction of procedural content is when it happens. On one extreme, content can be generated on the fly, during the game runtime, maybe as frequently as every frame. At the opposite extreme, content can be generated offline by an artist and used as a starting point for a piece of hand-created content, which is then saved and loaded through the static asset pipeline. That distinction is a continuous spectrum, so it's also possible to generate content at load time, or on demand at runtime, but cache it while it's being displayed.

The only type of content that's procedurally generated in FLOWER GARDEN is geometry. Even though the patterns on the petals might look different at first glance, they're just a small set of pre-created texture masks that are used to interpolate between two colors.

What is perhaps surprising is that the full mesh for each flower is re-generated each frame. That's because flowers grow in real time, so they can change appearance from frame to frame. Also, since there's no vertex shader support on the iPhone 3G, all the animation for the stems and petals is done on the CPU, and is based on simple spring systems. It was easy to fold the animation into the last step of the procedural mesh creation, so it happens every frame to achieve a smooth animation.

THINKING PROCEDURALLY

>> It seems that every other word so far has been "procedural," but we haven't defined yet what exactly we mean by "procedural content creation." One possible definition is "content that is created in code," as opposed to hand-created content like I mentioned earlier. That's a bit vague, so let's step back and look at what it really means to generate content procedurally. What is going on when we do that?

It turns out that most forms of procedural content creation are interpolations along some axes. There are some types of procedural content creation that are more purely mathematical, such as Perlin noise or fractals. But the vast majority of procedural creations are based around the idea of interpolating between two extremes. Another way to think about procedural content creation is as a mapping in n-dimensional space that associates input values with a huge space of content. It is by choosing the right axes

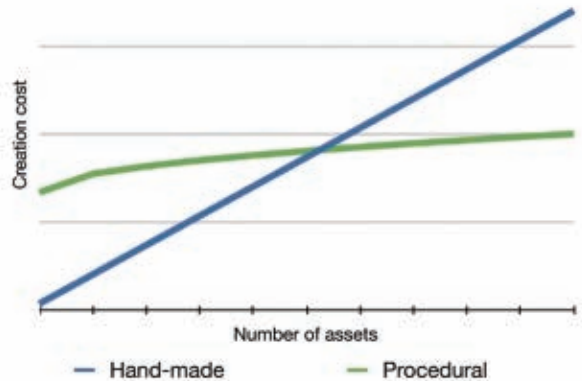


FIGURE 1 Hand created content vs. procedural content costs.

that we're parameterizing all the possible content that can be created.

By that definition, picking a random color for the armor of an enemy would be considered procedural generation. That is example is extreme because we're only interpolating parameters along a single axis. If in addition to color we can change size, armor rating, and decals, we're much closer to what we mean when we refer to procedural content creation.

A more common example is character creation. Many games can change character skin, hair, facial expression, build, clothes, and many other characteristics. In this case we have about 30–40 axes along which to vary our choices. The main limitation is that some of those axes are discrete, with only some limited values in each of them (hair style or clothing).

The flowers in FLOWER GARDEN are created from a DNA-like structure containing information about both their characteristics in full bloom (color, size, number of petals, number of leaves, shapes, and so forth) and the parameters controlling how it grows over time (rate of growth, germination time, or when different parts start growing). Each DNA structure has about 350 parameters that can be tweaked separately. All those parameters—except for the few referring to the texture mask used on the petals and leaves—are analog, and the space of possibilities created by all those axes is staggering.

Before you can write any code that creates content, you need to know really well how to create that content. After all, you can only write a program to do something you know how to do by hand. In my case, I checked out every library book I could get my hands on dealing with flower and plant morphology and learned all I could about the topic. This allowed me to learn new aspects (like the phyllotaxis arrangement of petals), and break the problem down into manageable chunks.

This research phase also allowed me to decide what things not to tackle. For example, I decided to keep plant structure as simple as possible, so I ignored different types of inflorescences and stuck with a single stem with multiple leaves and one flower.

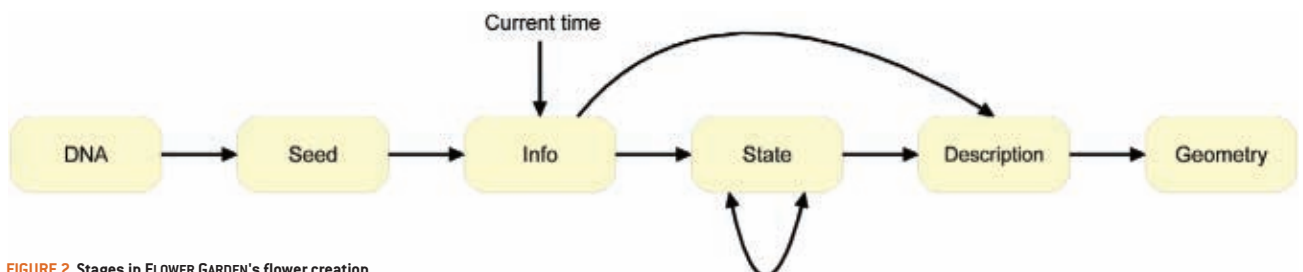


FIGURE 2 Stages in FLOWER GARDEN's flower creation.

Once you're familiar with the vocabulary of the space you want to generate content in, and you've decided which axes you will be using to spawn the space, it's time to start coding.

MAKING THINGS OUT OF THIN AIR

» Now that we completely understand the problem space, we're ready to start writing some code to generate that content based on some inputs. Except that it's not as simple as it sounds. How do we go about creating a plant that is swaying in the wind, half wilted, and about to start blooming? Or how about a large, residential building, a bit run down, and with the roof caved in? As usual, the answer is to break the problem into manageable steps.

I already mentioned the first type of breakdown, which is to tackle each part separately. The plant can be broken down into stem, leaves, petals, sepals, head, body, pistils, and so forth. Finding hierarchical relationships also helps to simplify the problem. For example, a building can be broken down into floors, each of which is made up of different rooms. Each part is more manageable in isolation and they can be assembled together as a final step.

The more interesting type of breakdown is at the creation level. Even once we've narrowed down the problem to creating a single petal, there's still too much data and complexity to go from a simple description all the way to the final geometry format (where is this petal, how bent is it, how shiny is it, what state of fluttering is it in, how grown is it?). To tackle this problem, it helps to identify several stages in the creation process, each of them more detailed and refined than the previous one. Each stage is used to generate the next one, and the final one is the actual geometry to render (or whatever type of asset you're creating).

In *FLOWER GARDEN* there are six distinct stages (see Figure 2).

DNA. Everything starts with a DNA structure, containing about 350 parameters. These parameters are very high level, and describe both the final look of the plant and how it will grow over time: size, color, shapes, arrangement, etc. As an example, in this stage, the stem is described simply in terms of length, radius, and color. Each parameter contains both a starting and an ending value to change as the plant grows.

Seed. From this ideal DNA structure, every time a seed is planted, I create a seed structure, which is identical to the DNA but has some randomness added in some of the parameters (height varies a bit between each plant, but not the number of petals for example).

Current info. Given this seed, and the amount of time passed, a structure containing a high-level description of the plant given its current growth is created. The stem info structure contains the same information as the DNA one, but without any ranges, just the current parameters of the stem at this point in the life of the plant.

Plant state. From the plant information and the previous frame state, a new state is computed. This state contains the simulation results for the spring

systems, as well as the watering information for the plant. The stem state includes information such as current curvature and velocity.

Description. The description is created from both the plant information and the plant state, and it's a more detailed description of the plant, without getting down to the polygon level yet. In this case, the stem description includes an array with joints, each of them with the correct length and orientation.

Geometry. Finally, from the description, we can generate the final geometry that will be rendered.

One advantage of having clearly-defined stages is that we can choose when to update each one. For example, it's clear that we only need to create the seed stage from the DNA once. But we might choose to only update the current info state once every few frames, or not at all even if the plant is not actively growing.

It's important to realize that this is just a conceptual way of thinking about content creation. In practice, you probably will want to keep some context as you're creating each part of the asset instead of treating them in complete isolation. By doing that, you'll be able to reuse potentially expensive calculations you just computed in a previous step (for example, by reusing the transformation of the neighboring petal and just adding an extra rotation to it). It will also allow you to create more efficient meshes by having a broader view of the content you're generating (for example, by combining all petals in a single mesh, or even in a single triangle strip).

LESSONS LEARNED

» This is not news, but it's worth reiterating: Procedural content can be great, but make sure it can be tweaked to the satisfaction of artists and designers. Otherwise you'll end up with worlds full of generic-looking landscapes, forests, or even flowers.

By the time you're done writing the code that generates the content, you're pretty close to an expert in that area. It's important to realize that you know all about the different axes along which your content will be generated, but your brain can't visualize every combination that results from it. Allow yourself time to explore the ranges of what's possible. Crank some of the sliders to the maximum (or beyond the maximum you had initially set), or add a feature to create truly random content by picking random values along each of the axes. I guarantee you'll surprise yourself and will discover things you never thought were possible. I ended up creating some unique flowers by cranking up the curvature of the petals to the point that they would interpenetrate with other petals, forming a very special bell shape.

It all comes down to three things: Learn the problem space, choose good axes to parameterize content, and implement it in small steps. With that, there's nothing you can't create. 🍀



NOEL LLOPIS has been making games for just about every major platform in the last ten years. He's now going retro and spends his days doing iPhone development from local coffee shops. Email him at nllolis@gdmag.com.