The Escape Buffer: Efficient Computation of Escape Time for Linear Fractals

Daryl H. Hepting
School of Computing Science
Simon Fraser University
Burnaby, British Columbia
V5A 1S6, CANADA
e-mail: darylh@cs.sfu.ca

John C. Hart
School of EECS
Washington State University
Pullman, Washington
USA 99164-2752
e-mail: hart@eecs.wsu.edu

Abstract

The study of linear fractals has gained a great deal from the study quadratic fractals, despite important differences. Methods for classifying points in the complement of a fractal shape were originally developed for quadratic fractals, to provide insight into their underlying dynamics. These methods were later modified for use with linear fractals. This paper reconsiders one such classification, called escape time, and presents a new algorithm for its computation that is significantly faster and conceptually simpler. Previous methods worked backwards, by mapping pixels into classified regions, whereas the new forward algorithm uses an "escape buffer" to mapping classified regions onto pixels. The efficiency of the escape buffer is justified by a careful analysis of its performance on linear fractals with various properties.

1 Introduction

Whether fractal geometry is considered a tool for computer art (e.g. [?]), a photorealistic model of natural phenomena (e.g. [?]), a fruitful example of scientific visualization (e.g. [?]), or an alluring introduction to higher mathematics (e.g. [?]), it typically enjoys a symbiotic relationship with computer graphics. This relationship is complemented by the incorporation of fractals as modeling primitives in computer graphics systems (e.g. [?, ?, ?, ?]).

The escape buffer continues this tradition by offering a new method useful for both the visual investigation of a family of fractal shapes called *linear fractals,* and the resulting generation of geometric models for image synthesis. Escape time was originally developed as a method for visualizing the dynamics of complex quadratic fractals, but has been extended to linear fractals [?, ?, ?] (and was the focus of the graduate art of Gordon Lescinsky [?]).

Section 2 reviews previous methods for determining the escape time of first quadratic, and later linear, fractals. Section 3 provides a brief summary of quadratic and linear fractals sufficient for this discussion. Section 4 defines the escape-time function for quadratic fractals and transfers it to linear fractals. Section 5 introduces the escape buffer method for determining escape time for linear fractals. Section 6 analyzes its performance. Section 7 relates the results to computer graphics and visualization, and highlights some open problems for future research.

2 Previous Work

This section briefly overviews the previous work leading to the escape buffer. Some details are omitted and filled in later, in Section 3 which defines of the complex quadratic and linear fractal models, and in Section 4 which fully describes the escape time classification.

Several algorithms for classifying divergent points of complex quadratic dynamical systems were described in [?], and listed in detail in [?]. The earliest methods plotted simple discrete level sets, counting the number of function applications required to transform a point outside a large circle. These level sets were later generalized into a continuous potential and escape time, producing continuous classifications of divergent points. These dynamics can be displayed with a continuous spectrum of color, or as a smooth height field yielding a variety of new landscapes for image synthesis.

Prusinkiewicz and Sandness [?] and Barnsley [?] described how the level set method originally developed for quadratic fractals could be extended to linear fractals modeled by iterated function systems, by inverting the IFS. Whereas complex quadratic dynamical systems consist of a single transformation, the analagous inverted IFS contains several transformations, the proper one of which must be chosen for each iteration.

Their method for choosing the proper transformation segregated space into distinct regions, such that the proper transformation would be applied to any point based on the region containing the point. These regions could be intuitively designed only for the simplest linear fractals, and the region's boundaries were themselves fractal in some cases.

Hepting et al. [?], expanding on an idea mentioned at the end of [?], avoided the use of regions by checking all possibilities from iterating all possible transformations of the inverted IFS. This avoids the problem of defining regions at the combinatorial expense of traversing a tree of possibilities. Determining level sets is a maximization of the number of iterations needed to escape a bounding circle, so they were able to prune the tree when a branch succeeded prematurely. They also developed a continuous classification of divergent points for iterated function systems analogous to the escape-time classification of quadratic fractals.

Hepting et al. [?] also proposed using a grid to store prior escape times to avoid their recomputation for later calculations. Although this scheme interpolated existing values, divergence from these values were very small and the time savings was substantial, as illustrated in Section 6.

Prusinkiewicz and Hammel [?] adapted the discrete and continuous escape-time classification to a larger class of linear fractals, specifically those represented by language-restricted iterated function systems [?]. Although the language of an LRIFS can be located anywhere in the Chomsky hierarchy, it is typically regular (although [?] demonstrates some examples resulting from a context-free LRIFS that can not be represented by a regular LRIFS). This exposition focuses on a representation, equivalent to the regular LRIFS, called the recurrent iterated function system, which is defined in Section 3.3.

This paper improves on these previous results by developing a significantly faster forward algorithm for determining escape time that neither defines regions nor follows all point iteration possibilities.

3 Background

Understanding an escape time visualization requires some knowledge of the underlying dynamical system on which it operates. Although this work focuses on linear fractals, this section begins with a brief review of the study of quadratic fractals, which produced the first incarnation of escape time. (More detailed reviews of this study can be found in [?, ?].) Summaries of iterated function systems and recurrent iterated function systems follow, which are greatly condensed from [?] and [?], respectively.

3.1 Quadratic Dynamics

The dynamics of all quadratic functions in the complex plane can be explored by investigating a single family of complex quadratic functions $f_c : \mathbf{C} \to \mathbf{C}$ defined as

$$f_c(z) = z^2 + c \tag{1}$$

parameterized by a single constant $c \in \mathbf{C}$ [?].

The dynamics of these functions become evident by examining the resulting orbit of their iteration on an initial point. Given an initial point $z_0 \in \mathbf{C}$ and a function f_c, the points in the orbit $(z_1, z_2, \ldots)$ are defined recurrently through iteration as

$$z_i = f_c(z_{i-1}).$$

For each function f_c, there is always a non-empty set of initial points z_0 whose orbits diverge. Such points are called the *basin of attraction of infinity,* whereas their complement is called the *filled-in Julia set* of f_c, denoted K_c. Although interesting dynamics occur in the filled-in Julia set, its complement (the basin of attraction of infinity) is more useful for mathematical analysis [?] and has subsequently been the primary focus of visualization algorithms.

For the sake of computation, infinity is often approximated by an *infinity circle,* a circle centered at the origin of radius sufficiently large such that the orbit of any initial point that travels outside this circle is known *a priori* to be divergent.

The *level set method* classified initial points with divergent orbits by counting the number of iterations required to escape the infinity circle. The *continuous potential method* smoothly interpolated the discrete steps of the level set method, by examining the magnitude of the first point in the orbit to escape the infinity circle [?, ?].

3.2 Iterated Function Systems

Although popularized in [?] for their natural modeling potential, iterated function systems had previously appeared in various forms [?, ?]. The following follows the form of Hutchinson [?].

This paper focuses on two-dimensional computer graphics, and the domain of its definitions is specifically $\mathbf{R}^2$ although they extend to $\mathbf{R}^n$ or any other complete topological space. The collection of all subsets of a space is called the *power set* of the space, denoted $\mathcal{P}(\cdot)$, and will be used to define the domain and range of transformations that pointwise map sets to sets.

A function $T : \mathbf{R}^2 \to \mathbf{R}^2$ is *Lipschitz* if and only if there exists some positive value λ such that

$$||T(\mathbf{x}) - T(\mathbf{y})|| \leq \lambda ||\mathbf{x} - \mathbf{y}|| \tag{2}$$

for all $\mathbf{x}, \mathbf{y} \in \mathbf{R}^2$. The minimum value λ satisfying (2) is known as the *Lipschitz constant* of T, denoted $\mathrm{Lip}T$. The function T is *contractive* if and only if $\mathrm{Lip}T < 1$.

An iterated function system $\mathcal{T} = \{T_i\}_{i=1}^N$ consists of a finite set of contractive affine maps[1] $T_i : \mathbf{R}^2 \to \mathbf{R}^2$. In computer graphics, such affine maps are typically represented by 3×3 homogeneous transformation matrices [**?**].

The *Hutchinson operator,* $\mathcal{T} : \mathcal{P}(\mathbf{R}^2) \to \mathcal{P}(\mathbf{R}^2)$ of an IFS $\mathcal{T}$

$$\mathcal{T}(A) = \bigcup_{i=1}^{N} T_i(A).$$

defines the application of the IFS on a given set $A \subset \mathbf{R}^2$.

Each IFS $\mathcal{T}$ describes a non-empty compact set $\mathcal{A} \subset \mathbf{R}^2$, called the attractor of $\mathcal{T}$, which is uniquely invariant under the Hutchinson operator

$$\mathcal{A} = \mathcal{T}(\mathcal{A}) \tag{3}$$

and is the limit set of repeated application of the Hutchinson operator

$$\mathcal{A} = \lim_{i \to \infty} \mathcal{T}^{\circ i}(B)$$

where $\mathcal{T}^{\circ i}$ is the i-fold composition of $\mathcal{T}$ and $B \subset \mathbf{R}^2$ is any non-empty bounded set.

The inverse of an IFS $\mathcal{T} = \{T_i\}_{i=1}^N$ consists of a set of dilation maps $\mathcal{T}^{-1} = \{T_i^{-1}\}_{i=1}^N$. If $\mathcal{A}$ is the attractor of $\mathcal{T}$, then one also has that

$$\mathcal{A} = \bigcap_{i=1}^{N} T_i^{-1}(\mathcal{A}) \tag{4}$$

although unlike (3), $\mathcal{A}$ is not a unique solution to (4). If an IFS $\mathcal{T}$ contains a projection, then its inverse is undefined.

An IFS $\mathcal{T}$ satisfies the *open set property* if and only if there exists an open set $O \subset \mathbf{R}^2$ such that $\mathcal{A} \subset \overline{O}$ (where $\overline{O}$ denotes the closure of O) and

$$\bigcap_{i=1}^{N} T_i(O) = \emptyset.$$

The efficiency of IFS algorithms generally depends on three factors: the number of the IFS maps N, their Lipschitz constants and whether the IFS satisfies the open set property.

3.3 Recurrent Iterated Function Systems

More complex and less strictly self-similar shapes result from restricting the application of IFS transformations by some mechanism. Such mechanisms have appeared in different forms and with various names, and with subtle differences that this section aspires to explain. Although the most recent work on escape time for linear fractals [**?**] used the language-restricted iterated function system, this work instead utilizes the recurrent iterated function system [**?**] (identical to the controlled iterated function system [**?**]).

A recurrent iterated function system consists of an IFS $\mathcal{T} = \{T_i\}_{i=1}^N$ as well as a *control digraph* G consisting of N vertices corresponding to the IFS maps, and directed edges, denoted by the ordered pair $\langle i, j \rangle$. The notation $\langle i, j \rangle \in G$ indicates that digraph G contains a directed edge from vertex i to vertex j, and implies that transformation T_j may be applied directly after transformation T_i.

A convenient mechanism for describing the dynamics of an RIFS replaces the set $A \subset \mathbf{R}^2$ with the set-vector $\mathbf{A} = (A_1, A_2, \ldots, A_N) \subset (\mathbf{R}^2)^N$. The elements of the set vector keep points distinct to maintain proper transformation composition, in that a point in the set-component A_i is the result of application of the transformation T_i.

The Hutchinson operator $\mathcal{T} : \mathcal{P}((\mathbf{R}^2)^N) \to \mathcal{P}((\mathbf{R}^2)^N)$ of an RIFS takes set-vectors to set-vectors as

$$\mathcal{T}(\mathbf{A}) = (\mathbf{T}_1(\mathbf{A}), \mathbf{T}_2(\mathbf{A}), \ldots, \mathbf{T}_N(\mathbf{A}))$$

where

$$\mathbf{T}_j(\mathbf{A}) = \bigcup_{\langle i,j \rangle \in G} T_j(A_i).$$

Application of the recurrent Hutchinson operatator applies each map T_j only to the results of a previous map T_i, if and only if the edge $\langle i, j \rangle \in G$.

As before, each RIFS $\mathcal{T}$ describes a non-empty compact set $\mathcal{A} \subset \mathbf{R}^2$, called the attractor of $\mathcal{T}$. This attractor is decomposed into possibly overlapping components of the set vector $\mathbf{A} = (\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_N)$ such that $\mathcal{A} = \cup_{i=1}^N \mathcal{A}_i$. This set-vector is uniquely invariant under the recurrent Hutchinson operator

$$\mathbf{A} = \mathcal{T}(\mathbf{A})$$

and is also the limit set of repeated application of the recurrent Hutchinson operator

$$\mathcal{A} = \lim_{i \to \infty} \mathcal{T}^{\circ i}(\mathbf{B})$$

where $\mathbf{B} \subset (\mathbf{R}^2)^N$ is any set-vector consisting of non-empty bounded components $B_i \subset \mathbf{R}^2$.

The RIFS representation is equivalent to the regular language-restricted IFS representation of [**?**]).

A directed graph is *strongly connected* if and only if there exists a directed path of edges between any two vertices. A directed graph is *weakly connected* if and only if the digraph is not strongly connected but there still exists a path of edges between any two vertices. A directed

[1]Some definitions of IFS include a corresponding set of probabilities for measure theoretic results [**?**, **?**]. The use of the IFS as a geometric representation in computer graphics requires no measure theory, and this paper avoids its use.

graph is *disconnected* if it is neither strongly connected nor weakly connected. We will call an RIFS strongly connected, weakly connected, or disconnected based on the topology of its control digraph. (The hierarchical IFS of [**?**] is equivalent to an RIFS, although the connotation of the term "hierarchical" implies a weakly-connected tree structure.)

The inverse of an RIFS $\langle \mathcal{T}, G\rangle$ is found by inverting its transformations as well as reversing the directed edges in the control graph G such that if and only if $\langle i,j\rangle \in G$ then $\langle j,i\rangle \in G^{-1}$. Graph inversion preserves topology.

An RIFS $\langle \mathcal{T}, G\rangle$ with attractor specified by the set vector $\mathbf{A}$ consisting of components $A_i \subset \mathbf{R}^2$ satisfies the *open set property* if and only if there exists a set-vector of open sets $\mathbf{O}$ consisting of components $O_i \subset \mathbf{R}^2$ such that $A_i \subset \overline{O_i}$ and

$$\bigcap_{\langle i,j\rangle \in G} T_j(O_i) = \emptyset.$$

The efficiency of RIFS algorithms generally depends on the same three factors as the IFS case: the number of the RIFS maps N, their Lipschitz constants and whether the RIFS satisfies the open set property. In addition, some algorithms require the control digraph to be strongly connected. This paper makes no such constraint on the RIFS control digraph topology.

4 Escape Time

The discrete form of the escape-time classification counts the number of iterations for a point to iterate outside the infinity circle. The continuous form smoothly interpolates the areas between the discrete escape-time boundaries based on the location of the point that escapes the infinity circle.

4.1 Discrete Escape Time

Determining the discrete escape time (the Level Set Method of [**?**]) is simple for the complex quadratic case.

Definition 4.1 (Quadratic discrete escape time) *Given the function* $f_c(z) = z^2 + c$, *a disk* D_R *centered at the origin and of radius* R *sufficient such that the filled-in Julia set* $K_c \subset D_R$, *then the discrete escape time* $DE : \mathbf{C} \to \mathbf{Z}$ *is given by*

$$\begin{aligned} DE(z) &= \min\{n : \| f_c^{\circ n}(z) \| \geq R\} && (5)\\ &= \begin{cases} 1 + DE(f_c(z)) & \textit{if } \| z \| < R, \\ 0 & \textit{otherwise.} \end{cases} && (6) \end{aligned}$$

Equation (5) defines the discrete escape time as the first iteration that takes the orbit outside the disk, whereas (6) recurrently defines an equivalent. The former is easier to understand, but the latter will be more useful in the development of escape time for linear fractals.

An important connection between linear and quadratic fractals was drawn in [**?**] by inverting Equation (1) into

$$T_1(z) = \sqrt{z-c}, \quad T_2(z) = -\sqrt{z-c}, \tag{7}$$

yielding a form analogous to an iterated function system on the complex plane.

The escape-time classification was originally defined for (1) and Equation (1) is related to (7) by inversion. By this analogy, inversion of an IFS is required to compute its escape-time classification.

Prusinkiewicz and Hammel [**?**] discuss a generalization of the results presented by Prusinkiewicz and Sandness [**?**] in defining the escape-time function for language restricted linear fractals.

The following was proven for an IFS in [**?**] and for an LRIFS in [**?**]. Given an IFS $\mathcal{T}$ (RIFS $\langle \mathcal{T}, G\rangle$) for any point $x \in \mathcal{A}$ there exists at least one sequence of inverse tranformations $T_{i_n}^{-1} \in \mathcal{T}^{-1}$ (allowed by G^{-1}) such that as $n \to \infty$

$$T_{i_n}^{-1} \circ \cdots \circ T_{i_2}^{-1} \circ T_{i_1}^{-1}(x)$$

remains bounded. In contrast, every sequence of inverse mappings (allowed by G^{-1}) applied to a point $x \notin \mathcal{A}$ will eventually send it to infinity as $n \to \infty$. This theorem suggests the following definition of a discrete escape-time function, based on the one given by Prusinkiewicz and Sandness [**?**].

Definition 4.2 (IFS discrete escape time) *Given an IFS* $\mathcal{T}$ *with attractor* $\mathcal{A}$, *let* D_R *be a disk centered about the origin*[2] *of radius* R *sufficiently large such that*

$$\mathcal{T}(D_R) \subset D_R. \tag{8}$$

Then the discrete escape time is given by the function $DE : \mathbf{R}^2 \to \mathbf{Z}$ *which is defined recurrently as*

$$DE(\mathbf{x}) = \begin{cases} 1 + \max\limits_{i=1\ldots N} DE(T_i^{-1}(\mathbf{x}), D_R) & \textit{if } \mathbf{x} \in D_R, \\ 0 & \textit{otherwise.} \end{cases}$$

The discrete escape-time function is the maximum number of inverse transformations $T_i^{-1} \in \mathcal{T}^{-1}$ necessary to iterate $\mathbf{x}$ to a point outside D_R.

The condition (8) insures the infinity circle is large enough that points will not exit it at one iteration and reenter on the next. Equation (8) implies $\mathcal{A} \in D_R$ but the opposite is not true in that for some attractors there can

[2]The infinity circle is centered about the origin as a matter of notational convenience. The definitions of this and all of the following escape-time methods can be easily extended to use an infinity circle centered about any location.

be a disk D_r of radius $r < R$ such that $\mathcal{A} \subset D_r$ but $\mathcal{T}(D_r) \not\subset D_r$. In short, the smallest disk that contains its Hutchinson-operator image is not necessarily the smallest disk that contains the IFS attractor. Finding the smallest disk that contains its images, as well as the smallest disk that contains an attractor is a difficult problem, although some partial solutions have recently become available [**?**, **?**, **?**].

When $N > 1$, implementation of Definition 4.2 parses through an N-ary tree, pruning branches whenever DE returns 0 [**?**].

Definition 4.3 (RIFS discrete escape time) *Given an RIFS $\langle \mathcal{T}, G\rangle$ with attractor $\mathcal{A}$, let D_R be a disk centered about the origin of radius R sufficiently large such that $T_i(D_R) \subset D_R \ \ (\forall i)$. Then the discrete escape time is given by*

$$DE(\mathbf{x}) = \max_{j=1\ldots N} DE_j(\mathbf{x})$$

where the functions $DE_j : \mathbf{R}^2 \to \mathbf{Z}$ are defined recurrently as

$$DE_j(\mathbf{x}) = \begin{cases} 1 + \max\limits_{\langle i,j\rangle \in G} DE_i(T_j^{-1}(\mathbf{x})) & \text{if } \mathbf{x} \in D_R, \\ 0 & \text{otherwise.} \end{cases}$$

In anticipation of Section 5 which describes the escape buffer, the following definition shows alternatively that one may map the disk D_R instead of inverse-mapping the point $\mathbf{x}$.

Definition 4.4 (Forward discrete escape time) *Given $\mathcal{T}, \mathcal{A}$ and D_R as in Definition 4.2. Then the discrete escape time is given by $DE(\mathbf{x}, I)$ where I is the unity transformation $I(\mathbf{x}) = \mathbf{x} \ \ (\forall \mathbf{x})$. The integer-valued $DE(\mathbf{x}, T)$ now operates on both a point $\mathbf{x} \in \mathbf{R}^2$ and a homogeneous 3×3 transformation matrix T, and is recurrently defined*

$$DE(\mathbf{x}, T) = \begin{cases} 1 + \max\limits_{i=1\ldots N} DE(\mathbf{x}, T \cdot T_i) & \text{if } \mathbf{x} \in T(D_R), \\ 0 & \text{otherwise.} \end{cases}$$

Definitions 4.2 and 4.4 are equivalent because determining if a point is in the image of a region $\mathbf{x} \in T(D_R)$ is equivalent to determining if the inverse image of a point is in the original region $T^{-1}(\mathbf{x}) \in D_R$. In practice, it is more efficient to inverse map points than to forward map the infinity circle because the image of the infinity circle under an affinity can be an arbitrary ellipse, which can be difficult to manage and scan convert.

As Definition 4.2 was extended to Definition 4.3, Definition 4.4 extends to the RIFS case [**?**].

4.2 Continuous Escape Time

A continuous escape-time function requires a means to interpolate smoothly between the boundaries of the discrete escape-time level sets. Such a method was presented for the IFS case in [**?**], and for the LRIFS case in [**?**].

The following was originally derived in [**?**], although is presented here in the notation defined in [**?**]. Given an (R)IFS $\mathcal{T}$ and the radius R of the infinity circle, interpolation between the boundaries of discrete escape-time level sets is given by the *residual* function $\mathrm{res}_i : D_R \setminus T_i(D_R) \to [0, 1)$ defined

$$\begin{aligned} \mathrm{res}_i(\mathbf{x}) &= \frac{\log R - \log \| \mathbf{x} \|}{\log \| T_i^{-1}(\mathbf{x}) \| - \log \| \mathbf{x} \|}, & (9) \\ &= \frac{\log(\| \mathbf{x} \|^2 / R^2)}{\log(\| \mathbf{x} \|^2 / \| T_i^{-1}(\mathbf{x}) \|^2)}, & (10) \end{aligned}$$

for each map T_i^{-1} of the inverted IFS. Equation (10) follows from (9) after a few simple logarithmic identities, and is less expensive and more robust to compute. We define the residue of an IFS as

$$\mathrm{res}(\mathbf{x}) = \max_{i=1\ldots N} \mathrm{res}_i(\mathbf{x}).$$

The residue of an RIFS is more complicated, and is denoted through the individual res_i functions in Section 5.2. The residue components and the resulting maximum are demonstrated on an IFS in Figure 1.

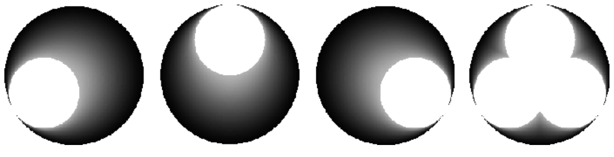

Figure 1: The left three images plot the $\mathrm{res}_i()$ function for $i = 1, 2$ and 3 respectively, for Sierpinski's gasket. The image on the right plots the resulting $\mathrm{res}()$ function.

The continuous escape time consists of an integral discrete escape time plus a fractional residue. Computation of continuous escape time consists of computing the discrete escape time, and determining the residual of the point just before it escapes the infinity circle.

Definition 4.5 (Continuous escape-time) *Given an IFS $\mathcal{T}$ with attractor $\mathcal{A}$, let D_R be a disk centered about the origin of radius R sufficiently large such that $\mathcal{T}(D_R) \subset D_R$. Then the continuous escape time is given by $CE(\mathbf{x})$ where the function $CE : \mathbf{R}^2 \to \mathbf{R}$ is defined recurrently as*

$$CE(\mathbf{x}) = \begin{cases} 1 + \max\limits_{i=1\ldots N} CE(T_i^{-1}(\mathbf{x})) & \\ \quad \text{if } \mathbf{x} \in D_R, T_i^{-1}(\mathbf{x}) \in D_R \ (\exists i) & \\ \mathrm{res}(\mathbf{x}) \quad \text{if } \mathbf{x} \in D_R, T_i^{-1}(\mathbf{x}) \notin D_R \ (\forall i) & \\ 0 \quad \text{otherwise.} & \end{cases}$$

A similar definition can be constructed for the RIFS case [?].

5 The Escape Buffer

The escape buffer is a new algorithm for computing the continuous escape time classification for the complement of a linear fractal. It uses height fields, as defined in Section 5.1 to support a forward definition of escape time in Section 5.2. Section 5.3 details the escape buffer method with pseudocode algorithms.

5.1 Height Fields

Let $B \subset \mathbf{R}^2 \times \mathbf{R}$ be a *height field* such that $(\mathbf{x}, z_1) \in B$ implies there exists no other point $(\mathbf{x}, z_2) \in B$ such that $z_1 \neq z_2$. Define the projection $\pi : \mathbf{R}^2 \times \mathbf{R} \rightarrow \mathbf{R}^2$ as

$$\pi(\mathbf{x}, z) = \mathbf{x}$$

such that $\pi(B)$ flattens the height field B. The height field evaluates like a function[3] $B : \mathbf{R}^2 \rightarrow \mathbf{R}$ as

$$B(\mathbf{x}) = \begin{cases} z & \text{if } (\mathbf{x}, z) \in B, \\ 0 & \text{otherwise.} \end{cases}$$

Furthermore, define the maximum of two height fields B_1, B_2 as

$$\begin{aligned} \max\{B_1, B_2\} \quad = \quad & \{(\mathbf{x}, z) : \mathbf{x} \in \pi(B_1) \cap \pi(B_2), \\ & z = \max(B_1(\mathbf{x}), B_2(\mathbf{x}))\}. \end{aligned}$$

The maximum operator performs the role of a z-buffer [?] on height fields.

5.2 Definition

The previous definition of height fields provides the final ingredient necessary to define a forward method for determining continuous escape time.

Definition 5.1 (Forward continuous escape-time) *Given an IFS $\mathcal{T}$ with attractor $\mathcal{A}$, let D_R be a disk centered about the origin of radius R sufficiently large such that $\mathcal{T}(D_R) \subset D_R$. Define a new* three-dimensional *IFS, $\mathcal{T}' = \{T_i'\}_{i=1}^N$, of maps $T_i' : \mathbf{R}^3 \rightarrow \mathbf{R}^3$ constructed from T_i as demonstrated by the homogeneous transformation matrices*

$$T_i = \begin{bmatrix} a_i & b_i & e_i \\ c_i & d_i & f_i \\ 0 & 0 & 1 \end{bmatrix} \longrightarrow T_i' = \begin{bmatrix} a_i & b_i & 0 & e_i \\ c_i & d_i & 0 & f_i \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

such that T_i' operates as T_i in the first two dimensions, but translates by 1 in the third. Furthermore, let a sequence of height fields $B_n \subset \mathbf{R}^2 \times \mathbf{R}$ be defined recurrently as

$$\begin{aligned} B_0 &= \{(\mathbf{x}, z) : \mathbf{x} \in D_R, z = \text{res}(\mathbf{x})\}, \\ B_n &= \max_{i=1 \ldots N} T_i(B_{n-1}) \end{aligned} \qquad (11)$$

Then the continuous escape time is given by the function $CE : \mathbf{R}^n \rightarrow \mathbf{R}$

$$CE(\mathbf{x}) = \bigcup_{n=0}^{\infty} B_n(\mathbf{x}).$$

The sequence of height fields B_n refines the IFS attractor as n increases, (in a manner similar to previous methods for approximating IFS attractors [?, ?, ?]). For example, B_0 contains the points whose continuous escape time is in $[0, 1]$, and in general B_n contains the points whose continuous escape falls between n and $n+1$ inclusive. These height fields form the *escape buffer,* and maximizes the N individual escape-time components in (11) to produce the continuous escape-time classification in the same way that a z-buffer maximizes the z component of geometry to produce correct visible-surface classification [?].

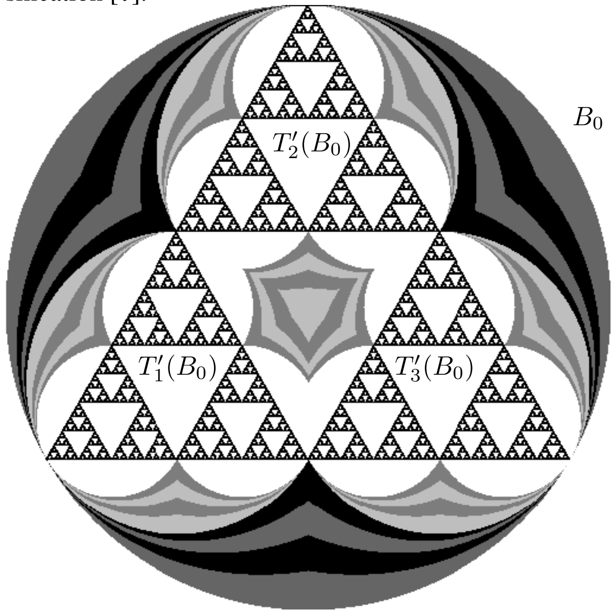

Figure 2: The first two iterations of the escape buffer for Sierpinski's gasket. The height field B_0 is computed using the residual function whereas B_1 is the maximum of $T_1'(B_0), T_2'(B_0)$ and $T_3'(B_0)$.

We can also define the forward continuous escape time on an RIFS, although distinct escapes buffer sequences $B_{i,n}$ must be maintained for each RIFS map T_i.

[3]We have used the terminology of a height field, which is more familiar in the domain of computer graphics, to replace the terminology of measures. In terms of measures, $\pi(B)$ is the support and $B(\mathbf{x})$ is the measure.

Definition 5.2 (Forward RIFS cont. escape time) *Given an RIFS $\langle \mathcal{T}, G\rangle$ with attractor $\mathcal{A}$, let D_R be a disk centered about the origin of radius R sufficiently large such that $\mathcal{T}(D_R) \subset D_R$. Let the transformations $\mathcal{T}'$ be the height-field extensions of $\mathcal{T}$ as prescribed in Definition 5.1, and let N sequences of height fields $B_n \subset \mathbf{R}^2 \times \mathbf{R}$ be defined recurrently as*

$$\begin{aligned} B_{j,0} &= \{(\mathbf{x}, z) : \mathbf{x} \in D_R, z = \mathrm{res}_j(\mathbf{x})\}, \\ B_{j,n} &= \max_{\langle i,j\rangle \in G} T_j(B_{i,n-1}). \end{aligned}$$

Then the continuous escape time is given by the function $CE : \mathbf{R}^2 \to \mathbf{R}$ as

$$CE(\mathbf{x}) = \max_{n\to\infty} \max_{j=1\ldots N} B_{j,n}.$$

The separate escape buffer sequences $B_{j,n}$ act as the individual elements of $\mathbf{B}$ used to define the attractor of an IFS. At each level, the height field $\max_{j=1\ldots N} B_{j,n}$ contains the points whose continuous escape time is *at least* between n and $n+1$ inclusive. These N height fields collectively refine the attractor as n increases.

5.3 The Escape Buffer Algorithms

The escape buffer uses the IFS to map image segments, which is analogous to the block coding IFS techniques of [**?**]. In fact, the escape buffer can operate solely in screen coordinates. Let (R)IFS maps $\mathcal{T}$ defined on $\mathbf{R}^2$ describe the attractor $\mathcal{A} \subset \mathbf{R}^2$. The window-to-viewport map $W : \mathbf{R}^2 \to \mathbf{R}^2$ maps world coordinates to screen coordinates [**?**]. The screen-space (R)IFS maps $\mathcal{S} = \{S_i\}_{i=1}^N$ defined

$$S_i = W \cdot T_i \cdot W^{-1}$$

describes an attractor $\mathcal{A}(\mathcal{S})$ equivalent to the mapped attractor $W(\mathcal{A})$ from the original (R)IFS maps $\mathcal{T}$.

```
0. initialize $E(\mathbf{x}) = 0$  $(\forall \mathbf{x})$
1. for every pixel $\mathbf{x}$
2.     $E(\mathbf{x}) = \mathrm{res}(\mathbf{x})$
3. for iterations $k = 1 \ldots n$
4.     for every pixel $\mathbf{x}$
5.         for every $S_i^{-1} \in \mathcal{S}^{-1}$
6.             $E(\mathbf{x}) = \max\{E(\mathbf{x}), E(S_i^{-1}(\mathbf{x})) + 1\}$
```

Figure 3: Escape buffer algorithm for an IFS.

Definition 5.1 suggests the algorithm shown in Figure 3. Lines 1–2 compute the residual whereas lines 3–6 use a forward method to map the residual to interpolate the boundaries of the discrete escape time level sets.

The algorithm computes continuous escape time iteratively in place. Any reader who has solved the radiosity equation will recognize this similarity between the escape buffer iteration and Gauss-Seidel iteration [**?**], which also operates in place causing some sections to converge more rapidly than others depending on the order of processing.

As with any iterative method, detecting convergence is not trivial. Let λ be the maximum Lipschitz constant of the maps $\mathcal{T}$. Then after the n iterations specified by line 3, the escape buffer will refine the infinity circle D_R to a precision of $\lambda^n R$. Hence for a desired precision p, the maximum number of iterations necessary is

$$n = \left\lceil \frac{\log p/R}{\log \lambda} \right\rceil. \tag{12}$$

Similar such equations appeared in [**?**, **?**, **?**] and were used to determine the necessary precision to approximate an attractor.

One added constraint of the escape-buffer method not shared by previous regional, tree-traveral and grid methods, is that the escape buffer must contain the infinity circle.

Organizing $\mathbf{R}^2$ into regions using the methods described in [**?**] can similarly increase the efficiency of the escape buffer by determining, in a point-by-point manner, which one of the N maps needs to be applied to determine the escape time.

```
0. Initialize $E_i(\mathbf{x}) = 0$  $(\forall \mathbf{x}, i)$
1. for $i = 1 \ldots N$
2.     for every pixel $\mathbf{x}$
3.         $E_i(\mathbf{x}) = \mathrm{res}_i(\mathbf{x})$
4. for iterations $k = 1 \ldots n$
5.     for $i = 1 \ldots N$
6.         for every pixel $\mathbf{x}$
7.             for every $S_j^{-1} \in \mathcal{S}^{-1}$ such that $\langle i, j\rangle \in G$
8.                 $E_j(\mathbf{x}) = \max\{E_j(\mathbf{x}), E_i(S_j^{-1}(\mathbf{x})) + 1\}$
9. for every pixel $\mathbf{x}$
10.   $E(\mathbf{x}) = \max_{i=1\ldots N} E_i(\mathbf{x})$
```

Figure 4: Escape buffer algorithm for an RIFS.

For the RIFS case, Definition 5.2 suggests the algorithm shown in Figure 4. This algorithm is extended from the IFS escape buffer only in that a separate escape buffer is maintained for each map, corresponding to each component of the set-vector notation used in the definition of the RIFS representation. Lines 9–10 maximize these distinct escape buffers into the final escape time classification of the RIFS attractor complement.

6 Results

6.1 Implementation

One implementation of the escape buffer works entirely in screen coordinates, and uses fixed-point arithmetic (eight bits fractional) to take advantage of the higher speed of CPU integer performance.

Although the algorithm runs fastest when the escape buffer is held in memory with the final results plotted after the algorithm's completion, plotting the pixels of the escape buffer as they change yields a fascinating and educational animation of the algorithm. The escape buffer is by far more fun to watch than any previous escape time algorithm.

The residual function is clamped such that the undefined points outside the disk D_R are set to zero whereas undefined points in the region $\mathcal{T}(D_R)$ are set to one. As the escape buffer iterates, pixels intersecting the attractor become noisy. After the final iteration, coloring pixels whose escape time exceeds some threshold yeilds an approximation of the attractor.

6.2 Tests

Tests of different implementations, weighing the factors mentioned above given the following results. A sample of each test attractor is given in Figure 5. Sierpinski's gasket, whose IFS consists of three maps which scale uniformly by 0.5 but translate in different directions, is connected and satisfies the open-set property. The disconnected gasket, whose IFS maps scale uniformly by 0.1, is disconnected but still satisfies the open-set property. The overlapping gasket[4], whose IFS maps scale uniformly by 0.9, is connected but does not satisfy the open-set property.

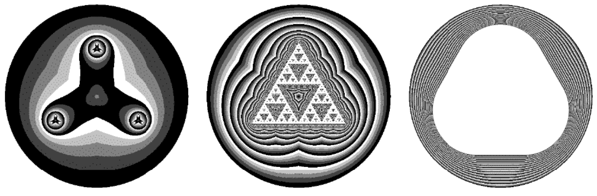

Figure 5: Test cases: Sierpinski's gasket (center), disconnected gasket (middle) and overlapping gasket (right).

The smashed gasket, shown in Figure 7, has an IFS similar to the Sierpinski gasket's augmented by a fourth transformation that scales by 0.5 but does not translate. This IFS does not satisfy the open-set condition. Its escape-time performance under the varios methods is shown in Figure 6.

[4]The overlapping gasket is a pathological example. Its image in Figure 5 is approximated by the same number of iterations as the others, but is not enough in this case to adequately converge to the attractor, as could be predicted by (12).

R	n	Tree Traversal	Grid	Escape Buffer
Sierpinski's Gasket (Lip$T_i = 0.5$)				
1	7	78.52	48.27	28.19
2	8	86.53	51.39	30.51
4	8	91.35	53.84	34.56
8	7	90.00	54.23	35.43
Disconnected Gasket (Lip$T_i = 0.1$)				
1	2	44.30	44.02	29.34
2	2	46.84	49.77	30.82
4	2	56.52	50.98	32.11
8	2	50.91	48.90	33.05
Overlapping Gasket (Lip$T_i = 0.9$)				
1	7	213.78	40.27	228.28
2	7	462.88	50.32	253.54
4	7	810.64	52.80	265.15
8	7	1166.46	55.07	270.50
Smashed Gasket (Lip$T_i = 0.5, N = 4$)				
1	7	118.81	64.50	34.82
8	7	172.51	68.27	45.65

Figure 6: Execution times for Sierpisnki's gasket tests, where R is the infinity circle radius and n is the maximum number of iterations.

6.3 Discussion

Recalling from Section 3, the *factors* affecting linear fractal efficiency are: 1. the number of transformations N, 2. their Lipschitz constants and 3. the open-set property.

Each of the escape methods is similarly affected by factor 2. Figure 6 shows that for a Lipschitz constant of 0.5, the escape buffer is clearly faster than its competitors. For a Lipschitz constant of 0.1 the escape-buffer's performance is excellent. For a Lipschitz constant of 0.9, the coherence which is exploited by the escape-buffer becomes more of a hindrance than a help. In this case, the grid method wins, but the escape buffer is still a reasonable choice over the tree-traversal method. A heuristic based on the scaling ratio would be useful to automatically select between the grid method and the escape buffer.

An (R)IFS that does not satisfy factor 3 adversely affects both the regional method of [?] and the tree-traversal method of [?], but does not impact the performance of the escape buffer.

Factor 3 provides an easy method for determining the regions for escape time (every example in [?] satisfies the open set property). Such regions are much more diffi-

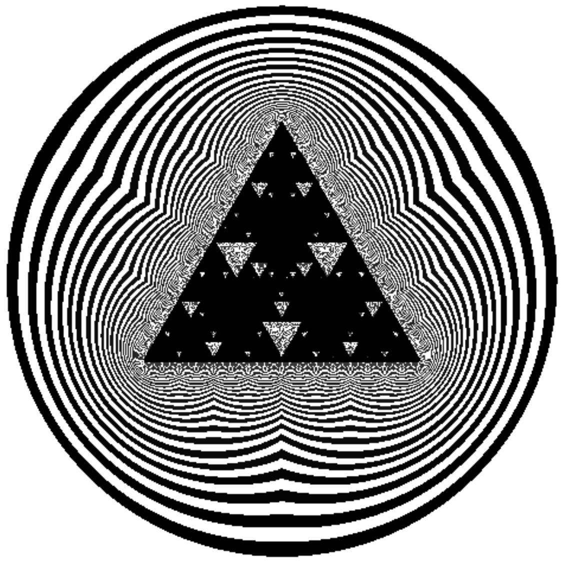

Figure 7: The smashed gasket whose IFS has an extra map.

cult to determine for an (R)IFS not satisfying the open-set property.

The tree-traversal method becomes exponential wherever the (R)IFS images of the infinity circle overlap. While satisfying the open-set condition certainly does not guarantee that the infinity circle's images will not overlap, not satisfying factor 3 does guarantee the infinity circle's images will overlap, and in general, tree-traversal becomes exponential more often for attractors whose (R)IFS fails the open set condition.

For the escape buffer, Figure 6 shows the execution times for the smashed gasket, whose IFS maps have the same Lipschitz constant as the Sierpinski's gasket's IFS maps, but otherwise differs in two ways: the smashed gasket's IFS consists of four maps, and the smashed gasket does not satisfy the open set propery. The extra map should increase the smashed gasket execution time, under all methods, by 33% over their corresponding Sierpinski's gasket times, and this is nearly the case with both the grid method and the escape buffer. However, the tree-traversal method increases not by 33%, but by 51% for $R = 1$ and 91% for $R = 8$. The tree-traversal method's lag is due to its becoming exponential in a larger portion of the image.

7 Conclusion

A critical review of previous algorithms for computing approximations to both quadratic and linear fractals has provided new insights into how more general and efficient algorithms may be designed specifically for linear fractals. The improved speed of the escape buffer makes linear fractal investigation more interactive.

7.1 Future Research

Many interesting problems regarding the escape-time function for linear fractals remain open. One such problem is extending the escape buffer to handle cases where it does not contain the infinity circle. This grid method is a first step toward a solution.

Tracking which transformations have been applied during escape-time computation hints at a new visualization method (vis-a-vis the index maps discussed in [**?**]). Through such index maps, it may be possible to gain further understanding about the dynamics of linear fractals by considering such things as flow.

If a similar forward algorithm can be constructed around distance instead of escape time, the result would greatly increase the efficiency of computing the distance transform of linear fractals, and perhaps of general shapes.

7.2 Acknowledgments

This work was carried out with the support of the Computer Graphics Research Laboratory at Simon Fraser University, and the Imaging Research Laboratory at Washington State University which is funded in part by the NSF under grant #CDA-9121675. The second author is supported by the NSF under grant #CCR-9309210. Dave Fracchia and Ken Musgrave read early versions of this paper and gave many helpful comments. This work builds on the study conducted with Przemyslaw Prusinkiewicz and Dietmar Saupe.