Untitled: A DirectX Game

Sam Drysdale

May 16, 2023

Contents

1 Summary **1**

2 User Controls **1**

3 Features **1**

3.1 Noise . . . 1

3.2 Procedural Terrain . . . 1

3.2.1 Case Study: Hexes . . . 2

3.2.2 Case Study: Landmarks . . . 2

3.3 Procedural Screen Textures . . . 2

3.3.1 Case Study: Blood Vessels . . . 2

3.3.2 Case Study: Runes . . . 2

3.4 Procedural Narrative . . . 2

4 Code Organisation **2**

4.1 Post-Processing . . . 2

4.2 GUI . . . 2

5 Evaluation **3**

5.1 Features . . . 3

5.2 Code Organisation . . . 3

6 Conclusions **3**

References **3**

1 Summary

2 User Controls

3 Features

3.1 Noise

3.2 Procedural Terrain

[Starting point: the problem of concavity!]

Introduce marching cubes as the central tenet of the modelling process...

Definitive... Paul Bourke's *Polygonising a scalar field* (1994)...

3.2.1 Case Study: Hexes

3.2.2 Case Study: Landmarks

3.3 Procedural Screen Textures

[*Untitled* uses screen textures, based on l-systems...].

In formal languages, a *grammar* is a tuple $G = (N, \Sigma, P, \omega_0)$. This contains two disjoint sets of symbols: *nonterminals* $A, B, \dots \in N$, and *terminals* $a, b, \dots \in \Sigma$. The *production rules* in P map nonterminals to strings $\alpha, \beta, \dots \in (N \cup \Sigma)^*$; applied recursively to the *axiom* $\omega_0 \in (N \cup \Sigma)^*$, these rules can produce increasingly complex strings of terminals and/or nonterminals.[1]

The Chomsky hierarchy [citations] categorises grammars, according to the form of their production rules:

Type-3. Regular grammars map $A \mapsto a$ or $A \mapsto aB$

Type-2. Context-free grammars map $A \mapsto$

Type-1. Context-sensitive grammars

Type-0. Unrestricted grammars map A

Note that [subsets].

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 string generated by applying the production rules n times, it follows that

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

[Introduce basics of L-systems, include the angles used for dragon curves...] produces the dragon curves in Figure 1.

3.3.1 Case Study: Blood Vessels

3.3.2 Case Study: Runes

3.4 Procedural Narrative

4 Code Organisation

4.1 Post-Processing

4.2 GUI

[Include HDRR/bloom here...]

[1] In mathematical literature, $S\omega_0 \in N$ [citation], but this paper is happy with a more informal approach.

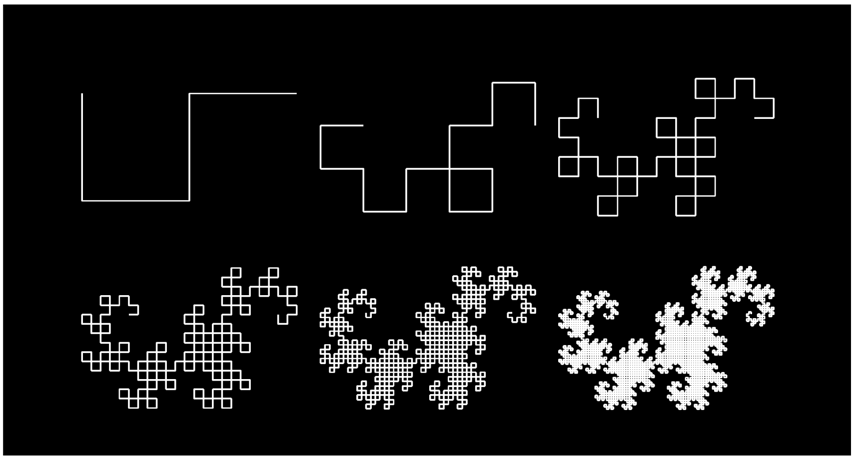

Figure 1: Dragon curves, generated by strings $\omega_2, \omega_4, \cdots, \omega_{12}$.

5 Evaluation

5.1 Features

5.2 Code Organisation

6 Conclusions

References

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