Las gramáticas en la generación procedural de modelos tridimensionales

Alejandro Oceja Trueba

Sumario:

Crear modelos y entornos tridimensionales es una tarea costosa, que requiere una importante inversión de tiempo y esfuerzo para realizar el diseño y el modelado. Por estos motivos, se están investigando diferentes aproximaciones a la generación procedural de modelos tridimensionales, buscando el método más eficiente, tanto en cuanto a coste como a tiempo. En este trabajo se va a hablar de algunos de estos métodos, así como a ver la implementación de uno de ellos.

Introducción:

Uno de los mayores costes en la industria de la creatividad digital consiste en generar modelos tridimensionales. Además, gracias al avance en las tecnologías de renderizado y el aumento de la potencia de los sistemas de visualización, y sobre todo en la industria de los videojuegos, cada vez se necesitan modelos con mayor nivel de detalle, así como un mayor volumen de estos.

Por esto, muchas grandes empresas de la industria dedican una gran parte de sus presupuestos y personal exclusivamente a la creación de estos contenidos, apartándose por lo tanto de otras áreas del producto, de igual o mayor importancia. Del mismo modo, las empresas más pequeñas de la industria no pueden competir con las grandes empresas en cuanto a calidad o cantidad de modelado, por lo que terminan con productos de menor calidad en este aspecto.

Debido a esto, se investiga cada vez más en el campo de la generación procedural aplicada al modelado tridimensional, para que el esfuerzo económico y la carga de trabajo en este campo se reduzca, permitiendo a las empresas del sector reorientar sus esfuerzos y presupuestos a mejorar otros aspectos de sus productos.

En este trabajo voy a hablar brevemente de los dos principales tipos de gramáticas utilizadas en la generación procedural de modelos tridimensionales, para finalmente presentar una aplicación escrita en Python, que genera y renderiza modelos tridimensionales de árboles en el entorno grafico del software Blender, utilizando sistemas de Lindemayer.

Shape Grammars

Las gramáticas de forma, o shape grammars, son un tipo de gramática que utiliza producciones para generar formas geométricas, ya sean de 2 o 3 dimensiones. El concepto fue definido por George Stiny y James Gips en 1971, en su artículo: "Shape Grammars and the Generative Specification of Painting and Sculpture".

Este tipo de gramáticas están compuestas por un conjunto de reglas, o producciones, que se aplican a un modelo inicial, modificando su forma, posición u orientación. Estas reglas están compuestas por dos partes, a la izquierda, una condición, compuesta por una forma y un puntero, y a la derecha las transformaciones que se deben aplicar a la parte izquierda y donde se colocara el puntero al terminar.

En el caso de este tipo de gramáticas, el proceso de aplicar las producciones se puede realizar tanto en serie, buscando una coincidencia en el modelo con alguna de las reglas y aplicando solo esa regla, o en paralelo, aplicando todas las reglas a todas las coincidencias encontradas en el modelo.

Este tipo de gramáticas se utiliza sobre todo para la generación procedural de edificios y fachadas, así como planos y maquetas de arquitectura, ya que, por su funcionamiento, los modelos resultantes son muy rectangulares y siguen un patrón de cuadricula y poco detallado. Esto hace que las gramáticas de forma no sean aptas para generar modelos orgánicos, pero sí que tienen un lugar en la generación de modelos arquitecturales sin muchos detalles.

Los Sistemas de Lindenmayer

Los sistemas de Lindenmayer, o L-Sistemas, son gramáticas formales introducidas en 1986 por el botánico teórico Aristid Lindenmayer. El objetivo principal de estos sistemas era simular el crecimiento de plantas y animales.

La naturaleza recursiva de estos sistemas hace que sea sencillo generar fractales, y las formas orgánicas de plantas son también fáciles de generar, ya que el nivel de recursión de estos sistemas crece con cada iteración, produciendo formas más complejas y orgánicas.

La definición formal de estos sistemas es:

$$\mathbf{G} = \{V, S, \omega, P\}$$

Donde V es el conjunto de símbolos no terminales, o variables.

S es el conjunto de símbolos terminales.

ω es el estado inicial.

P es el conjunto de producciones del sistema, formadas cada una por dos partes, la parte izquierda, compuesta por una o más variables, y la parte derecha, compuesta por un conjunto de variables y símbolos terminales.

A pesar de la similitud de los sistemas de Lindenmayer con las gramáticas de Chomsky, se diferencian fundamentalmente en que, en las gramáticas de Chomsky, las producciones se aplican a las cadenas de forma lineal, aplicando una única regla a un único elemento de la cadena. En los sistemas de Lindenmayer, las producciones se aplican de forma paralela, simultáneamente a todos los elementos de la cadena para los que exista una regla aplicable.

En el caso de este tipo de gramáticas, el resultado de aplicar sucesivamente una serie de producciones sobre una palabra es una cadena de caracteres, formada tanto por símbolos terminales como variables de la gramática, por lo tanto, es necesaria una forma de interpretar gráficamente estos símbolos para poder generar modelos, ya sean de dos o tres dimensiones. En este caso, la forma más sencilla de interpretar las palabras generadas por estas gramáticas es utilizando un gráfico tortuga.

Para esto, la tortuga cuenta con una posición en el espacio, que puede ser tridimensional o bidimensional, y una orientación. La posición representa el punto en el espacio en el que la tortuga se encuentra en cada momento, y la orientación representa cual será la dirección y sentido del próximo movimiento de la tortuga.

Si se crea una gramática en la cual todos sus símbolos terminales y variables se corresponden con cada uno de los posibles movimientos de la tortuga, que son: avanzar y girar en el caso de un modelo en dos dimensiones, y avanzar, girar, subir y bajar en el caso de un modelo en tres dimensiones, se puede lograr una gramática

para la cual todo su vocabulario se componga de palabras que son instrucciones interpretables por este sistema, pudiendo utilizarse para generar graficas.

Además, para lograr una simulación del crecimiento de árboles y plantas más realista, se puede introducir en la gramática un símbolo que indique a la tortuga que debe guardar su actual posición y orientación, y otro que indique a la tortuga que debe volver a la última posición guardada, de forma que se generen ramificaciones en la representación gráfica de la tortuga.

Generación de modelos tridimensionales de árboles usando sistemas de Lindenmayer

A continuación, explicare el funcionamiento y los pasos seguidos para el desarrollo en Python de un programa que genera modelos tridimensionales de arboles utilizando sistemas de Lindenmayer.

Para llevar a cabo este proyecto, he utilizado como lenguaje de programación Python y el programa de software libre Blender como herramienta para realizar el post-procesamiento y el renderizado de los modelos generados.

El objetivo de esta aplicación es generar una cadena de caracteres, compuesta por caracteres pertenecientes a la gramática utilizada, que una vez generada será interpretada gráficamente utilizando una grafica tortuga. Para esto, la tortuga contara con un vector posición, dado como coordenadas cartesianas en el espacio tridimensional, y un vector orientación, dado también como un vector de 3 elementos.

En primer lugar, realice un prototipo del programa en Python utilizando la librería turtle, la cual permite de forma sencilla representar gráficamente las cadenas de símbolos generadas por un l-sistema a través de la pantalla. Para la realización de esta prueba, la gramática utilizada está compuesta por los siguientes elementos:

Una variable: ‘f’.

Un conjunto de símbolos terminales: ‘+’, ‘- ‘, ‘[‘y ‘]’.

Una palabra inicial: “f”.

Una sola regla: f -> f[+f[+f-f][-f+f]][-f[-f+f][+f-f]]fffff

La variable ‘f’, junto con el conjunto de símbolos terminales, se utilizan para determinar el movimiento y la orientación de la tortuga a la hora de representar el grafico 2d generado usando la palabra resultante de aplicar la regla a la palabra inicial un número determinado de veces. Es necesario indicar el numero de iteraciones que debe realizar el programa ya que, al tener solamente una regla en la gramática, y esta ser no productiva, nunca se podrá generar una palabra compuesta únicamente por símbolos terminales.

Este primer prototipo cuenta con 4 funciones:

La función aplicarRegla, que recibe un carácter como parámetro, comprueba si existe alguna producción aplicable a este carácter, y retorna el resultado de aplicar dicha producción, o simplemente el carácter recibido, en el caso de no poder aplicar ninguna producción.

La función aplicarProducciones, que recibe como parámetro un string e itera sobre sus caracteres, llamando a la funcion aplicarRegla con cada uno de ellos, y concatenando el resultado en un nuevo string, que retorna al terminar las iteraciones.

La función creaSistema, que recibe dos parámetros, el numero de iteraciones, que será el numero de veces que se procese la palabra inicial, y el axioma, o palabra inicial. Esta función realiza tantas iteraciones como se haya especificado en el primer parámetro, y en cada una llama a la función aplicarProducciones, pasando como parámetro el axioma recibido, en la primera iteración, y el resultado procesar esta palabra en las siguientes iteraciones. Finalmente, al terminar las iteraciones, se retorna el string resultante.

La función pintaSistema, que recibe cuatro parámetros, un objeto de la clase tortuga, encargado de realizar la representación de la palabra final, el string que se va a representar, formado por los símbolos y variables del sistema, el ángulo de giro de la tortuga, en grados, y el tamaño de cada segmento que la tortuga pintará. De esta manera, cuando se lee el carácter 'f' de la palabra, se indica a la tortuga que avance recto, según su orientación, si se lee el carácter '+', la tortuga girara su orientación hacia la derecha el número de grados especificado, y con el carácter '- ', hacia la izquierda. Si se lee el carácter '[', la posición y orientación actuales de la tortuga se guarda en una pila, y cuando se lee el carácter ']', se extraen una posición y orientación esa pila y se mueve a la tortuga a esa posición, con la orientación extraída.

Por último, la función main, en la que se llama a creaSistema para crear la palabra a representar, después se crea el objeto de la clase tortuga, y se llama a pintaSistema con este objeto, la palabra creada y unos valores de ángulo de giro y tamaño de segmento de 80 y 15, respectivamente.

Con este programa, se pueden obtener representaciones en dos dimensiones de gráficos similares a árboles, como los siguientes ejemplos:

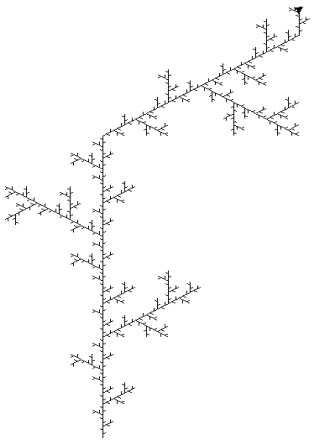

Imagen 1: Generado usando la regla F→F[+F]F[-F]F, en 5 iteraciones

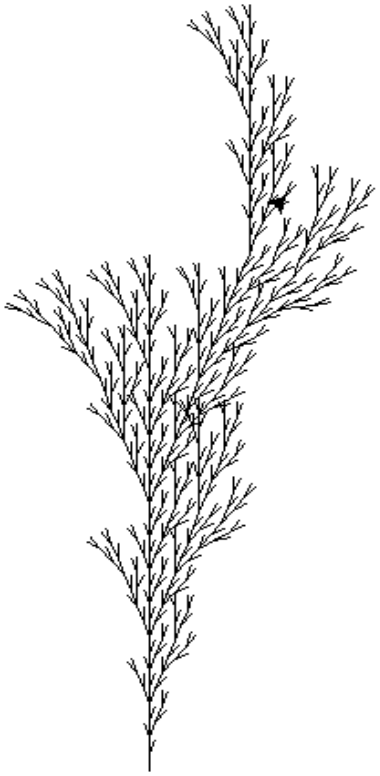

Imagen 2: Generado usando la regla F →F[+F]F[-F][F], en 5 iteraciones

Una vez comprobado que este sistema podía generar estructuras interesantes, se procede al siguiente paso, implementar este modelo de generación en el entorno tridimensional del programa Blender, para lograr la generación procedural de modelos tridimensionales de árboles.

Para la realización de la implementación en Blender de este prototipo, he realizado mi propia implementación de la tortuga, puesto que no existía ninguna forma de traducir el output de la librería turtle de Python al entorno de Blender. Para esto, he generado las palabras para crear los modelos de la misma forma a como lo hice en el prototipo, manteniendo las funciones aplicarProducciones y aplicarRegla, la primera con el mismo funcionamiento que en el caso anterior, y la segunda implementando también varias producciones para una misma variable, seleccionando una de forma aleatoria. Para la parte de crear el modelo tridimensional, he desarrollado una función llamada makePoly, que recibe como parámetro la palabra a partir de la cual crear el modelo.

El objetivo de esta función es, partiendo de un punto inicial en el espacio de coordenadas (punto 0,0,0) calcular y guardar en una lista las posiciones de todos los vértices y aristas del modelo tridimensional que se está generando. Para ello, se cuenta con una variable *posición*, que es una lista de coordenadas cartesianas x,y,z, y representa la posición actual de la tortuga en el espacio, y una variable *orientación,* también una lista, de tres elementos, correspondiendo cada uno a un eje (x,y,z), pero en este caso, cada elemento solo puede tomar como valores el -1, el 0 o el 1, donde el -1 simboliza orientación en el sentido negativo del eje, el 0 la orientación neutra en el eje, y el 1 la orientación en sentido positivo en el eje.

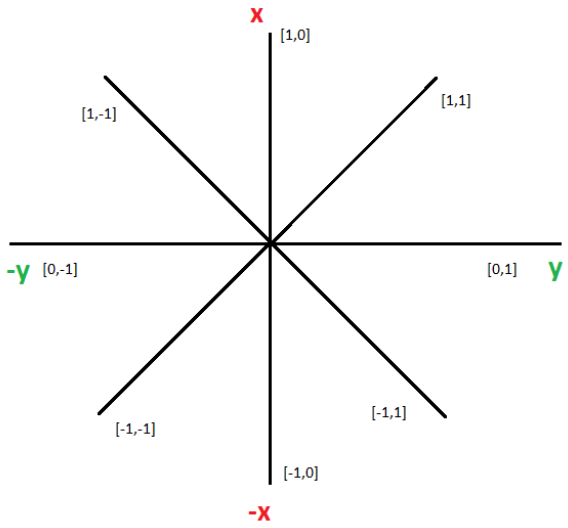

Imagen 3: Representación del vector orientación de la tortuga en el plano

Con todo esto, se comienza a iterar sobre cada uno de los caracteres de la palabra generada por las funciones anteriores, realizando las siguientes acciones para cada carácter:

Para el carácter 'f': La tortuga avanza en línea recta según su actual orientación, y partiendo de su actual posición, calculando el punto de destino, y estableciéndolo como su nueva posición, se añade la nueva posición a la lista de vértices del modelo, solamente en el caso de que no exista ya un vértice en esa posición y finalmente se crea una arista entre el nuevo vértice y el vértice anterior de la lista, siempre que el vértice anterior no fuese una hoja del árbol, en cuyo caso la arista se crea entre el vértice nuevo y el ultimo vértice antes de la última ramificación del árbol.

Para el carácter ‘[‘: Este carácter simboliza el inicio de una ramificación, por lo tanto, se guardan la posición y orientación actuales de la tortuga en una pila, para poder retornar más tarde a este punto.

Para el carácter ‘]’: Este carácter simboliza el final de una ramificación, por lo que se extraen de la pila la última posición y orientación de la tortuga, y además se prepara el programa para que el próximo vértice generado se marque como hoja.

Para el carácter ‘+’: Con este carácter, la tortuga gira hacia la derecha, tomando como referencia su actual orientación.

Para el carácter ‘-’: Con este carácter, la tortuga gira hacia la izquierda, tomando como referencia su actual orientación.

Para el carácter ‘*’: Este carácter indique que la tortuga cambie su orientación hacia arriba en el eje z, dependiendo de su actual orientación vertical.

Para el carácter ‘/’: Este carácter indique que la tortuga cambie su orientación hacia abajo en el eje z, dependiendo de su actual orientación vertical.

Estos son todos los movimientos posibles implementados para la tortuga. Una vez que se termina de procesar la palabra, los resultados se guardan en cuatro variables:

-Una lista de vértices, que contiene triplas con las coordenadas de cada uno de los vértices del modelo.

-Una lista de aristas, que contiene tuplas donde cada elemento es el índice de un vértice en la lista de vértices. Una tupla de la forma (1,2) indica una arista entre el vértice en la posición 1 y el vértice en la posición 2 de la lista.

-Una lista de índices de vértices, que contiene todos los vértices del modelo en los que se produce una ramificación del árbol.

-Una lista de índices de vértices, que contiene todos los vértices del modelo que corresponden a hojas del árbol.

Hasta este punto, solamente se han calculado los puntos en el espacio que corresponden a cada vértice, y las asociaciones entre ellos, por lo tanto, es el momento de usar estos datos para renderizar un modelo tridimensional. Para esto, es necesario recurrir a la librería bpy, propia de Blender, que contiene las funciones necesarias para renderizar el modelo en el entorno de este software.

En primer lugar, se genera el modelo a partir de la lista de vértices y la lista de aristas, lo cual generara un modelo compuesto únicamente por puntos en el espacio y aristas uniéndolos.

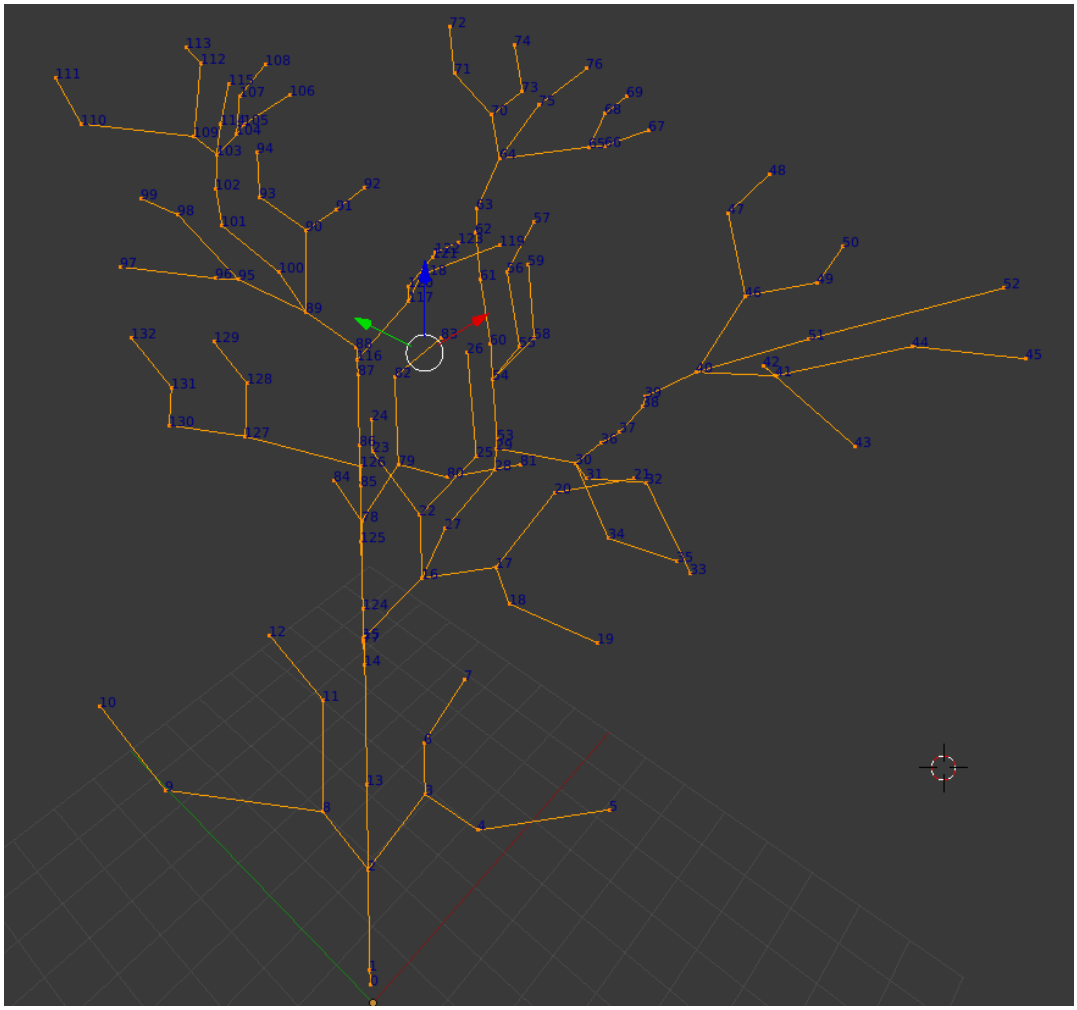

Imagen 4: Modelo generado a partir de los vértices y aristas calculados

Una vez generado este modelo, que se utilizara como esqueleto inicial para el modelo del árbol, se aplican una serie de modificaciones. En primer lugar, se realiza la subdivisión, con un factor de 1, del modelo. Esto quiere decir por cada dos vértices del modelo, unidos por una arista, se añade otro en el centro, suavizando en gran medida las formas del modelo y obteniendo resultados más orgánicos.

Después se aplica un modificador denominado Skin, cuyo efecto es el de generar geometría alrededor de los ejes del modelo, de forma que pasan de ser elementos unidimensionales a tener volumen. Hecho esto, se aplica de nuevo la subdivisión al modelo para obtener un aspecto aún más orgánico y suave.

Por último, se recorren todos los vértices del modelo modificado, modificando levemente el radio de cada uno, en un valor aleatorio, para dar más impresión de irregularidad, y además durante el recorrido se comprueba si cada vértice es una hoja, en cuyo caso, en el punto de ese vértice se crea un poliedro regular, de entre 20 y 80 caras, a modo de hojas del árbol.

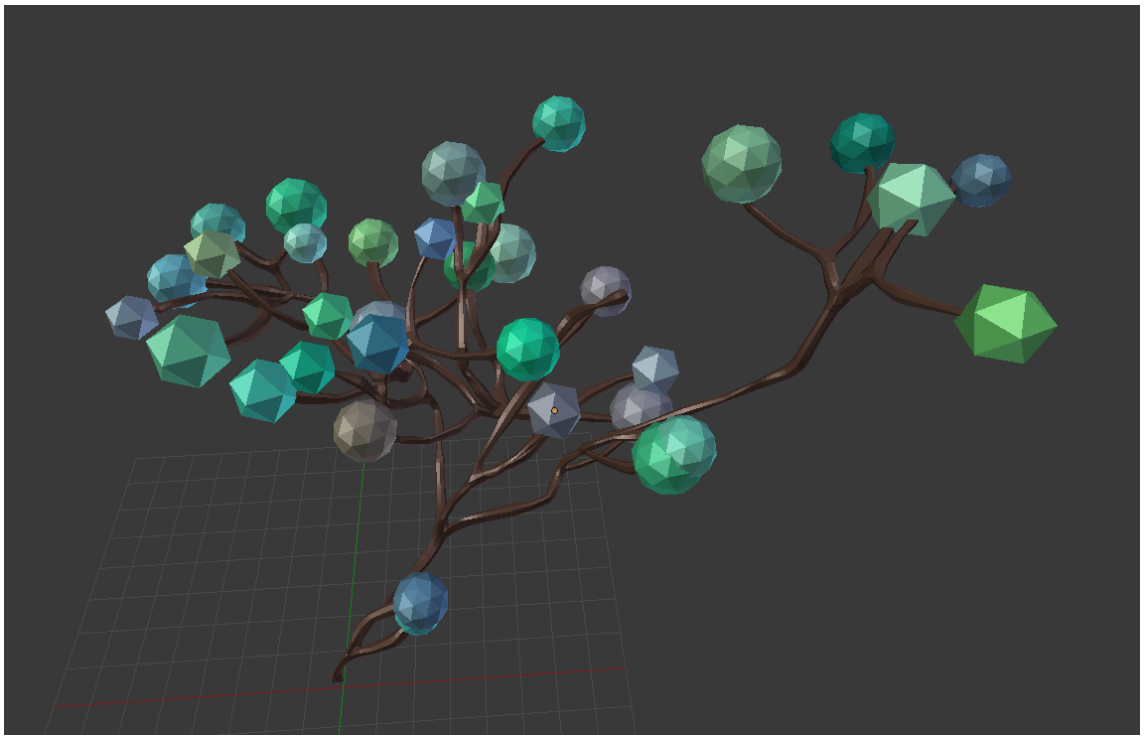

Imagen 5: Modelo de árbol procedural tras las modificaciones

Bibliografia

ROBERT F. TOBLER, STEFAN MAIERHOFER, ALEXANDER WILKIE. Mesh-based parametrized l-systems and generalized subdivision for generating complex geometry. International Journal of Shape Modeling.

Hansenmeyer, M., Dillenburger, B. (2013) Mesh grammars, Procedural articulation of Form.

Leopol, N. (2016) Algorithmische Botanik durch Lindenmayer Systeme in Blender.

Montell Serrano, J. A. (2017) Generación procedural de modelos tridimensionales de naves espaciales. Universitat Politècnica de València

Ortega de La Puente, A. (2000) Equivalencias entre algunos sistemas complejos: Fractales, Autómatas Celulares y Sistemas de Lindenmayer. Universidad Autonoma de Madrid