Ministry of Education and Science of Ukraine
V. N. Karazin Kharkiv National University
School of Mathematics and Computer Science
Department of Theoretical and Applied Informatics

Qualification work

Master

On the topic: Fractals and their application in computer modeling

Done by: 2-th year student, group MCS-64
Specialty: Computer Sciences and Information Technologies
Educational program: "Informatics"
Student full name: Zhang Siqi
Supervisor: Svetlana Ignatovich
Reviewer: Volodymyr Khalturin, PhD, Associate Professor of Department of Mathematical Modelling and Artificial Intelligence, National Aerospace University "Kharkiv Aviation Institute"

Kharkiv, 2024

Table of Contents

Introduction

Fractals are complex geometric structures that are mainly self-similar, that is, they show similar patterns in different minds. Unlike traditional Euclidean geometry, fractal geometry uses non-dimensionality to describe complex forms, making it particularly suitable for complex structures in representation. Since Benoît B. Mandelbrot proposed the concept of fractals in the 1970s, fractal geometry has become an important research direction in many fields such as mathematics, physics, art, and computer science . The unique properties of fractal structures can not only effectively represent natural phenomena such as mountains, river networks, and plant forms, but can also be used in modeling. Fractal technology provides a powerful tool to create fresh natural landscapes, plant structures and textures. While the self-similarity and nonlinear generation mechanism of fractal geometry can generate extremely novel models, traditional modeling methods usually have difficulty in accurately replicating the complexity and irregularity of computers. For example, in visual modeling, by combining fractal noise and iterative calculations in plant modeling, the branching structure and leaf arrangement of plants are simulated, such as the fractal generation method of LindenmayerSystem (Lsystem) in plant modeling. In addition, fractal technology is provided for computer graphics rendering and animation, such as simulating dynamic environments such as clouds and water flows.

This study aims to explore the application of fractals in graphics, especially the importance of generating visualizations and natural scenes. The paper proposes the basic computer concepts of fractals, classical generation methods, and the starting point of their application in different modeling scenarios, and analyzes how fractal geometry demonstrates its unique advantages and future potential in computer graphics . At the same time, the study explores the optimization strategies of fractal generation algorithms, including quantitative calculations and complexity management, to achieve efficient and accurate fractal generation.

Abstract

The application of fractal virtualization in computer graphics is studied, including the theoretical basis and practical value of fractals. First, the paper introduces the basic concepts of fractals, including definitions, core characteristics and classic examples, such as SerpinskyTriangle and Science. At the same time, the calculation method of fractal dimension and its practical application are discussed, and the self-similarity and self-organized criticality of fractals are studied in depth.

In terms of fractal generation, the paper systematically analyzes computational methods such as iterated function systems (IFS), Lindenmeier systems (L systems), and fractal noise that can efficiently simulate complex patterns in black holes. The study also discusses algorithm optimization, strategy calculation, and complexity optimization techniques to improve computational efficiency.

Keywords: fractal, fractal dimension, self-similarity, computer graphics, visual modeling

1 Basic concepts of fractals

1.1 Definition and basic properties of fractals

Fractals are a special type of geometric structure with self-similarity as its most prominent feature. Simply put, fractal structures always maintain a consistent pattern, no matter how large they are. This concept was first proposed by mathematician Benoit Mandelbrot in 1975, and has since been widely used in fields such as mathematics, physics, and computer graphics. Fractals are not just geometric figures, they also have some unique mathematical properties, especially in terms of dimensions, which are usually expressed in non-integer dimensions, which is significantly different from traditional European geometry. Self-similarity is a key feature of fractals, and its overall structure will maintain a similar shape no matter how we magnify a part of the fractal. This self-similarity may be completely accurate, statistical, or approximate, not in the general sense, and not in the general sense. In addition, another important feature of fractals is "fractal dimension", which reflects the complexity of fractals related to the shape of objects, spatial distribution and their filling efficiency, usually for integer and non-integer dimensions.

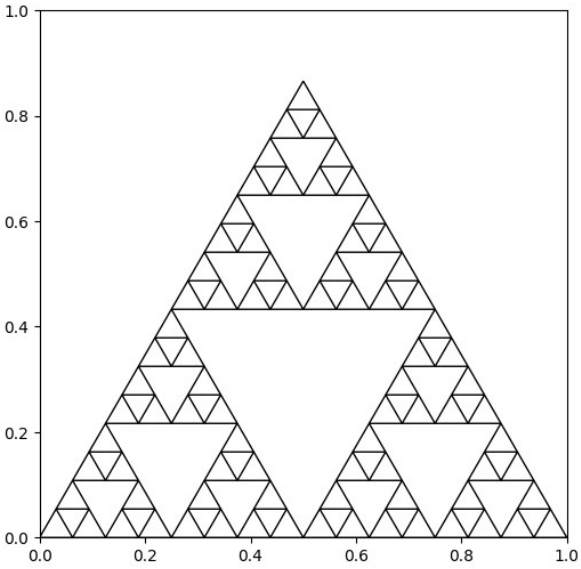
The main properties of fractals include:

1. Self-similarity: The shape of the fractal remains similar regardless of the scale.
2. Scale invariance: Certain features of fractals do not change at different scales.
3. Fractal dimension: The dimension of a fractal is usually not an integer, which reveals its spatial complexity.

1.2 Classic examples of fractals

1.2.1 Sierpinski triangle

The Serpinski Triangle is a fractal figure that is widely known for its simple construction method and remarkable self-similarity. Its construction starts with an equilateral triangle. First, a small triangle is removed from the center of the triangle, and the remaining parts form three smaller triangles: (1) and (1). Complex fractal figures can be generated by repeating this process. Then, this removal process is repeated on each small triangle. The structure is always similar to the entire figure, no matter which small triangle is viewed from. As the process is repeated, the details of the figure increase, but the overall shape remains unchanged.



1.2.1 Sierpinski Triangle

1.2.2 Koch's curve

1. The Koch curve is another classic fractal constructed in a recursive way. The construction process starts with a straight line segment. Then a small triangle is split from each line segment, and a new curve is generated by removing the middle segment of the original line segment. Next, the same steps are repeated on each newly generated line segment. The complexity of the curve gradually increases with the repetition of the operation.
2. Assume that the two endpoints of the initial line segment are:

$$A(x_1, y_1)$$
$$B(x_2, y_2)$$

The first subpointM_1

$$M_1 = \left(\frac{2x_1 + x_2}{3}, \frac{2y_1 + y_2}{3}\right)$$

The secondpointM_2

$$M_2 = \left(\frac{x_1 + 2x_2}{3}, \frac{y_1 + 2y_2}{3}\right)$$

2. By using the equilateral triangle vertex calculation formula :

Create an equilateral triangle in the center area , whose vertex coordinates $P$are expressed by the following formula :

$$P = \left(\frac{x_1 + x_2}{2} + \frac{\sqrt{3}}{6}(y_2 - y_1), \frac{y_1 + y_2}{2} + \frac{\sqrt{3}}{6}(x_1 - x_2)\right)$$

3. Formula to calculate circumference

Let L_0be the length of the initial line segment , and after $L_n$multiple recursions, be $n$the total length , which is expressed as :

$$L_n = \left(\frac{4}{3}\right)^n \times L_0$$

4. The expression calculates the fractal dimension

The fractal dimension of the Koch curve $D$is:

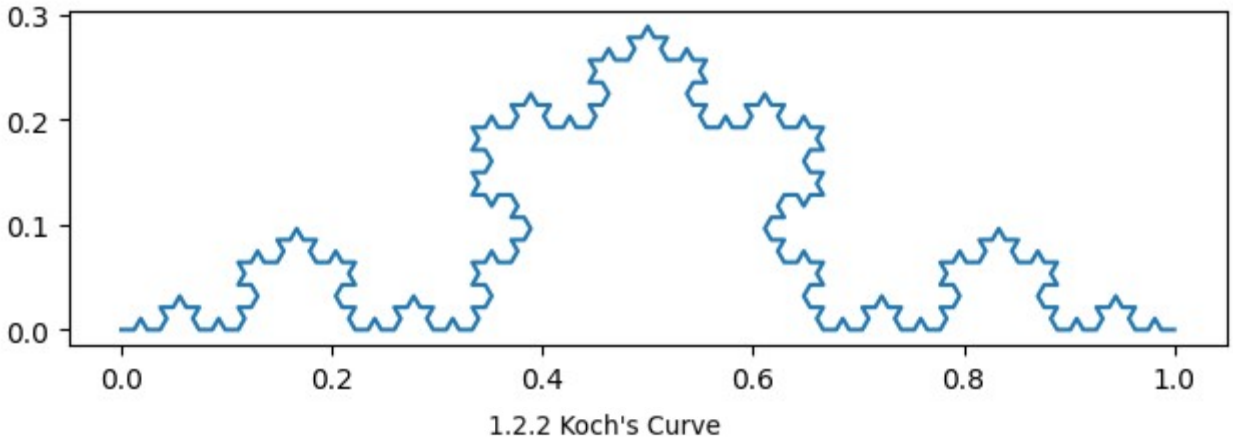
$$D = \frac{\log(4)}{\log(3)} \approx 1.2619$$

The recursive rule for the Koch curve is as follows:

First, select a line segment and split it into three parts.

Then, add an equilateral triangle above the middle part and remove the middle part of the original line segment.

Next, repeat this operation for each newly generated line segment.



1.2.2 Koch's Curve

After multiple rounds of recursion, the complexity of the curve increases significantly, showing a self-similar shape.

1.3 Fractal dimension

The fractal dimension is used to represent complex fractal structures. Several traditional shapes, such as lines, planes, and shapes, have 1, 2, and 3 dimensions, respectively. However, fractals usually exhibit a structure with a dimension called the fractal dimension, which is more complex than the integer dimension. The fractal dimension can reflect the degree of filling of the fractal figure in space, and is usually a non-integer indicating the complexity of the fractal.[1]

1.3.1 Hurst exponent

HesterIndex (H) is a special form of fractal dimension, often used to describe the self-similarity of time series data. It can reflect the persistence of data.[2]that is, whether the time series tends to maintain its current state or is prone to reversal, and is calculated through statistical analysis.

As we all know, the Hurst index has a value range between 0 and 1.
When the H value is greater than 0.5 points, it means that the data has a strong memory and will often continue the current trend.
When the H value is below 0.5, it indicates that the data tends to reverse and is prone to oscillation.
When the value of H is exactly 0.5, the fluctuations in the data are completely random, with the characteristics of white noise.
The Hurst exponent has been widely used in meteorology, finance and other fields to analyze the long-term dependence and volatility of data

1.3.2 Methods for calculating fractal dimension

The two common methods for calculating fractal dimension are grid counting method and maximum likelihood method.

1. Grid counting method
This method covers the fractal image with grids or frames of different sizes, and then counts the number of points contained in each frame. When the size of the frame decreases, the number of frames required will increase. By studying the relationship between the number of frames and the size, the dimension of the fractal can be inferred.

2. Maximum likelihood method
Maximum likelihood method is a statistical method that fits the data and maximizes the likelihood function to obtain the fractal dimension that best fits the data. This method helps to find the best mathematical model to describe the data.

3. R/S Analysis of Hurst Index
The Hurst index is often estimated by range-standard deviation analysis (R/S analysis). The specific steps are:
calculate the range (R) and standard deviation (S) of the time series.
As the time window increases, calculate the ratio of the range to the standard deviation of different windows.
Finally, by analyzing these ratios, the Hurst index can be estimated, which reflects the self-similarity and persistence of the data:
Average of a time series: Given a time series $X = \{x_1, x_2, \ldots, x_N\}$, calculate the average of the series:

$$\overleftarrow{X} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

deviation Cumulative amount sequence: Calculate the cumulative amount sequence of deviations $Y(t)$:

$$Y(t) = \sum_{i=1}^{t} \left(x_i - \overleftarrow{X}\right), \quad t = 1,2,\dots,N$$

Range $R(N)$: Calculates the deviation range of the cumulative amount series:

$$R(N) = \max_{1\le t\le N} Y(t) - \min_{1\le t\le N} Y(t)$$

Standard Deviation $S(N)$: Calculate the standard deviation of a time series:

$$S(N) = \sqrt{\frac{1}{N}\sum_{i=1}^{N} \left(x_i - \overleftarrow{X}\right)^2}$$

Range-Standard Deviation Ratio R/S:

$$\frac{R(N)}{S(N)}$$

Analysis of the expected value of R/Sdifferent lengths Nand the sample length shows a power law relationship:

$$E\left[\frac{R(N)}{S(N)}\right] \propto N^H$$

Taking the logarithm and doing a linear fit gives us the Hurst exponent H:

$$\log\left(\frac{R(N)}{S(N)}\right) = H \cdot \log(N) + C$$

Fractal dimension

Fractal dimension is a quantity used to describe the complexity of fractals. It is usually a non-integer that reflects the changes in object details at different scales. There are many ways to calculate fractal dimension, the most common of which are block dimension and Hosdorf dimension.

1.Box dimension calculation:

Divide the space into a grid: superimpose shapes on a grid of size n.

Number of squares n() to cover the pattern: Find the minimum number of squares n() to completely cover the fractal pattern.

Calculate the fractal dimension D: The fractal dimension is defined as:

$$D = \lim_{\epsilon \to 0} \frac{\log\ N(\epsilon)}{\log\ (1/\epsilon)}$$

In practical applications, different scales are used to approximate the N() fractal dimension. 2. Hausdorff dimension
The Hausdorf dimension is a more precise measure for describing more complex fractal objects. Its definition is more complicated and involves metric theory, but the basic idea is to describe its dimension through the relationship between the number of balls and the radius of the balls covering the object.
Cover space: ϵ Cover the target space with balls of radius ϵ.
Calculate the coverage metric Hs(ϵ): For a given coverage, calculate the minimum metric that satisfies the following conditions:

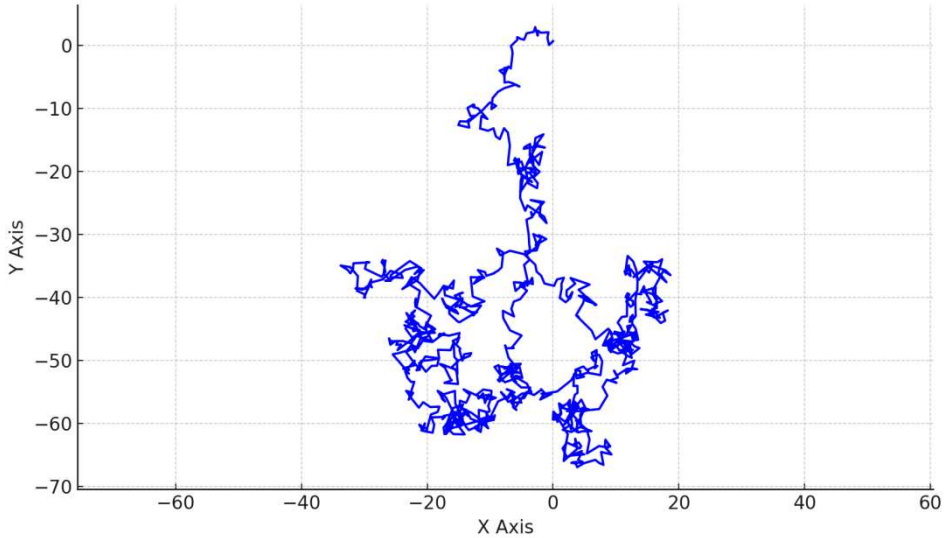
$$H^{s}(\epsilon) = \inf\left\{\sum_{i} (\text{ diameter of cover })^{s}\right\}$$

Definition of Hausdorff dimension: The critical value of Hs() from 0 to infinity: Hausdorff dimension is:
D = infs:Hs(ϵ) = 0

1.3.3 Practical Applications of Fractal Dimension

Fractal dimension is widely used, especially in analyzing natural phenomena. For example, geologists use fractal dimension to describe the distribution of rivers, the growth process of plants, the shape of coastlines, etc. Scientists can simulate the complex structure and changes of nature more accurately by calculating the fractal dimension of these natural phenomena.

In computer graphics, fractal dimension is used to produce realistic terrain and textures. The level of detail of the virtual terrain can be controlled by adjusting the fractal dimension, thus producing terrain models of varying complexity.



1.3.3 Brownian Motion

1.4 Fractal Self-similarity and Self-organizing Structure

One of its core characteristics is the self-similarity of the fractal, which refers to the shape of the fractal at different scales when the structure is similar.

The self-organizing structure of the fractal refers to the fact that the fractal system can spontaneously form a complex overall form, relying on local interactions without external interference to form a kind of self-organization.

1.4.1 Concept and classification of self-similarity

According to their characteristics, they can be divided into the following three categories: self-similarity of fractals:
1. Exact self-similarity: Exact self-similarity is the appearance of the exact same shape at different scales of the fractal. For example, the Shannon triangle is a representative of this type.

2. Statistical self-similarity: Fractals have similar statistical characteristics at different scales, such as distribution patterns and frequencies. Many objects in nature show this similarity (such as clouds and mountains).

3. Approximate self-similarity: Fractals show similar features at different scales, although these features are not exactly the same. This type of self-similarity is widely present in natural phenomena.

1.4.2 Scale invariance of fractals

Scale invariance is another important property of fractals, which means that the structure of fractals does not change significantly at different spatial scales. Regardless of the scale of observation, certain structures and properties of fractals remain the same. This allows fractals to better describe many complex phenomena in nature, such as climate change and land morphology.

1.4.3 Self-Organized Criticality (SOC)

Self-organized criticality (SOC) refers to the ability of a fractal system to naturally develop to a critical state and maintain that state through local interactions without external intervention. [3] This phenomenon is widely present

in nature, such as earthquakes, forest fires, and financial markets. SOC systems often produce sudden and unpredictable events in a critical state, and the scale of these events presents a power-law distribution with typical fractal characteristics.

1.5 Applications of fractals

Fractals are not only widely used in academic research, but also have important significance in real life. From nature to artistic creation, from scientific research to engineering applications, its influence covers many fields.[4]

1.5.1 Fractals in Nature

The natural world is full of fractal phenomena, such as the distribution of tree branches, the shape of clouds, the outline of mountains and the shape of

1.5.1 Fractal Noise (Cloud Simulation)

coastlines, which can all be described and simulated using fractal theory. These natural phenomena often have scale invariance and self-similarity, and their geometric characteristics can be analyzed using fractal models.

1.5.2 Application of fractals in art

Fractals not only play an important role in science, but also have a wide range of applications in artistic creation. In many modern artworks, artists use the concept of fractals to create complex, layered images. Fractal art is not only a combination of mathematics and art, but also reflects the laws and aesthetics of

nature.

1.5.3 Application of fractals in computer graphics

Fractals are widely used in computer graphics to create realistic terrain, textures, and natural landscapes. Using fractal algorithms, we can simulate the appearance of natural landscapes such as mountains, seas, and clouds in the real world. Fractals are also used to create dynamic and realistic animations and simulate virtual environments. By adjusting the fractal dimension and recursive depth, terrain models with enhanced graphic realism can generate terrain models with different levels of complexity and detail. In addition to enhancing the realism of visual effects, the use of fractals in computer graphics also provides an effective tool for building virtual worlds.

2 Computational method of fractal generation

2.1 Iterated Function System (IFS)

2.1.1 Definition and Principle of IFS

The mathematical tool for generating self-similar fractals is defined as an Iterated Function System (IFS). IFS transforms the initial shape into a complex fractal structure by continuously iterating and combining multiple affine transformation functions. Since IFS can create complex, self-similar patterns with infinite details, it is a key technology for constructing fractals.[5]
Principle: The core idea of IFS is to make the new points generated in each iteration gradually approach the final form of the fractal by repeatedly iterating the initial points through a set of affine transformation functions. These affine transformations are usually represented by matrices.

$$f(x,y) = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} e \\ f \end{pmatrix}$$

Parameters b and c are responsible for adjusting the scale and rotation of the image in this process, while another parameter affects the translation. The core of the iterated function system (IFS) is the multiple application of a series of different transformation rules, and C represents the number of transformations performed. As the number of iterations increases, the image gradually approaches the final fractal structure. As the number of iterations continues to increase, the image gradually approaches the final graphic structure.
Example application: Barnsley fern
The Barnsley fern is one of the famous examples in the Barnesian fractal model designed by mathematician Michael Barnsley. The model uses a specific set of affine transformations to transform the graphics of similar real ferns through an iterated function system (IFS), showing the complex and mathematical beauty of nature. The Barnsley fern not only embodies the self-similar characteristics of fractal geometry, but also, through the iterative process, illustrates that simple mathematical rules can create highly complex visual effects.

The main features and construction process of the model are as follows:
Principle Overview The core mechanism of Barnesleyes is based on a set of affine transformation rules. In each iteration, one of these rules is randomly

selected to update the position of the current point. After countless changes, the collection of these points gradually presents the unique morphology of ferns, including unique leaf branches and self-similarity (selference) in the overall structure. This is not only a reproduction of the appearance of plants in nature, but also keeps the structural features consistent at different scales.
Affine transformations Description of the IFS of the Barnsley fern consists of specific matrix parameters and corresponding selection probabilities involving four specific affine transformations. These parameters directly affect the pattern generation of different components - such as the trunk, side leaves, and smaller branches. Deformation 1 (Trunk): The trunk structure generated by relatively small ferns.

$$f_1(x,y) = \begin{pmatrix} 0 & 0 \\ 0 & 0.16 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

Probability: 0.01

Variant 2 (Main Branch): Generates most of the structure of the main branch of the fern.

$$f_2(x,y) = \begin{pmatrix} 0.85 & 0.04 \\ -0.04 & 0.85 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0 \\ 1.6 \end{pmatrix}$$

Probability: 0.85

Variant 3 (Secondary Branching): Secondary branches are generated at the edge of the leaves.

$$f_3(x,y) = \begin{pmatrix} 0.2 & -0.26 \\ 0.23 & 0.22 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0 \\ 1.6 \end{pmatrix}$$

Probability: 0.07

Morph 4 (Tip Branching): Generates tiny structures at the ends of fern leaves.

$$f_4(x,y) = \begin{pmatrix} -0.15 & 0.28 \\ 0.26 & 0.24 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0 \\ 0.44 \end{pmatrix}$$

Probability: 0.07

2.1.1 Barnsley Fern

2.1.2 Advantages of IFS in generating fractals

IFS has the following significant advantages in fractal generation:
1. Self-similarity: The characteristics of IFS ensure that the generated graphics remain consistent at all scales. This self-similarity is particularly suitable for imitating repetitive structures that exist in nature, such as trees and snowflakes, and is also particularly suitable for imitating coastlines.
2. Efficient computing performance: IFS uses simple affine transformation matrices and randomly selected transformation rules with easy-to-implement algorithms and low computational costs. Even in an environment with limited computing resources, complex graphics can be generated efficiently.
3. Save storage space: IFS does not record every generated pixel or coordinate position, but only needs to save a series of parameter matrices of transformation functions, which can greatly reduce the required storage space. This feature makes IFS a powerful tool for image compression and efficient data expression in the application of fractal image compression.
Thanks to these advantages, IFS is not only widely used in graphics generation, but also finds applications in data compression and efficient image representation, such as in fractal image compression technology.

2.2 Lindenmayer system (L system)

2.2.1 Rules and applications of the L system

Definition: The Lindenmayer System (L-System) is a recursive string generation mechanism used to simulate the growth process of plants. It consists of three main components: a character set (i.e., a series of action identifiers), a starting string (called axioms), and a series of rewriting rules (used to recursively generate new strings).
The Lindenmayer System (L-System) is a model for generating fractal graphics and natural forms based on recursive rules. The model was first proposed by Aristide Lindenmayer in 1968, originally intended to describe the growth patterns of plants. By using symbolic rules to perform simple string transformations, the L-System is able to generate complex patterns, and has therefore been widely used in many fields such as computer graphics and natural form simulation. [6]

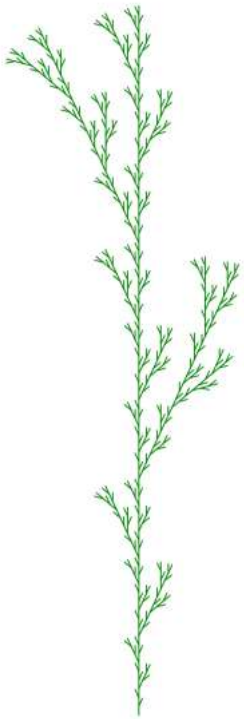
The core components of the L system mainly include the following:
Character set: A set of basic symbols used to construct plant structures or fractal graphics. Commonly used symbols include F (forward, dashed line segment), +

(turn right), - (turn left), etc.
Initial string: This is the starting string of the L-system that represents the initial state of the system. From this basic state, the L-system gradually expands into a more complex sequence of strings through recursive rules.
Transformation rules: These rules define how specific symbols in a string are replaced during recursion. For example, the rule f->f+f-f+f means that every time f is encountered, a new combination of symbols is used to replace it.
Rotation Angle: This is the angle used in the L system to determine the direction in which the line is drawn. Usually, the angle is pre-set, but the specific value of the angle may vary in different application scenarios.
Here are a few examples of basic L-system rules:
Cantor Set: The symbol set is A and B, the initial string is A, and the conversion rules are A-ABA and B-BBB. Analysis: A represents a line segment, and B represents no line segment. After several changes, the structure of the Cantor Set is formed.
Koch snowflake: The symbol set is f, +, -, the initial string is f, the conversion rule is f->f+f-f+f, and the rotation angle is 90 degrees. Analysis: The morphology of Koch and snowflakes can be generated by repeatedly applying the substitution rule of F.
Fractal fern: The symbol set is x, f, +, -, the initial string is x, the conversion rule is x->f+[x]-x]-f[-fx]+x, f->ff, and the rotation angle is 25 degrees. Analysis: This set of rules can produce fractal shapes, similar to fern leaves.
The L system is widely used in practice, especially in simulating various forms of nature. Here are some typical application cases:
Plant Modeling: The L system enables more realistic plant modeling by defining specific generation rules to replicate the branching patterns of trees, leaves, and flowers.
Fractal graphics creation: L system can generate fractal curves, such as Koch snow flakes, Serbinski gaskets, etc. With the accumulation of recursive rules, it is possible to create graphics that are extremely similar to self-similar patterns in nature, which is what we call "self-similar patterns in nature".
Urban Planning: L-system can also be used to simulate urban road layout and generate road network models with fractal characteristics. This technology helps to build complex block layouts with real urban characteristics.
Biomolecular simulation: L-systems can be used to simulate the development of molecular structures in molecular biology research. For example, L-systems can describe the replication process of DNA chains, which then produce the biological meaning of the molecular shape.
Art and Design: The fractal patterns produced by the L-system are also widely used in digital art, architectural design, natural landscapes in games and other artistic creation and design fields. Using the L-system, artists and designers can create unique elements of patterns or decorations.

The L system provides a simple and powerful mechanism for generating fractal graphics and natural forms with self-similar properties. The system is particularly suitable for reproducing fractal structures observed in nature, especially plant branching structures. By adjusting the generation rules, a variety of growth patterns can be effectively simulated, such as trees, shrubs, corals, etc. The L system plays an important role in computer graphics, especially in biological modeling, animation and design based on natural structures.

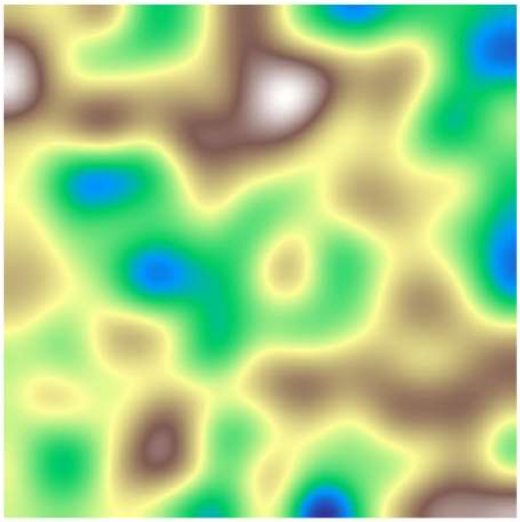
2.2.2 Example of using L-system to generate fractal tree

In the L system, a special rule is generally used to form the tree. For example, if the initial string is "F", the rule is: FFF+[+FFF]+
This rule simulates the branching process of a tree by continuously rewriting the string, which can produce a tree structure that becomes increasingly complex. The generated serial port will eventually be converted into drawing instructions, so that the tree can be drawn with fractal characteristics. This method is particularly suitable for generating complex tree and plant graphics when simulating natural scenes, and can achieve a high degree of realism. [7]



2.2.2 L-System Generated Generative Tree Structure Example

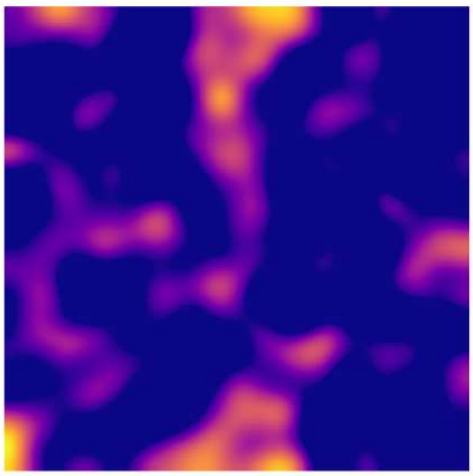
2.3 Fractal Noise and Procedurally Generated Scenes



2.3 Fractal Noise and Procedural Terrain Generation

Fractal noise is a technology used to create dynamic and realistic scenes, mainly including Perlin noise and Simplex noise (SimplexSounds). This technology is achieved by superimposing basic noise, which can show delicate changes and large-area displacement effects.
The process of generating scenes usually goes through the following stages: 1. Noise generation: using noise patterns with different gradient characteristics, using gradient noise (ie, frequency-doubling gradient noise) to create noise patterns.
2. Frequency and amplitude adjustment: The change of the frequency and amplitude of the noise will produce different effects. For example, micro changes can be simulated by high frequency and low frequency, while macro changes are more suitable to be formed by low frequency and high amplitude.
3. Potential correction: Smoothing or adding post-processing of noise values generated by slope control, etc., makes the final result look more natural.
Fractal noise is widely used in many fields, especially in game development, animation production and geographic information system (GIS), and can effectively create complex and realistic natural landscapes.

2.4 Application of fractals in computer graphics



2.4 Fractal Applications in Computer Graphics

Fractal generation technology is widely used in computer graphics, mainly in the following aspects:

1. Game Development

In game development, fractal generation is often used to create complex natural environments such as mountains, rivers, and forests. This technology can not only enhance the player's sense of immersion, but also dynamically adjust the environment according to game requirements and save storage space.

2. Movie Animation

In the production of film and television dramas, complex effects such as science fiction scenes and natural disasters all use fractal generation technology. For example, fractal generation technology can be used to achieve volcanic eruptions, snowflake generation, desert scenes, etc. It can maintain realism and improve production efficiency.

3. Virtual Reality (VR) and Augmented Reality (AR)

Fractal generation technology makes it possible to generate large-scale natural landscapes in virtual environments. In VR and AR applications, fractal generation technology not only provides rich environmental details, but also adjusts the scene in real time according to the user's perspective, enhancing interactivity and immersion.

4. Architectural Design

In the field of architectural design, fractal generation technology helps designers quickly generate complex graphics and structures, simulate natural phenomena,

and even create abstract art patterns, which brings more creative possibilities to architectural design.
5. Visual Arts
Fractal generation technology is widely used in visual arts to create unique patterns and designs. Artists can use fractal technology to create works with high complexity and beauty, enriching the expression of visual art.
6. Scientific simulation
In scientific research, fractal generation technology is used to simulate natural phenomena such as terrain, landforms, and climate patterns. This allows scientists to more effectively analyze and predict complex natural processes. Professionals in various fields can use fractal generation technology to improve work efficiency while providing users with a richer and more realistic experience and quickly generate complex graphics and environments.[8]

2.5 Optimization and improvement of fractal generation algorithm

2.5.1 Fractal generated furniture calculation

The amount of calculation and resource consumption required for fractal generation calculation becomes more complicated as the fractal generation process becomes more complex, and increases significantly when high-resolution fractal images or large-scale fractal scenes are generated. The generation process can be effectively accelerated by using task parallel processing technology. In particular, the efficiency of fractal generation can be greatly improved by using GPU for parallel computing and independently processing the generation of each pixel.

Controlling the "roughness" or "complexity" of a fractal is crucial in the design of fractal-generated furniture. This affects both the appearance and the degree of detail in the design. The dimension of a fractal is an important parameter to measure its complexity. By adjusting the dimension, the details and complexity of the fractal can be precisely controlled by the designer.
The commonly used formula is:

$$D = \frac{\log(N)}{\log(s)}$$

exist:
D is the fractal dimension;
N is the number of repeated substructures in the graph;
s is the shrinkage factor, which determines the amount of shrinkage at each level.

The details of the furniture surface and the complexity of its shape can be effectively controlled through the use of different fractal dimensions.

2. LindenMayerSystem (LSystem

SLOCHI is a recursive fractal generation method suitable for generating branching structures, such as carvings on table legs and chair backs. LSOCHI uses a serial return method to generate complex graphics:

Axiom: Define the initial state, such as "F" for drawing a line segment.

Production rules: Each time recursively, replace the symbols on the axis according to specific rules . For example:

"F"→"F[+F]F[-F]F" means that a branch structure is generated after drawing a line segment.

3. Midpoint displacement

The midpoint displacement method is often used to generate furniture surfaces with natural bumps and depressions, such as wood grain or stone simulation. This method generates a surface with random undulations by displacing the midpoints of a line segment or a grid. The basic formula is as follows:

Set the start and end height values, such as $\mathrm{H}(0)$and $\mathrm{H}(1)$.

The midpoint displacement is calculated as follows:

$$H_{\text{mid}} = \frac{H(0) + H(1)}{2} + R$$

where $R$is a random displacement, usually following a normal distribution to simulate natural roughness.

4. Affine transformation equation - Iterated function model (IFS)

IFS is a method for generating self-similar graphics through affine transformation, which is suitable for generating complex geometric figures such as engravings or repeated grid patterns. The affine transformation formula is:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} e \\ f \end{bmatrix}$$

In: $a, b, c, d$is the element of the transformation matrix, used to control scaling and rotation; $e, f$is the translation factor, which controls the position of the graphics.

Combining multiple affine transformations can generate complex repeating patterns, which are often used in grid structures or decorative details of furniture.

Mandelbrot Set and Julia Set

The formulas of the Mandelbrot set and the Julia set are often used to generate complex surface patterns, which can be used for surface decoration of furniture. The basic formula is:

$$z_{n+1} = z_n^2 + c$$

exist:

$z$is a complex number, representing the coordinates of the current point;
$c$is a constant, the Mandelbrot set $z$is iterated with different values, and the Julia set $z$is iterated with a fixed value.

These collected patterns can be applied to furniture surfaces to create a rich and varied texture and decorative effect. These formulas and methods can be combined to generate furniture designs that simulate natural or artistic styles, with complex textures and good geometry. Adequate computing technology can distribute the computing tasks of IFS and L systems on multiple processors or GPU cores, allowing the calculation process to be carried out simultaneously, thereby significantly reducing the excessive generation time. Especially in the environment of multi-core processors or GPU acceleration, it is suitable for real-time rendering and interactive applications, and can achieve computational adequacy for real-time fractal generation.

2.5.2 Complexity Analysis of Fractal Generation Algorithm

Complexity of IFS:
examples in the transformation function
The complexity of the algorithm is O(N×M) in some applications. The key to optimizing the algorithm is to choose the appropriate transformation function and number of iterations to produce high-resolution fractal images.
The complexity of the L system:
The gradient depth and generation rules mainly affect the complexity of the L system. As the gradient depth increases, the computational workload will also increase, and the generated fractal structure will become more complex. In order to maintain efficiency, the string generation process can be optimized by using a server to reduce repeated calculations.[9]

2.5.3 Modern technology for efficient fractal generation

Modern technology has been widely used in fractal generation, which improves the efficiency of generation and the texture of images. The most important ones are as follows:
1. Artificial Intelligence and Deep Learning
By training the model in deep learning, a highly similar graphic can be generated. Fractal generation can reduce the amount of calculation. Artificial neuroscience technology learns and predicts the fractal structure, thereby quickly generating images generated by complex natural images.
2. GPU acceleration can quickly form fine fractal images, which is achieved through the powerful performance of the graphics processing unit. GPU acceleration technology is particularly suitable for the real-time generation of fractal scenes in virtual reality applications, which is a particularly suitable technology for real-time rendering requirements.

Real-time rendering

Modern real-time rendering technology enables fractal graphics to be generated in real time in interactive applications, especially in virtual reality and augmented reality systems. Real-time rendering can not only reduce latency, but also dynamically adjust the details of the fractal according to the user's perspective, thereby enhancing the interactive experience.

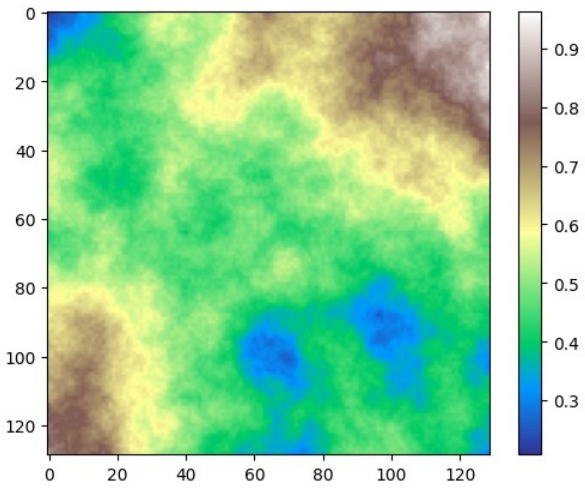
The performance and usability of fractal generation have been greatly improved in the application of modern technology, especially in algorithm-optimized systems and resource-constrained systems. Fractal generation is no longer just a graphics generation technology, but a key technology to promote various application scenarios (such as games, virtual reality, scientific simulation, etc.).

3 Application of fractals in visual modeling

The application of fractals in visual modeling enables computers to efficiently generate natural landscapes, such as mountains, canyons, and hills. Self-similarity enables visual modeling, and as the most important part, it is a property of fractal algorithms. Natural vision is often similar at different scales. Through fractal methods, we can generate textures of different sizes to simulate natural landforms.

3.1 Generation method of fractal vision

Fractal vision generation methods mainly include generation methods based on fractal noise and generation methods based on midpoint earthquakes. These methods generate virtual visual nature through different mathematical models and are suitable for both large-scale visual structures and small-scale areas with rich details.



3.1 Fractal Terrain Generation Method: Midpoint Displacement

3.1.1 Visual Generation Based on Fractal Noise

Fractal Noise:

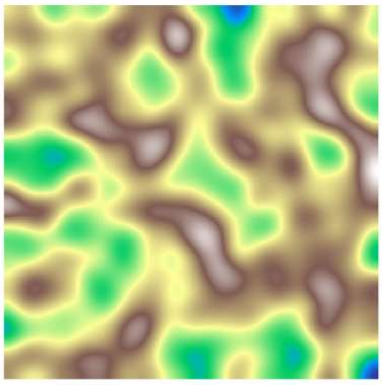
Fractal noise is a technique for generating complex noise by combining elementary noises of different frequencies and amplitudes (such as Perlin noise or simple noise). The generated noise has fractal properties: it varies consistently at different scales, which makes it well suited for simulating complexity.[10]

Multi-frequency gain:

The core module of fractal noise generation can form dynamic gain based on different noise gains. This process is called octave stacking and usually follows the following steps:

Low frequency, high pulse width: For the width structure, I will give a very simple example: the pulse width of the overall shape can be used as an important

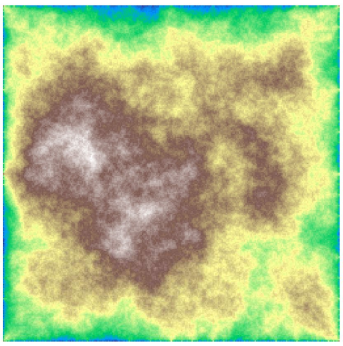
basis for generating a large pulse width structure . Weight, low weight: used to increase the weight of people who are too underweight, such as weight, weight, etc.
The frequency and amplitude of each layer of noise can be adjusted so that the final generated position has both macroscopic morphological structure and rich details.
application:
Fractal noise, especially fractal noise in games and virtual environments, has been widely used in scenes in evaluation programs. By combining different levels of noise together, similar virtual landscape structures can be generated, thus obtaining a realistic feeling and beauty in the visual scene.



3.1.1 Fractal Noise-Based Visual Terrain Generation

3.1.2 Virtual generation based on midpoint geological algorithm

Midpoint Geological Algorithm:
The midpoint geology algorithm is a fractal generation method that progressively refines the location by introducing random geology at the midpoint of each line segment or approximate location. This process can be performed in a gradient fashion until the desired level of detail is reached. [11]

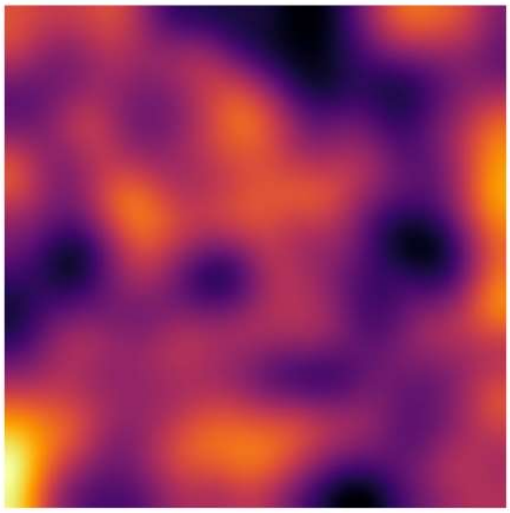


3.1.2 Terrain Generation Using Midpoint Displacement Algorithm

Algorithm principle:
The basic steps of the midpoint algorithm are as follows:
Initial Shape: Start with a simple shape (like a line or a straight line).
Midpoint: Introduce a random height offset at the midpoint of the shape.
Extensive research has shown that the offset value is inversely proportional to the number of iterations, which is a decreasing relationship .
Gradient refinement: Continuously dissect the shape and introduce fine details at the new midpoint until the fine details reach the required accuracy.
The terrain generated by the algorithm has self-similar properties and can therefore be used to simulate complex terrains such as mountains, valleys, etc.
application:
The easy and adjustable midpoint inference algorithm for detail generation makes it a useful tool for bringing scale to visual effects such as mountains and hills. The algorithm has a wide range of virtual scenes and scene generation fields, and can generate random continuity and randomness in poses and shapes.

3.2 Application of fractals in visual rendering

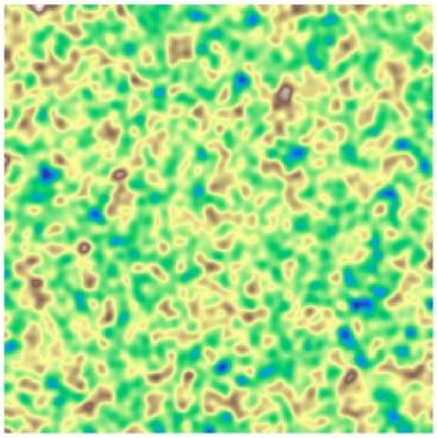
After the visual is generated, in order to further enhance the visual effect, fractal technology can also preview the visual rendering process. Through multi-layer fractal technology and detail enhancement, the visual presentation is more natural, and even has rich detailed textures when observed at close range.



3.2 Application of Fractals in Visual Rendering

3.2.1 Multi-layer segmentation technology

Multi-layer fractal:
Multi-layer fractal technology can generate fractals of different levels in the same visual model, making the visual sense continuous and rich from large scale to small scale. By combining low-frequency fractals with high-frequency fractals, the generated state has the underlying sense of hierarchy and detail expression. [12]



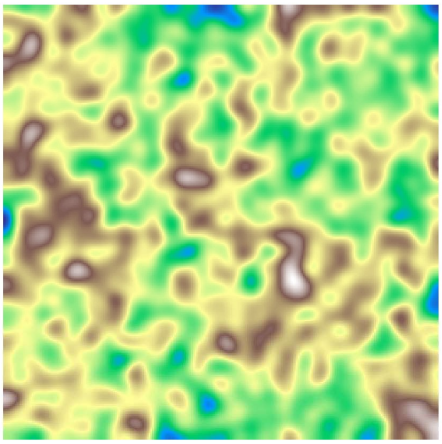
3.2.1 Multi-Layer Segmentation Technique in Fractal Rendering

Hierarchy:
First layer (base layer): Low-frequency noise creates the overall shape of the large electrode array, such as peaks and valleys.
Second layer (middle layer): Mid-frequency noise measures fluctuations in the measurement, such as hills and knolls.
The third layer (detail layer): high-frequency noise enhances small texture layers, such as power consumption textures, power consumption textures, etc.
The frequency and amplitude of each layer are modulated to ensure that the layers remain visually coherent, and to refine trends and improve structure.
Multiple layers of fractals not only increase the complexity of the movement, but also enhance the depth, making the fluctuations remain realistic at different viewing distances.
application:
Multi-layer fractal technology is used comprehensively in high-quality rendering of film and television special effects, virtual reality environments, and simulation scenes with positioning requirements. Through layering, the details of the distant and near scenes are visually presented naturally, making the scene as a whole

immersive.

3.2.2 Adding natural details

Detail Enhancement:
After generating the base terrain, you can add details to make the rendering more realistic. These details include tiny textures, texture bumps, vegetation textures, etc. The method of adding details is often accompanied by the combination of texture mapping technology and fractal noise to be truly realized .
method:
Noise Detail: Adds high frequency noise to existing potentials to simulate fine natural noise. This noise is effective on a small scale and can produce realistic weight and grit noise gain.
Texture Mapping: Use realistic texture layers (such as grass, rock, snow, etc.) to map images onto visual surfaces to further enhance visual effects.
Normal map: The light and shadow effects of the line of sight are produced by the normal map, which highlights the uneven details of the line of sight surface and makes the line of sight more three-dimensional under the illumination of light.
Main materials: Apply different materials (such as stone at high places and grass at low places) to further increase the diversity of sight lines according to the height and slope of sight lines.
Natural visual performance:
By enhancing the details, the line of sight is closer to nature, especially when combined with the light and dark effects, the line of sight presents a real sense of convexity and concavity and layering, ensuring that no matter from which angle the audience observes, they can visually appreciate the richness of the layers.



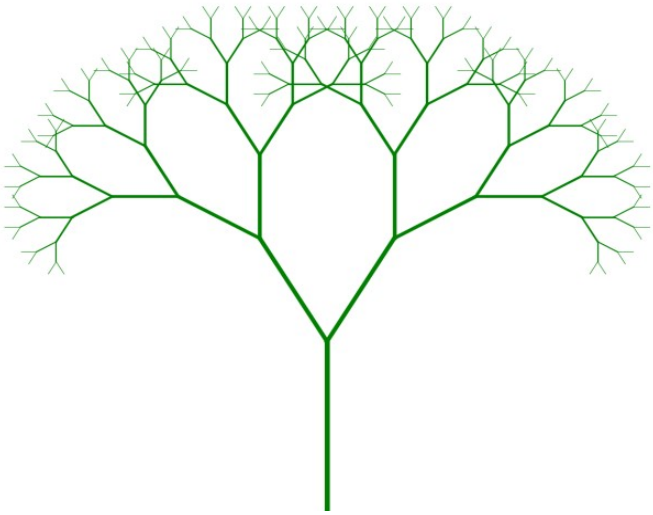
3.2.2 Enhanced Natural Details in Fractal Terrain

application:
The improvement of focus details is crucial for high-definition scenes with high fidelity requirements, such as virtual reality (VR) environments, movie special effects, and important simulation environments. The enhancement of details makes the focus not smooth when zooming in, or lacks texture when watching up close, and the visual effect and immersive feeling are improved.

Fractals generated by these methods not only play a pivotal role in visual basic modeling, but also play an important role in adding details and rendering details. Highly virtual vision can be generated through multi-layer fractal technology and enhancement methods, and empower details in virtual scenes. This technology has broad application potential in many fields: film production, game design, geographic information system (GIS).

4 Applications of fractals in other areas of computer modeling

4.1 Application of fractals in plant modeling



4.1 Fractal-Based Plant Modeling: Tree Structure

4.1.1 Application of L-system in plant structure generation

The Lindenmeyer system, or L-system, was created by biologist Arist Lindenmeyer to simulate the growth process of plants. The core of this system is to generate complex plant models through the repeated application of simple rules. [13]

The L-system consists of the following basic elements:

Letters: represent the basic units in the fractal process, such as "f" for "normal drawing", "+" or "-" for "rotation", etc.

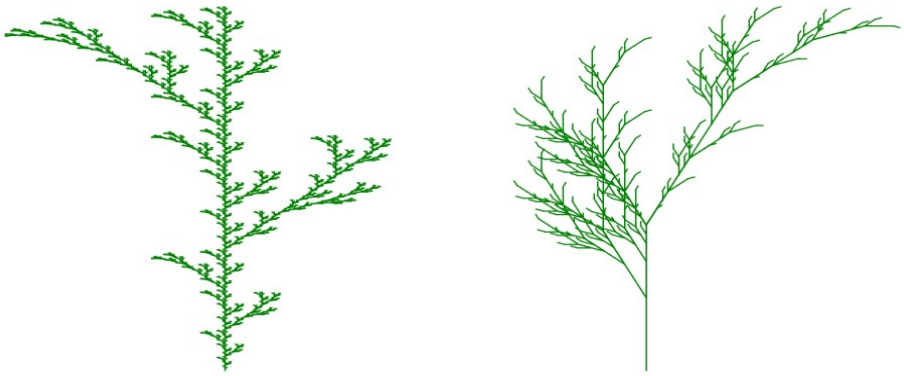
Initial state (axioms): The starting point for generating a plant pattern, for example "F".

Production rules: define substitution rules, such as “F->F+F-F+F” that controls the growth direction and branching structure.

By repeatedly iterating the starting conditions and applying transformation rules, the L-system can construct complex fractal plant models. For example, the initial state of the fractal tree is "F", and the transformation rule "F -> F[+F][-F]" is designed to simulate the branching morphology of plants. , ultimately forming a self-similar plant morphology.

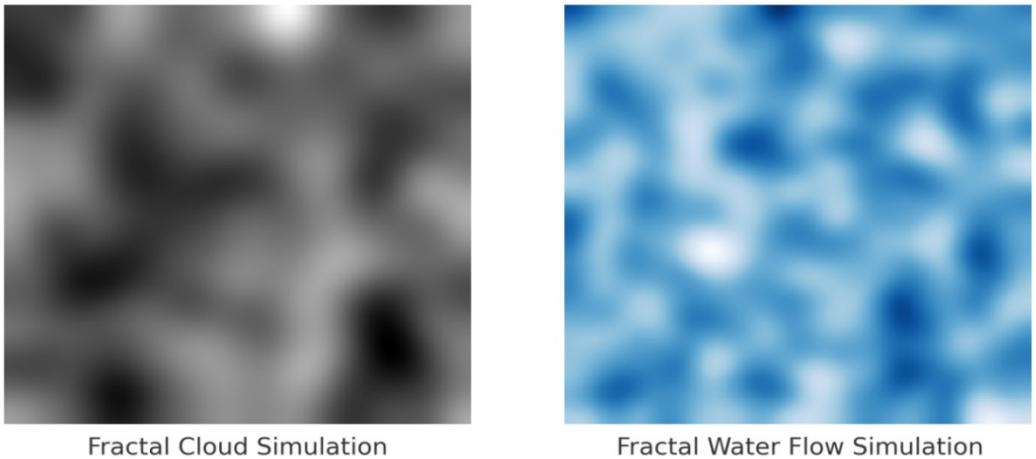
4.1.2 Examples of using L-systems to generate fractal trees and ferns

We can set the angle of the L system from the fractal tree to 25 degrees, so that the branches branch into a natural tree structure at a fixed angle at each iteration. The following is the generation of a simple fractal tree:
Set axiom (initial state) "F".
Each iteration of the generation rule will cause the trunk to split and generate new branches, i.e. “F->F[+F][-F]”. The rotation angle is set at 25 degrees, so that the branches gradually expand.[14]
We can generate fractal trees with natural shapes by recursively applying these steps multiple times.



4.1.2 L-System Generated Fractal Tree and Fern

4.2 Fractal simulation of clouds and water flow



4.2 Fractal Simulation for Clouds and Water Flow

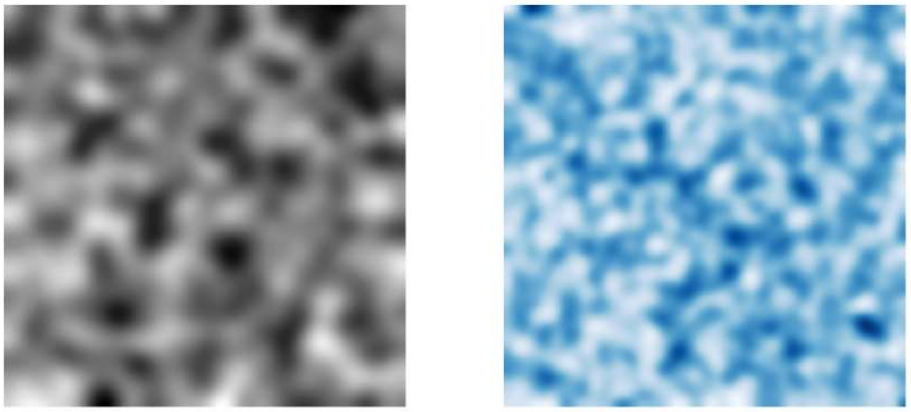
4.2.1 Fractal noise generates random details of clouds and water surfaces

In the modeling of natural environments, it is often necessary to present complex random details, such as clouds and water surfaces. Fractal noise (such as Perlinnoise) is a technology that simulates the gradual texture of nature to make the simulation of scenes such as clouds, mountains, and water more realistic, thereby generating smooth random patterns.[15]

Definition of Perlin noise:

In 1983, Kemperlin invented a noise technology that can smoothly transition, which is Perlin noise. Its basic construction is carried out according to the following steps:

Generate basic random gradient vectors: In two-dimensional space, random vectors of unit length gradient are generally generated. Calculate distance vectors: Generate distance vectors from given coordinates to the nearest grid point.



4.2.1 Fractal Noise for Cloud Layers and Water Surface Details

Interpolation calculation: The gradient vector is smoothed by interpolation method to form continuous noise.

Formula expression:

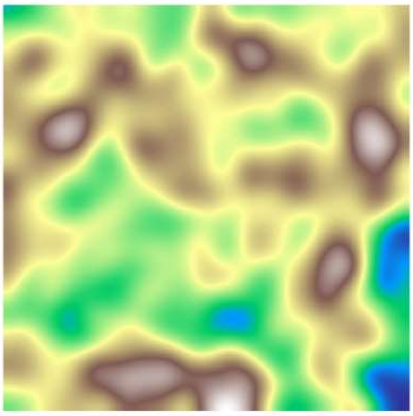
For a point in two-dimensional space (x, y), its Perlin noise value is obtained by the following formula:

$$\mathrm{PerlinNoise}(x, y) = \mathrm{fade}(x) \cdot g(x, y)$$

Among them, $\mathrm{fade}(\mathrm{x})$is the gradient function, usually a smooth cubic interpolation function is used; $\mathrm{g}(\mathrm{x}, \mathrm{y})$as a transformation of the gradient vector field, it is used to create a smooth gradient effect.[16]

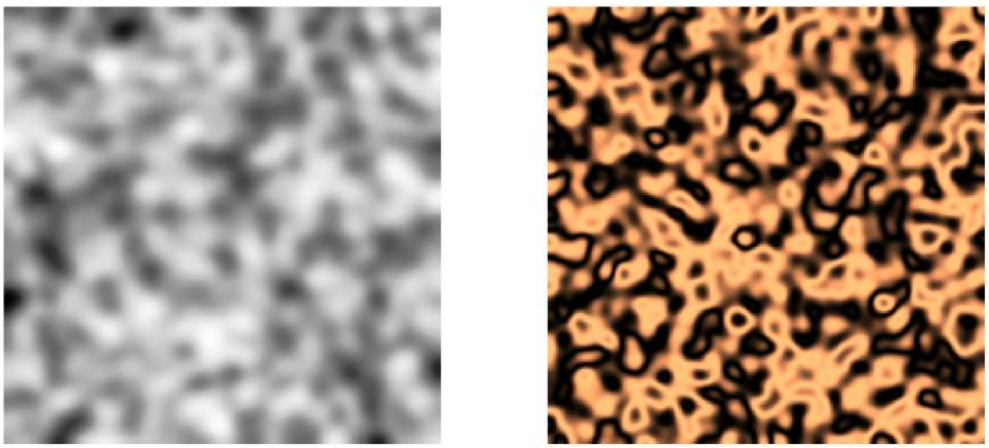
4.2.2 Application of fractals in environmental simulation

Perlin noise plays an important role in simulating natural environments such as terrain, clouds, and water. By superimposing multiple noise layers of different frequencies and amplitudes (called "fractal Perlin noise"), multi-layered, natural and realistic scenes can be generated. [17]
For example, by combining noise of different scales, the undulation of clouds or the wave effect of water flow can be achieved.



4.2.2 Fractal Application in Environmental Simulation

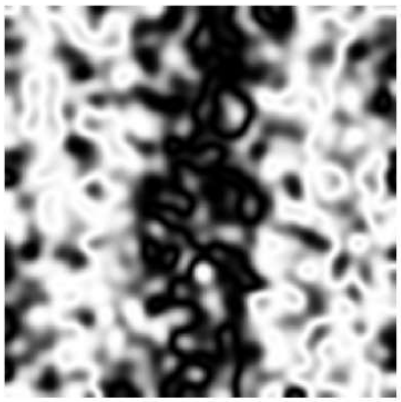
4.3 Application of fractals in texture and material modeling

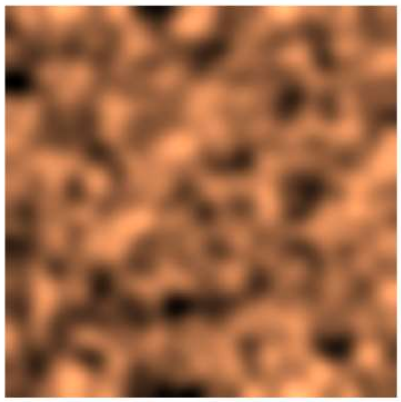


4.3 Fractal Application in Texture and Material Modeling

4.3.1 Natural material texture generation

In computer graphics, fractals can be used to create complex natural material textures such as stone surfaces, tree bark, and beach textures. Natural materials often have fine and random textures, and the self-similarity and randomness of fractal noise are well suited to simulating this effect.[18]
For example, fractal Brownian motion (FBM) can be used to generate tree bark texture, combining noise of different scales together and gradually reducing the frequency and amplitude of each layer to form a more layered and arbitrary texture.
fBm formula:
The general formula for fractal Brownian motion is:





4.3.1 Natural Material Texture Generation

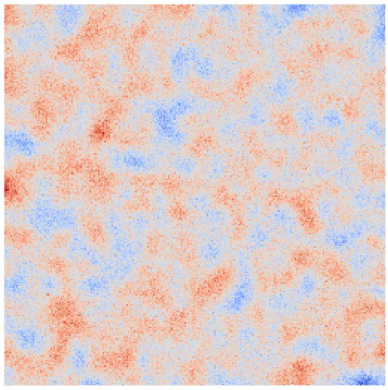
$$f(x,y) = \sum_{i=0}^{N} \left(\frac{1}{2}\right)^{i} \cdot \text{Noise}(2^{i} \cdot x, 2^{i} \cdot y)$$

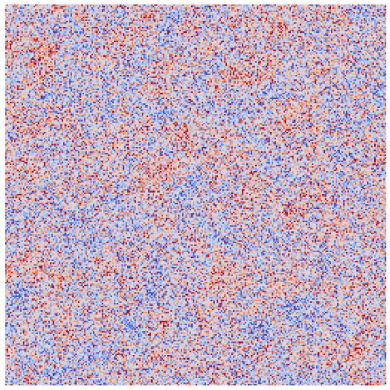
Among them, Noise(x,y)represents the noise function, iis the number of layers, $\frac{1}{2}^{i}$ is the amplitude attenuation factor, $2^{i}$and is the frequency increase factor.

4.3.2 Combination of controllability and randomness of fractal patterns

Controllability and randomness can be combined with fractal methods in texture generation. This technique is widely used in creating unique surface textures by adjusting the frequency and intensity parameters of the noise, making the virtual environment more realistic and achieving different levels of random details.
For example, from rough rock surfaces to delicate beach textures, a variety of natural textures can be produced by adjusting the frequency and amplitude of

different layers in FBS. The benefit of fractal method is its adjustability while saving storage space, which is very useful when producing large scenes.



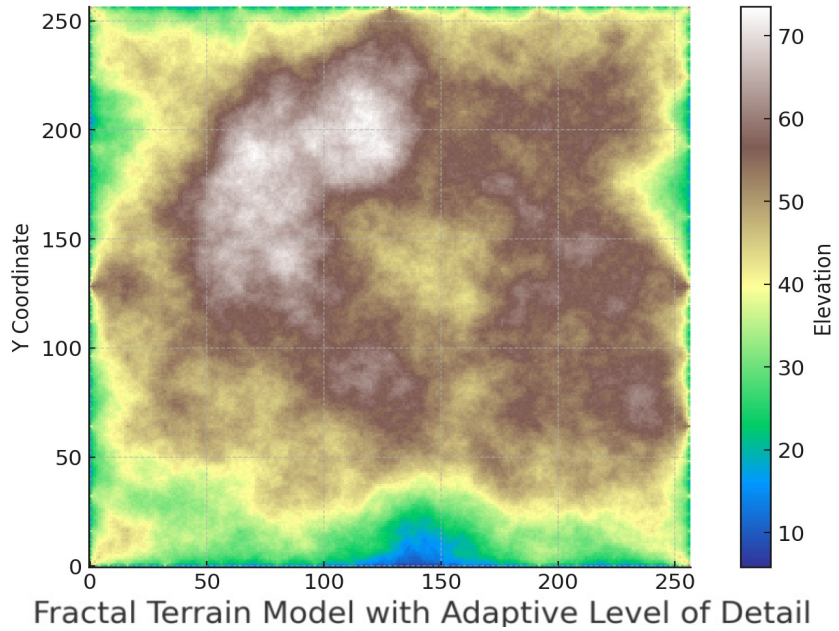


4.3.2 Combining Control and Randomness in Fractal Patterns

Through the above analysis, fractals have a wide range of application scenarios in the field of computer modeling. Their mathematical background and self-similar properties enable fractals to not only produce complex structures in nature, but also be realized through simple rules and a small amount of storage. They show very high efficiency and accuracy in natural environment simulation, plant growth modeling, material generation, etc.

5 Optimization and practical application of fractal modeling

5.1 Optimization of fractal generation algorithm



5.1.1 Algorithm efficiency optimization

For real-time rendering and large-scale computing, it is very important to optimize the efficiency of fractal generation algorithms. Fractal generation is generally based on recursive algorithms, which will bring high computational overhead when the recursion is deep. The following optimization methods can be implemented to improve the efficiency of the algorithm.[19]

Pruning technology: During the recursive generation and shaping process, the algorithm can reduce the recursive depth by pruning the substructures that do not affect the overall shape. This method is suitable for scenarios where it is not necessary to fully display local details.[20]

Dynamic resolution control: When generating the formation pattern, the long-distance part has a lower resolution, while the close-distance part maintains a higher resolution. This dynamic resolution adjustment method is suitable for real-time rendering, especially dynamic resolution adjustment in 3D rendering.

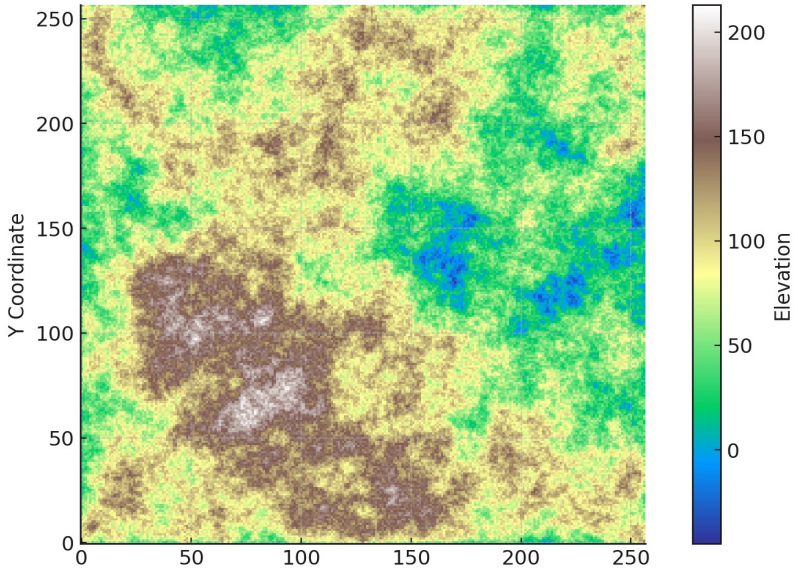
Storage and reuse: For self-similar fractal shapes , computation time can be saved by caching and reusing already computed parts. This avoids computing the same shape multiple times.[21]

5.1.2 GPU Parallel Computing Applications

GPU (Graphics Processing Unit) has strong parallel computing capabilities and can effectively accelerate fractal generation. A large number of fractal algorithms are suitable for parallel computing on GPU, which greatly improves the generation efficiency. In particular, the advantages of GPU parallel processing are particularly prominent when high-resolution output is required. Different areas in fractal generation can be assigned to multiple GPU threads for independent calculation by using GPU programming frameworks such as CUDA and OpenCL.

Take the Mandbroot set for example, the calculation of each pixel is independent, so the calculation work of the pixel can be assigned to a separate GPU thread, which greatly improves the calculation efficiency.

5.2 Practical Applications of Fractal Modeling in Computer Graphics



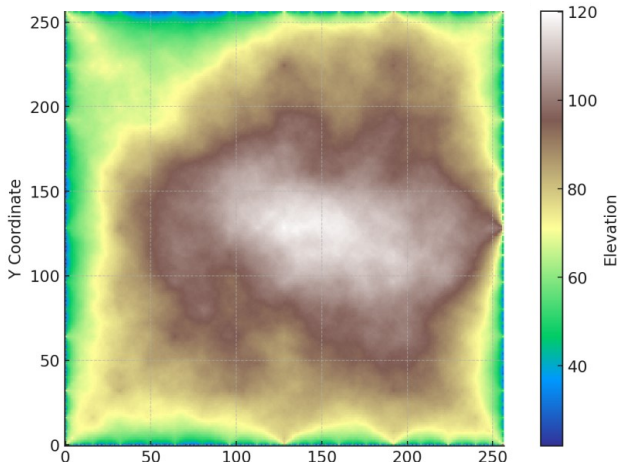
5.2 Fractal Terrain for Game Environment

5.2.1 Application in games and movies

The natural, detailed and rich environments are created using fractal modeling, a technique commonly used in games and movies.

1. Terrain generation: The use of fractals to generate natural landforms such as mountains, coastlines, and forests can create realistic natural landscapes.[22]

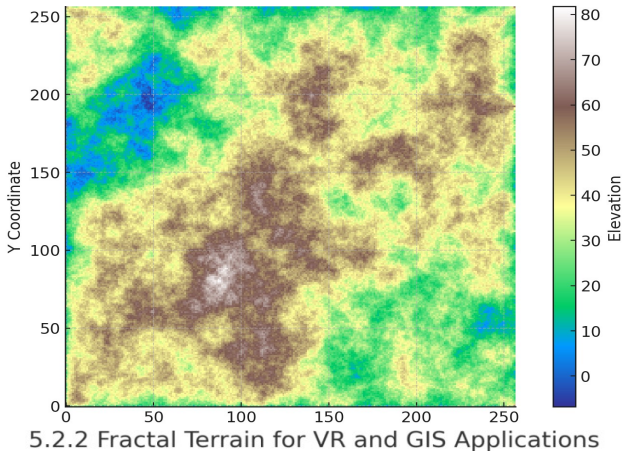
2. Cloud and water simulation: Use fractal noise (such as Perlin noise) to make the scene look more realistic, thus creating clouds and waves. 3. Special effects and abstract vision: The details generated by fractal structures can be used to enhance the impact of the picture, some complex visual effects such as magic, particle explosions and other scenes.
In movies and television, many sci-fi scenes and interstellar landscapes in Avatar and Doctor Strange use fractal algorithms. These fractal techniques are helpful in creating grand, complex scenes.[23]



5.2.1 Fractal Terrain for Cinematic Landscapes

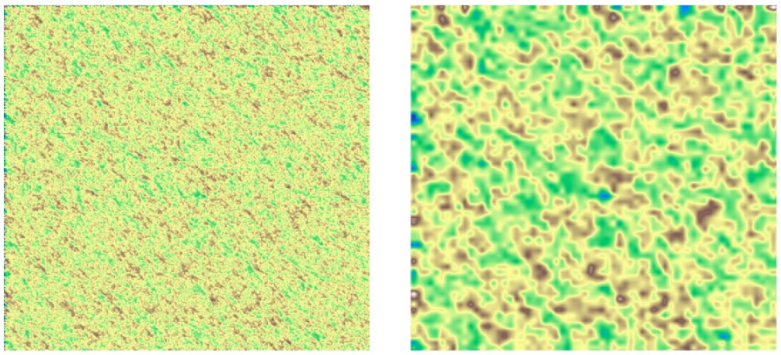
5.2.2 Application of fractals in VR and GIS

VR (Virtual Reality): In VR, the scenes that users want are real and meticulous. Natural and delicate virtual scenes can be created using fractal models. This reduces storage requirements and increases image rendering speed by generating fractals of terrain such as trees, mountains, and rocks.[24]
GIS (Geographic Information System): The application of fractal technology in terrain modeling and landform research can create realistic surface features. Fractal models provide multi-scale self-similarity that can simulate complex terrain details. Geographic Information Systems can use fractal-generated terrain models to analyze landforms and apply them.[25]



5.2.2 Fractal Terrain for VR and GIS Applications

5.3 Fractal Image Compression



5.3 Fractal Image Compression: Original vs Compressed

Original Fractal Image Compressed Fractal Image (Reduced Detail)

5.3.1 Basic principles of fractal image compression

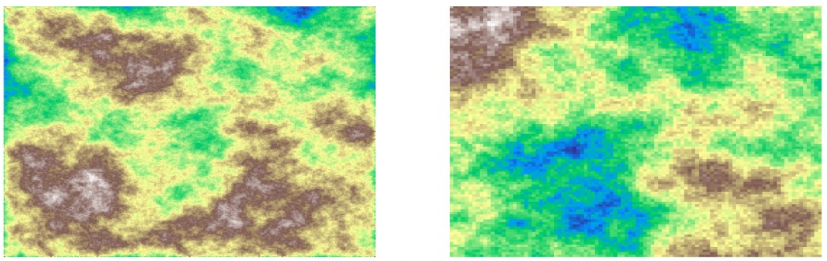
When an image is processed by fractal image compression technology, it is divided into multiple small areas based on the principle of self-similarity and matched to a larger range that is similar to it.
Since these areas are self-similar, only their descriptions can be saved without storing the information of each pixel, which significantly saves storage space.[26]

Basic principle:
1. Block matching: Cut the image into multiple non-overlapping small blocks (called "range blocks") and look for similar blocks in a larger area of the image (called "domain blocks").

2. Transformation rules: For each range block, find the domain block that is most similar to it and record the corresponding transformation rules, which include rotation, scaling, and translation.

3. Store rules instead of pixels: Convert the entire image into a set of transformation rules instead of storing the specific information of each pixel.

During the decompression process, the image is repeatedly constructed using these transformation rules. By continuously applying these self-similar transformations, the original image can be restored.



5.3.1 Fractal Image Compression: Basic Principle of Self-Similarity

Full Fractal Image with Self-Similarity Zoomed-In Section (Self-Similarity)

5.3.2 Barnsley’s contribution

Mathematician Michael Barnsley first introduced the theory and methods of fractal image compression. He developed a fractal compression algorithm by utilizing fractal self-similarity and published the classic book "Fractal Image Compression: Theory and Practice". Barnsley's research laid the foundation for fractal image compression and revealed the potential of fractal self-similar structures in image compression.[27]

The iterated function system (IFS) developed by Barnsley constructed a mathematical model for fractal image compression technology. It proved that fractals can efficiently represent complex images, especially natural images. In the compression process, IFS can be used to express the repetitiveness and self-similarity in the image, thereby significantly reducing the storage requirements of image data. Barnsley's contribution not only promoted the development of image compression, but also provided important mathematical tools and ideas for the application of fractal theory and fractal modeling in the field of image processing.[28]

Conclusion

This article comprehensively analyzes the basic principles of fractals, generation techniques, and their various applications in the field of computer modeling. The theory and technology of fractals provide efficient methods and tools for computer graphics and simulation of natural phenomena. By studying the definition, basic properties, self-similarity, and self-organization characteristics of fractals, it is found that fractals are not just abstract concepts in mathematics. Its theory and characteristics have wide applications in many fields.

Many algorithms, such as iterated function system (IFS), Lindenmayer system (LSystem), and fractal noise, can effectively generate complex natural patterns. The self-similarity and immutability of these methods ensure the stability of the structure and the authenticity of the structure. With the continuous advancement of parallel computing and GPU acceleration technology, real-time rendering has been supported by fractal computing. Fractal technology is widely used in fields such as virtual reality (VR) and geographic information system (GIS) to efficiently create simulated natural landscapes. In terrain modeling and rendering, fractal methods are very suitable for simulating complex and irregular natural landforms (such as mountains, forests, clouds, etc.). In games, movies, and virtual exhibitions, fractal generation technology enhances the realism of the virtual world. In addition, fractal texture generation technology can realistically simulate natural materials such as stone, wood, clouds, and water, providing innovative design methods for the fields of digital art and architectural visualization.

In addition to the above applications, this study also conducted in-depth research on the application of fractals in plants, clouds, and material modeling. These applications make the modeling more natural and can better display the controllable randomness in the details. The efficiency and quality of fractal generation have been significantly improved with the improvement of algorithm efficiency and the advancement of GPU computing power, and have been widely used in many fields.

The application range of fractal technology is very wide, from simulating natural phenomena to providing the necessary technical support for computer graphics. Fractal technology also shows important value in the fields of image compression and texture generation. In particular, researchers such as Barnsley have achieved efficient data storage under limited storage space through fractal image compression technology. In general, fractal modeling has become an indispensable tool in computer science, natural sciences, and art design. With the

continuous advancement of fractal generation algorithms and hardware technology, fractals will play an increasingly important role in simulating complex natural phenomena, generating virtual world scenes, and enhancing the realism of digital experience.

Reference

[1] Fox, C.G., 1989. Empirically derived relationships between fractal dimension and power law from frequency spectra. Pure Appl. Geophys., 131:211-239.

[2] Gneiting, Tilmann, and Martin Schlather. 2001. Stochastic models which separate fractal dimension and Hurst effect. arXiv arXiv:physics/0109031.

[3] D. Dhar, Studying self-organized criticality with exactly solved models (1999b), arXiv:cond-mat/9909009.

[4] Lai,D., 2004. Estimating the Hurst effect and its applications in monitoring clinical trials. Comput. Statist. Data Anal. 45, 549–562

[5] K.-S. Lau and S.-M. Ngai, A generalized finite type condition for iterated function systems. Adv. Math. 208 (2007), 647–671.

[6] Farah Ben-Naoum. A survey on L-system inference. INFOCOMP Journal of Computer Science, 8(3):29-39, 2009.

[7] Rosindell, J., Harmon, L.: OneZoom: A fractal explorer for the tree of life. PLOS Biology 10(10) (2012)

[8] Beier, Thaddeus and Shawn Neely. "Feature-Based Image Metamorphosis," Computer Graphics, Proceedings of SIGGRAPH 92, 26 (2), July 1992 pp. 35-42.

[9] Milne, B.T., 1991. Lessons from applying fractal models to landscape patterns. In: M.G. Turner and R.H. Gardner (Editors), Quantitative Methods in Landscape Ecology: the Analysis and Interpretation of Landscape Hetergeneity, Springer, New York, pp. 199- 235.

[10] B.B. Mandelbrot and J. W. Van Ness, 1968."Fractional Brownian motions, fractional noises and applications," SIA M Rev., vol. 10, no. 4, pp. 422-437.

[11] Sastry, S.P. A 2D advancing-front Delaunay mesh refinement algorithm. Comput. Geom. 2021, 97, 101772. [CrossRef] [PubMed]

[12] [Shi, 00] Shi, J. and Malik, J.: Normalized cuts and image segmentation,

IEEE Trans. Pattern Anal. Mach. Intell., 22(8): 888-905, Aug. 2000.

[13] Prusinkiewicz, P., Mundermann, L., Karwowski, R. et Lane, B., 2001. The use of positional information in the modeling of plants. In: SIGGRAPH'01, Los Angeles, California, ACM. Computer Graphics; pp. 36-47.

[14] Smith, A.R., 1984. Plants, fractals, and formal languages. Computer Graphics, 18(3).

[15] Lovejoy. S. 1983. La geometrie fractale des regions de pluie et les simulations aleatoires. Houille Blanche, 516.43 1436.

[16] Schertzer, D. and Lovejoy, S. 1985. Fractales anisotropes et dimensions elliptiques (available from the authors).

[17] Schertzer,D.,and S.Lovejoy,1988: Multifractal simulations and analysis of clouds by multiplicative processes.Atmos. Res., 21,337-361.

[18] Keller, J., Crownover, R., & Chen, S. (1989). Texture description and segmentation through fractal geometry. Computer Vision Graphics and Image Processing, 45, 150–160.

[19] A.Hill and C.J.Taylor. Model-based image interpretation using genetic algorithms. Image and Vision Computing,10(5):295-301,June 1992.

[20] Kaelo, P., Ali, M.M., 2006. A numerical study of some modified differential evolution algorithms. European Journal of Operational Research 169, 1176–1184.

[21] Deb, K., Tiwari, S., 2008. Omni-optimizer: A generic evolutionary algorithm for single and multi-objective optimization. European Journal of Operational Research 185 (3), 1062–1087.

[22] A. Bernhardt, A. Maximo, L. Velho, H. Hnaidi, and M.-P. Cani. Real-time terrain modeling using cpu-GPU coupled computation. In Proceedings of the 2011 24th SIBGRAPI Conference on Graphics, Patterns and Images, SIBGRAPI ’11, pages 64–71, Washington, DC, USA, 2011. IEEE Computer Society.

[23] Gombrich, E, H. 1999. The Uses of Images: Studies in the Social Function of Art and Visual Communication. London, Phaidon. p. 49.

[24] Spehar, B, Clifford, C, Newell, ,. and Taylor, R,P. 2003. Universal Aesthetic of Fractals. Computers & Graphics Vol 27, no. 5. p.819.

[25] S. Wu, An user intention mining model based on fractal time series pattern, Fractals-Complex Geometry Patterns Scal. Nat. Soc. 28 (2020) Article Number 2040017.

[26] Sun, Y., Xu, R., Chen, L., Hu, X.: Image compression and encryption scheme using fractal dictionary and Julia set. IET Image Process. 9(3), 173–183 (2015)

[27] Jaseela, C.C., James, A.: A new approach to fractal image compression using DBSCAN. Int. J. Electr. Energy 2(1), 18–22 (2014)

[28] Dyke, Mel. Grimethorpe Revival: Famous Faces Support a Coalfield Community. (Barnsley: Wharncliffe Books, 2013)

Acknowledgements

I would like to thank my advisor, Svetlana Ignatovich, who gave me the opportunity to write my senior thesis on fractals and their applications in computer modeling at the Department of Computer Science, for her excellent guidance and constant feedback. I would also like to thank my family and friends who always supported me and kept me going.

Appendix

Programming

1. Sierpinski Triangle

```
import matplotlib.pyplot as plt
import numpy as np

# Define the recursive function to draw the Sierpinski Triangle
def sierpinski(vertices, depth):
    if depth == 0:
        triangle = plt.Polygon(vertices, edgecolor='black', fill=None)
        plt.gca().add_patch(triangle)
    else:
        mid01 = (vertices[0] + vertices[1]) / 2
        mid12 = (vertices[1] + vertices[2]) / 2
        mid20 = (vertices[2] + vertices[0]) / 2
        sierpinski([vertices[0], mid01, mid20], depth-1)
        sierpinski([mid01, vertices[1], mid12], depth-1)
        sierpinski([mid20, mid12, vertices[2]], depth-1)

# Initial vertices of the triangle
vertices = np.array([[0, 0], [1, 0], [0.5, np.sqrt(3)/2]])
plt.figure(figsize=(6,6))
sierpinski(vertices, 4)  # Recursive depth is 4
plt.gca().set_aspect('equal', adjustable='box')
plt.show()
```

2. Koch's curve

```
import numpy as np
import matplotlib.pyplot as plt

def koch_curve(p1, p2, depth):
    if depth == 0:
        return [p1, p2]
    else:
        # Calculate the division points
        p1p2 = p2 - p1
        p3 = p1 + p1p2 / 3
        p4 = p1 + 2 * p1p2 / 3
        p5 = (p1 + p2) / 2 + np.array([-np.sqrt(3)*p1p2[1], np.sqrt(3)*p1p2[0]]) / 6
        return koch_curve(p1, p3, depth-1) + koch_curve(p3, p5, depth-1) + \
               koch_curve(p5, p4, depth-1) + koch_curve(p4, p2, depth-1)

# Define the initial points for the Koch curve
p1 = np.array([0, 0])
p2 = np.array([1, 0])

# Generate the points for the Koch curve with a recursion depth of 4
points = koch_curve(p1, p2, 4)
points = np.array(points)

# Plot the Koch curve
plt.plot(points[:, 0], points[:, 1])
plt.gca().set_aspect('equal', adjustable='box')

# Remove the default title and add it as a caption directly below the figure
plt.tight_layout()
plt.subplots_adjust(bottom=0.0)  # Completely remove extra spacing below the plot
plt.figtext(0.5, 0.0, "1.2.2 Koch's Curve", ha='center', fontsize=10)  # Align caption tightly
plt.show()
```

3. Barnsley Fern

```
import random
import matplotlib.pyplot as plt

def barnsley_fern(iterations):
    x, y = 0, 0
    points = []
    for _ in range(iterations):
        r = random.random()
        if r < 0.02:
            # Transformation 1
            x, y = 0, 0.16 * y
        elif r < 0.86:
            # Transformation 2
            x, y = 0.85 * x + 0.04 * y, -0.04 * x + 0.85 * y + 1.6
        elif r < 0.93:
            # Transformation 3
            x, y = 0.2 * x - 0.26 * y, 0.23 * x + 0.22 * y + 1.6
        else:
            # Transformation 4
            x, y = -0.15 * x + 0.28 * y, 0.26 * x + 0.24 * y + 0.44
        points.append([x, y])
    return points

# Draw the Barnsley Fern
points = barnsley_fern(100000)
x_vals, y_vals = zip(*points)
plt.scatter(x_vals, y_vals, s=0.1, color="green")
plt.axis("off")
plt.show()
```

4. Fractal Terrain Generation

```
import numpy as np
import matplotlib.pyplot as plt

def midpoint_displacement(size, roughness):
    # Initialize the grid
    grid = np.zeros((size, size))

    # Set initial corner points
    grid[0, 0] = np.random.rand()
    grid[0, -1] = np.random.rand()
    grid[-1, 0] = np.random.rand()
    grid[-1, -1] = np.random.rand()

    step_size = size - 1
    displacement = roughness

    while step_size > 1:
        half_step = step_size // 2

        # Square step
        for x in range(0, size - 1, step_size):
            for y in range(0, size - 1, step_size):
                avg = (grid[x, y] + grid[x + step_size, y] +
                       grid[x, y + step_size] + grid[x + step_size, y + step_size]) / 4
                grid[x + half_step, y + half_step] = avg + (np.random.rand() - 0.5) * displacement
        # Diamond step
        for x in range(0, size, half_step):
            for y in range((x + half_step) % step_size, size, step_size):
                neighbors = []
                if x >= half_step:
                    neighbors.append(grid[x - half_step, y])
                if x + half_step < size:
                    neighbors.append(grid[x + half_step, y])
                if y >= half_step:
                    neighbors.append(grid[x, y - half_step])
                if y + half_step < size:
                    neighbors.append(grid[x, y + half_step])
                avg = sum(neighbors) / len(neighbors)
                grid[x, y] = avg + (np.random.rand() - 0.5) * displacement

        # Reduce the displacement
        step_size //= 2
        displacement *= roughness

    return grid

# Generate and plot fractal terrain
size = 129  # 2^n + 1 for midpoint displacement
terrain = midpoint_displacement(size, 0.6)

plt.imshow(terrain, cmap="terrain")
plt.colorbar()
plt.title("Fractal Terrain Generation")
plt.show()
```

5. Fractal Image Compression

```
import numpy as np
import matplotlib.pyplot as plt
from skimage.transform import rescale

# Generate a fractal-like image (e.g., random noise smoothed iteratively)
def generate_fractal_image(size=256):
    """
    Generate a fractal-like image using random noise.
    :param size: The size of the square image (size x size)
    :return: A fractal-like 2D array
    """
    # Create a base random noise grid
    fractal_image = np.random.rand(size, size)

    # Smooth the grid using cumulative averaging to simulate fractal detail
    for i in range(1, size):
        for j in range(1, size):
            fractal_image[i, j] = (
                fractal_image[i, j] +
                fractal_image[i - 1, j] +
                fractal_image[i, j - 1] +
                fractal_image[i - 1, j - 1]
            ) / 4
    return fractal_image

# Simulate fractal compression
def fractal_compression(image, scale_factor=2):
    """
    Simulate fractal compression by downscaling and upscaling the image.
    :param image: Input image
    :param scale_factor: Downscale factor
    :return: Compressed and reconstructed image
    """
   # Downscale the image
    small_image = rescale(image, 1 / scale_factor, anti_aliasing=True)
    # Upscale the image back to the original size
    reconstructed_image = rescale(small_image, scale_factor, anti_aliasing=True)
    # Match the original image dimensions
    reconstructed_image = reconstructed_image[:image.shape[0], :image.shape[1]]
    return reconstructed_image

# Generate the fractal-like image
original_image = generate_fractal_image(size=256)

# Compress the fractal image
compressed_image = fractal_compression(original_image, scale_factor=4)

# Plot the original and compressed images
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))
ax1.imshow(original_image, cmap="terrain")
ax1.set_title("Original Fractal Image")
ax1.axis("off")

ax2.imshow(compressed_image, cmap="terrain")
ax2.set_title("Compressed Fractal Image (Reduced Detail)")
ax2.axis("off")

plt.suptitle("Fractal Image Compression: Original vs Compressed", fontsize=16)
plt.tight_layout()
plt.show()
```