Concrete Earth: A DirectX Game

Sam Drysdale

May 16, 2023

Contents

1 Summary

Concrete Earth is a game about nuclear semiotics. Ostensibly, the story is set in a medieval fiefdom, cursed by the occult ciphers long-etched into its landscape. The twist is this is set millenia *after* a nuclear apocalypse - the 'curse' is radiation poisoning, the ciphers the dead language of the previous civilisation, warning of nearby waste repositories. Though it is still in development, this prototype demostrates how the DirectX 11 functionality might bring such a concept to life. Indeed, for a game so concerned with language, it is appropriate that many key features are grounded in linguistics; while the terrain itself is generated by marching cubes, *Concrete Earth* uses grammars to create both procedural screen shaders and a procedural narrative.

A video demonstration can be found within the "docs" directory, and online.

2 User Controls

The user plays as a carriage driver, journeying forwards over an endless map. Periodically, they will be stopped by a 'multiple-choice' scene, where an option must be selected before continuing. Such is the core game loop of *Concrete Earth*.

The application uses the following key mappings:[1]

W .... Move north
A .... Move northwest
D .... Move northeast
1 .... Select option #1
2 .... Select option #2
3 .... Select option #3
Esc .... Quit

Clicking **LMB** will also select an option. The application runs at a fixed 1920×1080 resolution.

3 Features

The following is a technical discussion of the key features of *Concrete Earth*, with a focus on its advanced procedural generation. Mathematical models and code snippets are kept to a minimum, edited for clarity rather than accuracy to the original application.

3.1 Procedural Terrain

Given the concept, this game's terrain is of course intended to evoke a sense of "shunned land" (Trauth et al. 1993). In generating the jutting thorns and blocks so essential to the post-nuclear setting, though, there comes a problem - concavity. Height maps are perfectly adequate for creating rolling hills and valleys, but the fact that each of their (x, z)-coordinates can correspond to only one y-value would prevent any overhangs in *Concrete Earth*'s barren landscape (see Figure 1).
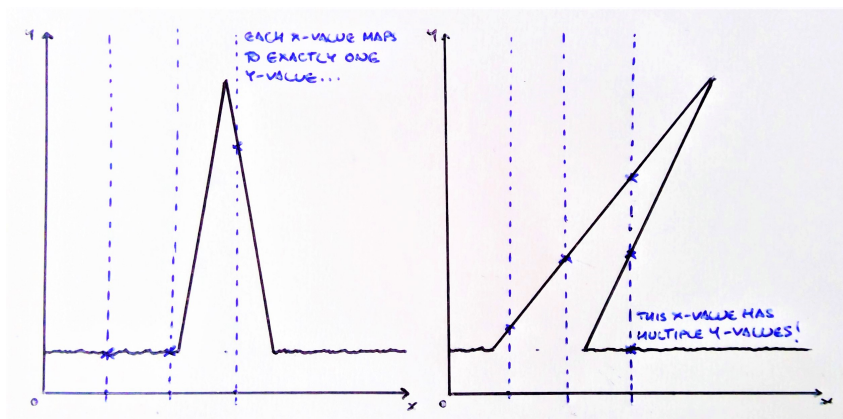

Figure 1: Two sketches of 'thorny' terrain; the former can be generated by a height map, whereas the latter is too concave.
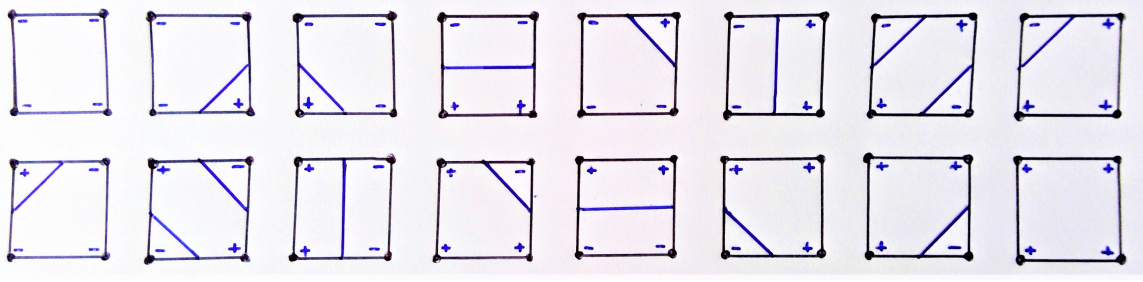
Figure 2: Linear partitions of a marching square. With 4 gridpoints, each existing in one of 2 states ('+' for $f_{i,j} \geqslant c$, '−' for $f_{i,j} < c$), exactly $2^4 = 16$ partitions are possible.
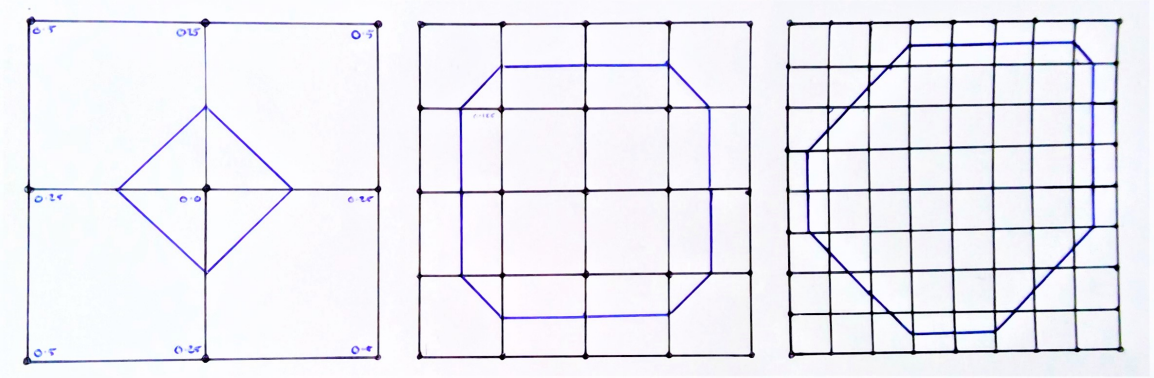

Figure 3: Bounding curve $f(x, y) = 0.16$, at increasing levels of granularity.

Take the two-dimensional analogue of this problem: how does one construct the bounding curve of a (potentially concave, or even disconnected) area? Consider an $n \times n$ square grid, with $n \in \mathbb{N}$,[2] and let a scalar field $f : [0, 1]^2 \to \mathbb{R}$ assign each gridpoint (i, j) a corresponding value $f_{i,j} = f(i/n, j/n)$. Figure 2 shows how to partition a cell so as to enclose the gridpoints with $f_{i,j} < c$ constant. Drawing such a partition into every cell of the grid will therefore approximate the *isoline*[3] $f(x, y) = c$, with increasing accuracy as $n \to \infty$.

This is the *marching squares* algorithm, so named for how it uses a `for` loop to 'march through' and fill in each of the $n \times n$ square cells. Figure 3, for example, illustrates how it might apply to

$$f(x, y) = \min\left((x - 0.5)^2 + (y - 0.5)^2 , 4\left((x - 0.75)^2 + (y - 0.75)^2 \right) \right),$$

a scalar field describing two overlapping circles.

Given adjacent gridpoints $\mathbf{a}$, $\mathbf{b}$ such that $f_{\mathbf{a}} < c < f_{\mathbf{b}}$, Figure 2 would have their partitioning line bisect edge AB exactly. However, knowing the partition represents an isoline, this accidentally implies $f(0.5\mathbf{a} + 0.5\mathbf{b}) \approx c$, which may not be true of the scalar field. Using the standard, first-order approximation that f is linear over short distances

$$f((1 - t)\mathbf{a} + t\mathbf{b}) \approx c, \text{ where } t = \frac{c - f_{\mathbf{a}}}{f_{\mathbf{b}} - f_{\mathbf{a}}}.$$

Understanding that it is therefore more accurate for the partition to intersect AB at $(1 - t)\mathbf{a} + t\mathbf{b}$, such linear interpolation is integrated into the marching squares algorithm (see Figure 4).

---

[1]While the framework registers inputs on press, note that options are only selected *on release*.

[2]This could equally be an $n \times m$ grid; keeping the dimensions the same is just neater.
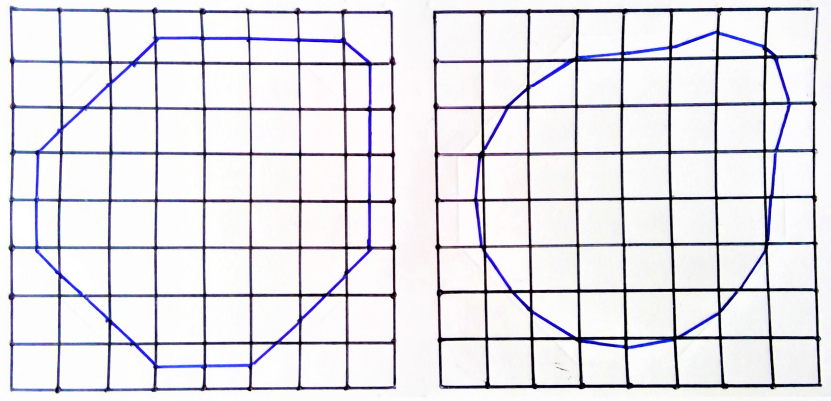
[3]Informally, a 'contour line' of the $2D$ field.

Figure 4: Bounding curve $f(x, y) = 0.16$, before and after linear intepolation.

Returning to the problem at hand, *Concrete Earth* uses *marching cubes* to generate the *isosurfaces* bounding 3D volumes. This requires an $n \times n \times n$ grid, and a new field $f : [0,1]^3 \rightarrow \mathbb{R}$, the algorithm now filling cubic cells with planar surfaces to partition out the gridpoints (i, j, k) satisfying $f_{i,j,k} < c$ (again using linear interpolation to best approximate the 'true' isosurface). Since 3D fields can be hard to visualise, marching cubes are best understood in relation to marching squares. The intuition established above is readily generalised to higher dimensions; whatever the superficial differences, the guiding principle remains the same.

This section will not cover the finer points of the DirectX marching cubes implementation - not how it deconstructs partitioning surfaces into triangles for the vertex/index buffers, nor how it identifies adjacent marching cubes so as to calculate 'smooth' vertex normals[4]. The report is only interested in this technique as it relates to *Concrete Earth*'s terrain. Paul Bourke's *Polygonising a Scalar Field* (1994) remains the definitive source on the algorithm itself.
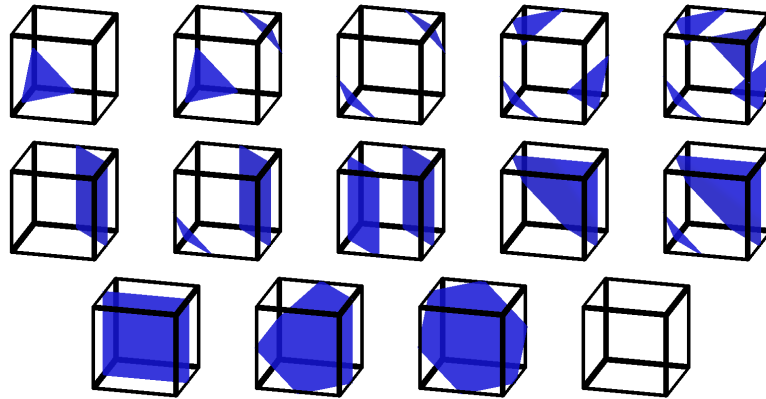

Figure 5: Planar partitions of a marching cube, before linear interpolation. There are 14 fundamental cases, ignoring symmetries; the code itself considers all $2^8 = 256$ possible configurations.

---

[4]A weighted average is taken over the surface normals of the surrounding triangular faces. If a face has vertices $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$, it will be weighted by $1/|(\mathbf{b} - \mathbf{a}) \times (\mathbf{c} - \mathbf{a})|$, proportional to the inverse of its area

Figure 6: Isolines of $\mathbf{Hex}(x, y_0, z)$, a horizontal slice of the hexagonal field.

3.1.1 Case Study: Hexes

Having now established a `MarchingCubes` class, generating a landscape becomes a matter of writing the right scalar field. Indeed, the first step is easyl: for any isovalue $c \in (0, 1)$,

$$\mathbf{Height}(\mathbf{x}) = y + 0.1 \cdot \mathbf{FBM}(\mathbf{x})$$

models a stark, sweeping tundra, its *fractal brownian motion* component scaled to add roughness whilst avoiding 'floating islands'. The application treats the underlying simplex noise as a black box.

Taking a stylised approach, *Concrete Earth* presents its terrain not as a contiguous mass, but as a board of isometric, hexagonal tiles. To that extent, it relies equally on the second field

$$\mathbf{Hex}(\mathbf{x}) = \frac{\sqrt{(2x-1)^2 + (2z-1)^2}\,\cos\left((\theta \bmod \pi/3) - \pi/6\right)}{\cos\left(\pi/6\right)}, \text{ where } \theta = \arctan2\left(2z - 1, 2x - 1\right).$$

For all the trigonometry, this just describes a vertical hexagonal prism centred at $x = 0.5$, $z = 0.5$, much the same as $\sqrt{(2x-1)^2 + (2z-1)^2}$ describes a cylinder (see Figure 6).

$$\mathbf{Terrain}_c(\mathbf{x}) = \max\left(\mathbf{Height}(\mathbf{x}),\, c \cdot \mathbf{Hex}(\mathbf{x})\right)$$

then takes a maximum to enclose only those gridpoints that fall below the isovalue in *both* fields, imposing hexagonal bounds on the existing terrain (the scale factor of c normalising the width of this hex for all isovalues).

3.1.2 Case Study: Landmarks

Physical markers are similarly integrated. To generate one of the afore-mentioned thorns with apex $\mathbf{a}$, base centred on $\mathbf{b}$, and half angle ϕ, for instance, the framework defines field

$$\mathbf{Thorn}_{\mathbf{a},\mathbf{b},\phi}(\mathbf{x}) = \frac{1}{\phi} \arccos\left(\frac{(\mathbf{x}-\mathbf{a})\cdot(\mathbf{b}-\mathbf{a})}{|\mathbf{x}-\mathbf{a}|\cdot|\mathbf{b}-\mathbf{a}|}\right).$$

Using a minimum, isosurfaces of

$$\mathbf{Landmarked\ Terrain}_{c;\,\mathbf{a},\mathbf{b},\phi}(\mathbf{x}) = \min\left(\mathbf{Terrain}_c(\mathbf{x}),\, c\cdot\mathbf{Thorn}_{\mathbf{a},\mathbf{b},\phi}(\mathbf{x})\right),$$

contain all gridpoints below the isovalue of *either* field, adding volume just as a maximum subtracts.[5]

The `Hex` class handling high-level field generation adds a stochastic element, randomising $\mathbf{a}$, $\mathbf{b}$, ϕ. Moreover, any number of thorns can be added; taking the minimum over more than two fields just expands the isosurface further. Perhaps the only surprising choice made in procedurally generating these landmarks is keeping their surfaces completely smooth. Rather than simulating weathering and erosion, the conceit of *Concrete Earth* is that these strange, idealised geometries feel distinctly unnatural. The physical markers of shunned land are man-made designs that *should* tap into a sense of the uncanny, in the hopes of communicating to future civilisations that 'THIS IS NOT A PLACE OF HONOUR'.
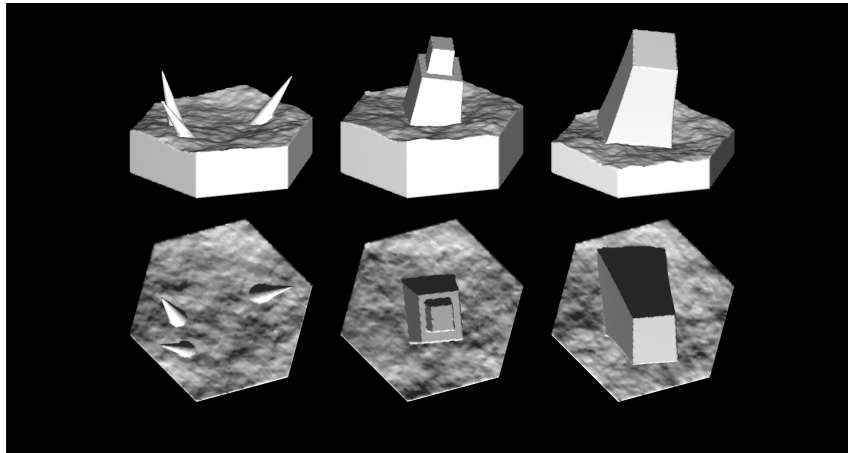

Figure 7: *Concrete Earth*'s landmarks. Three designs have been prototype so far, with more to be added now the terrain pipeline is feature-complete.

3.2 Procedural Screen Shaders

The post-processing in *Concrete Earth* is, in many respects, trivial. `vignette_ps.hlsl`, for instance, demands only two renders-to-texture on every frame: the board itself, and an alpha map of blood vessels sprouting from the edges of the screen. As striking as the final effect is, the shader is surprisingly straightforward in blending the textures into a single, pulsing eye strain overlay; far more worthy of further discussion is how the blood vessels themselves are generated.

---

[5]These operations do not commute! Taking the maximum first allows thorns that 'stick out' from the bounding hex - this is a conscious, if purely aesthetic, decision on the part of the game.

In formal languages, a grammar is a tuple $G = (N, \Sigma, P, \omega_0)$. This contains two disjoint sets of symbols: nonterminals $A, B, \cdots \in N$, and terminals $a, b, \cdots \in \Sigma$. The production rules in P map nonterminals to strings $\alpha, \beta, \cdots \in (N \cup \Sigma)^*$; applied recursively to the axiom $\omega_0 \in (N \cup \Sigma)^*$, these rules can produce increasingly complex *sentences* of terminals and/or nonterminals.[6]

The Chomsky hierarchy (Chomsky 1956) classifies grammars by their production rules:

> *Type-3. Regular grammars* map $A \mapsto a$ or $A \mapsto aB$.
>
> *Type-2. Context-free grammars* map $A \mapsto \alpha$.
>
> *Type-1. Context-sensitive grammars* $\alpha A \beta \mapsto \alpha \gamma \beta$.
>
> *Type-0. Unrestricted grammars* map $\alpha \mapsto \beta$, where α is non-empty.

Note that all Type-3 grammars are also Type-2, all Type-2 grammars also Type-1, and so on.

Suppose, for example, that $N = \{F, G\}$, $\Sigma = \{+, -\}$, $P = \{F \mapsto F + G, G \mapsto F - G\}$, $\omega_0 = F$. Letting ω_n denote the sentences generated by applying the production rules n times, it follows that

$$
\begin{aligned}
\omega_1 &= F + G, \\
\omega_2 &= F + G + F - G, \\
\omega_3 &= F + G + F - G + F + G - F - G, \\
\omega_4 &= F + G + F - G + F + G - F - G + F + G + F - G - F + G - F - G, \cdots.
\end{aligned}
$$

While these definitions are rather abstract, Lindenmayer (1968) provides a remarkable application. Treating each symbol as an instruction like 'go forward' or 'turn right', *L-systems* visualise sentences using turtle graphics; when the sentences have been generated recursively by a grammar, the line drawings will inherit their self-similar structure. Consider the above example - reading non-terminals F, G as 'draw a line while moving 1 unit forwards,' and terminals $\pm$ as 'turn by $\pm \pi/2$ on the spot,' produces dragon curve fractals (see Figure 8).
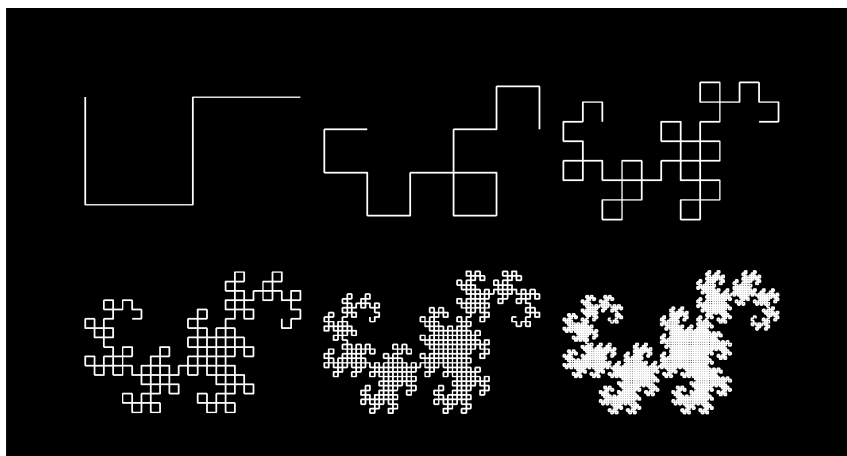

Figure 8: Dragon curves, generated by strings $\omega_2, \omega_4, \cdots, \omega_{12}$.

While L-systems are most common in the modelling of 3D plants and other branching structures (Prusinkiewicz & Lindenmayer 1996), *Concrete Earth* only uses them to generate 2D alpha maps. Moreover, it restricts its attention to L-systems paired with context-free grammars.

---

[6]In mathematical literature, $\omega_0 \in N$ (Hopcroft, Motwani & Ullman 2000), but this project takes an informal approach.

3.2.1 Case Study: Blood Vessels

Generalising the above, *parametric L-systems* act not on symbols A but *modules* $\mathbf{A}(x_1, \cdots, x_n)$. By treating non-terminals as structures containing parameters $x_1, \cdots, x_n$, production rules can then be written to act on said parameters.

This project's own `LSystem` class tracks three main values: the length l, width w, and orientation θ with which to draw each line segment. The benefit is, if nothing else, organisational. Even in the basic L-system for dragon curves, terminals $\pm$ are made redundant by the altogether more readable set of production rules

$$
\begin{aligned}
\mathbf{F}(l,w,\theta) &\mapsto \mathbf{F}(l,w,\theta)\mathbf{G}(l,w,\theta + \pi/2) \\
\mathbf{G}(l,w,\theta) &\mapsto \mathbf{F}(l,w,\theta)\mathbf{G}(l,w,\theta - \pi/2).
\end{aligned}
$$

Zamir (2001), meanwhile, uses this parametric formulation to visualise bifurcations of blood vessels. Suppose a branch with length l, width w bifurcates into two branches M and m, such that $l_M \geq l_m$. Defining the *asymmetry ratio* $\alpha = l_m/l_M$, it follows that

$$
l_M = \frac{l}{(1+\alpha^3)^{1/3}}, \quad l_m = \frac{\alpha \cdot l}{(1+\alpha^3)^{1/3}}, \quad w_M = \frac{w}{(1+\alpha^3)^{1/3}}, \quad w_m = \frac{\alpha \cdot w}{(1+\alpha^3)^{1/3}}.
$$

Furthermore, the branches diverge from their parent at angles

$$
\theta_M = \arccos\left(\frac{(1+\alpha^3)^{4/3} + 1 - \alpha^4}{2(1+\alpha^3)^{2/3}} \right), \quad \theta_m = \arccos\left(\frac{(1+\alpha^3)^{4/3} + \alpha^4 - 1}{2\alpha^2(1+\alpha^3)^{2/3}} \right).
$$

This application is therefore capable of reproducing Zamir's results (see Figure 9), using an L-system with the single production rule $\mathbf{C}(l,w,\theta) \mapsto \mathbf{X}(l,w,\theta)[\mathbf{C}(l_M,w_M,\theta + \theta_M)]\mathbf{C}(l_m,w_m,\theta - \theta_m)$.
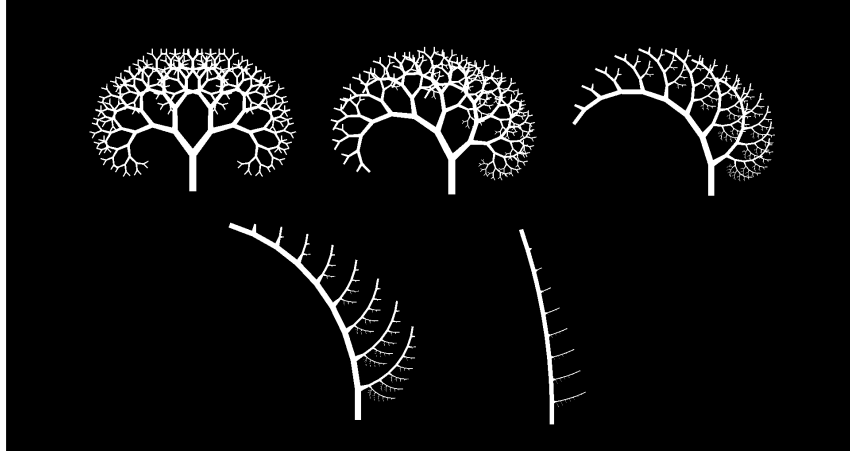

Figure 9: Zamir's model of arterial branching, generated with asymmetry ratios $\alpha = 1.0, 0.8, \cdots, 0.2$. This is an explicit copy of Zamir (2001, Figure 8), recreated within the DirectX framework.

Liu et al. (2010) expand on this research with a further stochastic component - they assign *multiple* production rules to a single non-terminal, choosing which to apply using a weighted probability

distribution. *Concrete Earth* incorporates such randomness into its own rules for blood vessels:

$$\mathbf{C}(l,w,\theta) \xrightarrow[0.45]{} \mathbf{X}(l,w,\theta)[\mathbf{L}(l_M,w_M,\theta+\theta_M)]\mathbf{R}(l_m,w_m,\theta-\theta_m)$$
$$\mathbf{C}(l,w,\theta) \xrightarrow[0.45]{} \mathbf{X}(l,w,\theta)[\mathbf{L}(l_m,w_m,\theta+\theta_m)]\mathbf{R}(l_M,w_M,\theta-\theta_M)$$
$$\mathbf{C}(l,w,\theta) \xrightarrow[0.10]{} \mathbf{X}(l,w,\theta)\mathbf{C}(l,w,\theta)$$
$$\mathbf{L}(l,w,\theta) \xrightarrow[1.00]{} \mathbf{X}(l,w,\theta)\mathbf{C}(l_M,w_M,\theta-\theta_M)$$
$$\mathbf{R}(l,w,\theta) \xrightarrow[1.00]{} \mathbf{X}(l,w,\theta)\mathbf{C}(l_M,w_M,\theta+\theta_M)$$

These describe a capillary with a 45% chance of bifurcating with branch M tacking clockwise, a 45% chance of bifurcating with M tacking anticlockwise, and a 10% chance of extending forwards without any branching. The deterministic production rules on $\mathbf{L}$, $\mathbf{R}$ provide course correction, ensuring the models can be pointed towards the centre of the screen without veering off in either direction. Further informal tweaks can be found in the `LBloodVessel` class itself.

Finally, assigning each `line` segment a (strictly positive) `depth` value and taking the `maxDepth` across all such segments, a normalised `intensity` factor scales the model as follows:

```
for (int i = 0; i < line.size(); i++)
{
    float growth = 1.0f-(1.0f-intensity)*maxDepth/line[i].depth;
    line[i].length *= std::max(0.0f,std::min(growth, 1.0f));
}
```

As `intensity` increases from `0.0f` to `1.0f`, the blood vessels grow, the 'deepest' branches emerging first.[7] Rendering these to texture results in a visceral, procedurally-animated post-process effect, more responsive than passing in a static alpha map.
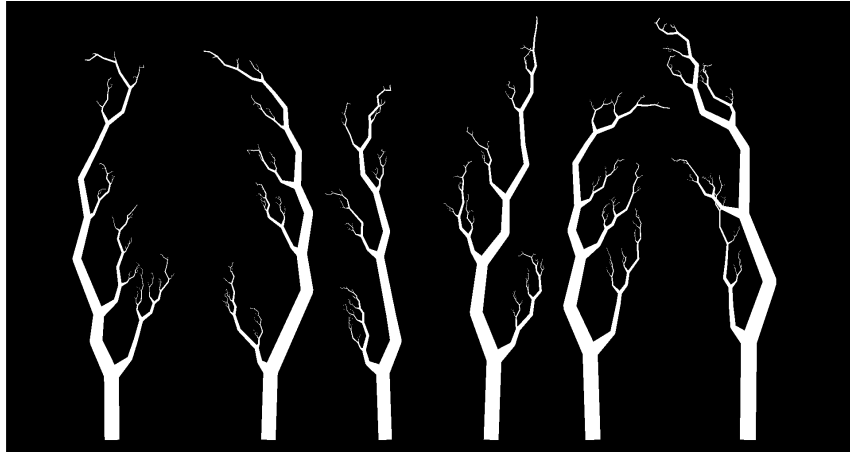

Figure 10: *Concrete Earth*'s stochastic model of arterial branching, generated with different seeds.

3.3   Procedural Narrative

Concrete Earth was originally conceived as a work of interactive fiction, a mix of authorship and procedural text generation. Only a fraction of the content has been written so far, so this section will focus on the intended narrative, and the infrastructure *already in place* to facilitate it.

---

[7]On a more technical level, *Concrete Earth* treats a branch's `depth` as a function of the `length`s of its child branches. Though it came about from a typo in the code, the definition stuck - by delaying growth around bifurcation points, it achieve a nice 'sprouting' effect!

3.3.1 Grammars

Given their origin in linguistics, it is perhaps unsurprising that grammars (see Section 3.2) find use in the field of procedural narrative - *Tracery* (Compton et al. 2015) being a prime example. This is, formally, a stochastic, context-free grammar, with uniformly-distributed production rules for each non-terminal; it iterates a given axiom until all non-terminals (demarcated by {`BRACE`} delimiters) have been replaced. To bemoan the generator as better suited to Twitter bots than interactive storytelling is to misunderstand its design. Compton et al. present a lightweight tool that is left *deliberately* narrow, intended for ease-of-use amongst even the most fledgling authors.

More substantial works, then, adapt *Tracery*'s grammar-based approach to their own specifications. In *The Annals of the Parrigues*, Short uses a consistent, context-sensitive generator that "defines any facts about the world that aren't already defined at the moment of generation" (2015, p. 83). *Voyageur* (Dias 2018) attaches weightings to the distributions of its production rules (Dias 2019). Produceral storytelling is an art form; as much as *Concrete Earth* wears the above influences on its sleeve, to try and consolidate these into an 'optimal', one-size-fits-all generator would be missing the point. This project therefore uses an implementation of *Tracery* with its own custom modifications.

Recency To reduce repetition of a word or phrase, one might consider when it has last been used. The custom grammar treats this as a rank: given a non-terminal with N possible production rules, the most recently used rule will have a *recency*[8] of $N - 1$, the second most recent $N - 2$, and so on down to the rule that was used longest ago (or indeed, never used), with recency 0. Though it does not account for the intervals between uses, nor the second, third, etc, most recent use, there isn't any reason to overcomplicate a metric that already achieves the desired effect (Kazemi 2019).

These ranks are straightforwardly integrating into the production rules' probability distribution. The likelihood of a rule with recency n is scaled by $p^{n/(N-1)}$, such that (all other weightings being equal) the most recently used rule will be p times as likely as the least recent.[9]

Consistency Intended to create a feeling of transience, landmarks in *Concrete Earth* can be visited once and once only, and as such do not need tracked by an overall 'world model'. The game's approach to characterisation is not so simple. With a narrative largely centred around picking up passengers, journeying with them, then parting ways, it should follow that if a detail about a party member has already been established, the text generator must be able to consistently recover it.

The grammar's `GenerateSentence` function, therefore, accepts (up to) two pointers to `StoryCharacter` structs. These are handled by three *consistency markers*:

- **:* The term {`*A`} tells the grammar to check the first character argument for key `A`. If a value exists, this will be substituted [If yes; if else], and stored in the character struct for future use.
- ***:* The term {`**A`} similarly applies to the second character argument.
- *?:* .[10]

Conditionality By their very nature, grammars struggle "to encode high level relationships between different things generated at different points" in a sentence (Compton 2019). [Explainer of *nested non-terminals*]. [Doesn't scale, but author-friendly enough when it only tracks a few key conditions].

Consider how *Concrete Earth*. Using a

---

[8]Or 'dryness', as *Improv*, the generator for *Voyageur*, would call it.

[9]The same string `alpha` might replace two different non-terminals {`A`}, {`B`}. Recognising these as distinct production rules, the recency with which one was applied to {`A`} has no bearing on when the other is applied to {`B`}, or vice versa.

[10]Note that `?` can also inherit an empty string, if the preceding non-terminal is not being remembered.

$$\{*\{*\text{GENDER}\} \text{ FORENAME}\} \{*\{*\text{GENDER}\} \text{ PATRONYMIC}\} \{*\text{SURNAME}\}$$

Concrete Earth's grammar processes this as follows:

1. `{*{*GENDER} FORENAME}` ↦ `{*MASCULINE FORENAME}`: The grammar starts with the nested `{*GENDER}` non-terminal. The asterisk tells it to use the character's established gender, but - finding nothing - it randomly decides whether they are a man, woman, or non-binary, and passes this same value into the sentence and the `StoryCharacter` struct.
2. `{*MASCULINE FORENAME}` ↦ `LEO`: ; even if [never made explicit], the internal consistency
3. `{*{*GENDER} PATRONYMIC}` ↦ `{*MASCULINE PATRONYMIC}`: Now that their gender has been established, the grammar now directly substitutes in the value from the `StoryCharacter` struct.
4. `{*MASCULINE PATRONYMIC}` ↦ `NIKOLAYEVICH`: The character's forename is matched to a patronymic of the same gender ("Nikolayevich" here meaning "Son of Nikolay") [11]
5. `{*SURNAME}` ↦ `TOLSTOY`: The final non-terminal comes with no conditions, and ; this, too, is saved within the

In this case, the grammar terminates after a single iteration. Should `{*BARREL #1}`–`{*BARREL #2}` (signifying a double-barrelled surname) have replaced `{*SURNAME}`, however, the new non-terminals would needed handled on a second parse.

This might feel too limited, from a dramatic perspective. Personality is not immutable; for all this focus on consistency, good storytelling is just as reliant on character *development*. Indeed, as a piece of interactive fiction, the player requires a means of meaningful interaction. The grammar surely requires some sort of override...

3.3.2 Content Selection Architectures

Storylets (Kreminski & Wardrip-Fruin 2018) are [definition].

In *Concrete Earth*, these storylets are envisioned as the means for these [global changes]. [Discuss a beginning-middle-end structure].

The current implementation is limited. [Conditions: randomness]. [Effects: entering/exiting the party].

---

[11]As far as this report can tell, there are no gender-neutral patronymics in Russian - without actually speaking the language, it seems most appropriate to drop these from non-binary characters' names entirely. Cook (2019) offers an invaluable discussion of representation within procedural narrative, and how generators can (whether by accident or authorial intent) reveal prejudices within the societies they create.

4   Code Organisation

This application is, amongst over things, a continuation of previous work. Being built on my framework for CMP502, *Concrete Earth* has been a chance to re-evaluate much of my prior code. Shaders, for instance, were once handled with an unwieldy number of C++ classes: a `LightShader` that passed lighting details into the pixel shader, a `SkyboxShader` passing in an environment map, etc. These have now been folded back into their parent `Shader` class, with `public` functions to turn on the relevant buffers. Though it might seem very basic, identifying and acting on such bad practices early on has been invaluable to the overall development process.

Procedural modelling comes with two key challenges - establishing general, multi-purpose frameworks, then applying these to specific use cases. The code has been organised to reflect this dichotomy. While a single , child classes like `LBloodVessels` and `LDragonCurves`; the description of how `MarchingCubes` should be initialized is generated by and passed in as a separate `Field` class. After initialisation, `Game.cpp` renders these much as it would any models, handling them within the DirextX pipeline using the preferred vertex and pixel shaders.

[The other major point is the board...]

The narrative pipeline is as follows. On every move, the `Board` checks if it should start a new scene. If so, it passes the player's current location (i.e. whether they are near thorns, or a towering monolith) down to the `StoryEngine`, which in turn receives a . ...passes back up to the `Board`, having first replaced any non-terminals via its `Grammar`. The `Board` then tells the `StoryEngine` the player's choice, which returns the corresponding consequence (again having been parsed by `Grammar`).

Text events are rendered to the screen with using the ImGui library - the UI is minimal, handled almost entirely within a single `Game::SetupGUI` function. This application also relies on nlohmann's JSON API.[12] The <u>font</u> and <u>Geiger counter SFX</u> were sourced online; the code has been cited as appropriate.

Finally, *Concrete Earth* was written with good coding practices in mind. Functions and variables follow a consistent naming convention, focused on readability. The DRY principle has been followed, at least within reason. Even the current, informal use of comments proves effective, these breaking down the logic behind key snippets, highlighting potential bugs, and overall acting as incisive memos-to-self[13] (a collaborative project would obviously require a higher standard of documentation). GitHub was used for version control.

---

[12]Indeed, it is a conscious organisational choice to store word banks of terminals and non-terminals within their own self-contained JSON files, rather than hardcode them into the `Grammar` itself.

[13]This is notably untrue of `Game.cpp`. All figures in this report have been expressly produced for CMP505 - though unused in the final application, the code used to render these figures remains, commented out in large and unclear blocks solely for the sake of posterity.

5 Evaluation

5.1 Features

If one element of this project constitutes an unambiguous success, it is the L-systems. *Concrete Earth* provides not only a robust framework, but a genuinely interesting use case. Indeed, [...]. Not that there isn't more to do - the procedural animation of the blood vessels, for instance, still feels quite limited - but the `LSystem` class is in a good place. Above all else, it feels *versatile*, a foundational piece of code that could be carried over to any number of future projects.

The role of marching cubes, by contrast, seems much more narrow: height maps have a well-known limitation, to which this algorithm is the well-known solution (see Section 3.1). Implementing the code was a significant technical challenge, but largely successful, and though the generation process does not run especially quickly[14] a potential solution has already been identified. Bourke's discussion of marching cubes is accompanied by an introduction to *marching tetrahedrons* (1997), a variation of the algorithm that would hopefully improve the efficiency of the code.[15] Working within a tetrahedral grid will come with additional nuances (e.g. cell orientation), but I now feel I've built enough intuition to take on this more advanced technique.

The more interesting question here is how effectively the marching cubes enhance *Concrete Earth*'s setting. As much as they are used to procedurally generate terrain, that terrain can feel somewhat lacking - while this report has already acknowledged the relatively few landmarks available, it's no less notable that they are left untextured. With more time, I would have been keen to write a pixel shader to handle this, but having already experimented with procedural texturing in CMP502 it felt more fruitful to focus on new techniques. Note also that all terrain hexes are generated at runtime; updating these dynamically would be a relatively minor change that greatly reduces repetitivity.

Figure 11: 3D Voronoi noise. Experiments with voxel textures did not make it past initial prototyping.

To the extent that *Concrete Earth* has any single 'special feature', though, it would surely be its approach to procedural storytelling. While there's a lot , this has not exactly . For all my interest in *coding* narrative frameworks, I am not myself a writer - a fact reflected in the lack of content available to the grammar. [MORE CONTENT = MORE TESTING]. [Genuinely pleased with the functionality of the grammar]; it is a shame that functionality has not been fully utilised.

---

[14]Not on a low-end laptop, at least!

[15]Where there are 256 possible partitions of a cube, tetrahedrons only come with $2^4 = 16$...

That said, I am genuinely pleased with how the `Grammar` functions - not so much the surrounding `Architecture`. The same cannot be said of the surrounding `Architecture`. [The priority was getting a serviceable game loop in place].

5.2 Code Organisation

This relates, in part, to a broader organisational problem. [An architecture that grows *with* the content, as my alternative strategy.]

[That's not to detract from... - bridge!]. [While the end result of the architecture feels unstructured, overly reliant on special cases... - the rest of the code is good!]

More than anything else, I've genuinely tried to . [...]. For instance, [...performance!].

6 Conclusions

[General lessons]

[Features vs. content throughline]

[Last: iteration on CMP502...]

References

Bourke, P. (1994), 'Polygonising a Scalar Field', Available at: `http://paulbourke.net/geometry/polygonise/`. (Accessed: 9 February 2023).

Bourke, P. (1997), 'Polygonising a Scalar Field Using Tetrahedrons', Available at: `http://paulbourke.net/geometry/polygonise/`. (Accessed: 9 February 2023).

Chomsky, N. (1956), 'Three Models for the Description of Language', *IRE Transactions on Information Theory* **2**(3), 113–124.

Compton, K. (2019), 'Getting Started with Generators', *in* T. Short & T. Adams, eds, '*Procedural Storytelling in Game Design*', 2nd edn, Boca Raton, FL, USA: Taylor & Francis, pp. 3–17.

Compton, K., Kybartas, B. & Mateas, M. (2015), Tracery: An Author-Focused Generative Text Tool, *in* '*8th International Conference on Interactive Digital Storytelling*', Copenhagen, Denmark: 30 November-4 December, pp. 154–161.

Cook, M. (2019), 'Ethical Procedural Generation', *in* T. Short & T. Adams, eds, '*Procedural Storytelling in Game Design*', 2nd edn, Boca Raton, FL, USA: Taylor & Francis, pp. 49–62.

Dias, B. (2018), *Voyageur*, self-published.

Dias, B. (2019), 'Procedural Descriptions in *Voyageur*', *in* T. Short & T. Adams, eds, '*Procedural Storytelling in Game Design*', 2nd edn, Boca Raton, FL, USA: Taylor & Francis, pp. 193–207.

Hopcroft, J., Motwani, R. & Ullman, J. D. (2000), *Introduction to Automata Theory, Languages, and Computation*, 2nd edn, Boston, MA, USA: Addison-Wesley.

Kazemi, D. (2019), 'Keeping Procedural Generation Simple', *in* T. Short & T. Adams, eds, '*Procedural Storytelling in Game Design*', 2nd edn, Boca Raton, FL, USA: Taylor & Francis, pp. 17–22.

Kreminski, M. & Wardrip-Fruin, N. (2018), Sketching a Map of the Storylets Design Space, *in* '*11th International Conference on Interactive Digital Storytelling*', Dublin, Ireland: 5-8 December, pp. 160–164.

Lindenmayer, A. (1968), 'Mathematical Models for Cellular Interactions in Development II. Simple and Branching Filaments With Two-Sided Inputs', *Journal of Theoretical Biology* **18**(3), 300–315.

Liu, X., Liu, H., Hao, A. & Zhao, Q. (2010), Simulation of Blood Vessels for Surgery Simulators, *in* '*2010 International Conference on Machine Vision and Human-machine Interface*', pp. 377–380.

Prusinkiewicz, P. & Lindenmayer, A. (1996), *The Algorithmic Beauty of Plants*, 2nd edn, Berlin, Germany: Springer-Verlag.

Short, E. (2015), *The Annals of the Parrigues*, self-published.

Trauth, K., Hora, S. & Guzowski, R. (1993), Expert Judgment on Markers to Deter Inadvertent Human Intrusion Into the Waste Isolation Pilot Plant, Technical report, SANDIA National Laboratories, Albuquerque.

Zamir, M. (2001), 'Arterial Branching Within the Confines of Fractal L-System Formalism', *The Journal of General Physiology* **118**, 267–276.