UNIVERSIDAD AUTÓNOMA METROPOLITANA UNIDAD AZCAPOTZALCO
DIVISIÓN DE CIENCIAS BÁSICAS E INGENIERÍA
Ingeniería en Computación

Generación de polígonos con triangulaciones ortogonales

Nombre : *Olaguibert Segura Carlos Alberto*
Matrícula : 203307487
Trimestre : 09I
Fecha de entrega : 15 de abril de 2009

Profesor : *Dr. Francisco Javier Zaragoza Martínez*
Número económico : 20197

Reporte final del proyecto terminal
"Generación de polígonos con triangulaciones ortogonales"

Olaguibert Segura Carlos A.

15 de abril de 2009

Agradecimiento.

Deseo agradecer primeramente a mi madre y mi abuela†, que con esfuerzo y trabajo siempre apoyaron dentro de sus posibilidades mis estudios.

En segundo lugar a mi esposa, que siendo solo mí novia creyó en mi cuando ya casi nadie lo hacia y me motivo para retomar mis estudios, ahora teniendo como cede esta gran universidad.

Por ultimo pero no por eso menos importante, deseo agradecer a mis profesores, que me han brindado conocimientos invaluables que me han permitido ponerme a un nivel realmente competitivo, pero en especial a quienes me brindaron su confianza, amistad y apoyo, Hugo Moncayo López†, Silvia B. González Brambila, Germán Téllez Castillo, Carlos Germán Pavía Miller y mi a asesor y amigo Francisco Javier Zaragoza Martínez.

† En paz esten.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción.

Existen muchos problemas en diferentes áreas del conocimiento que estudian las formas geométricas a partir de figuras básicas, ya sea en crecimientos de tejidos en biología, topografía en ingeniería civil, formación de figuras en diseño, graficación en computación, etc., entre muchas otras.(Ejemplo 1.1 [1])

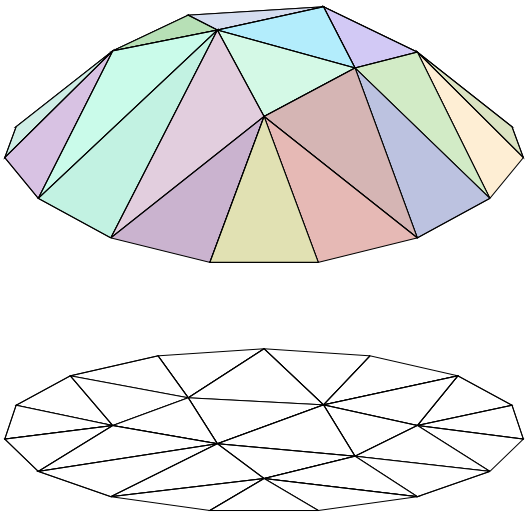


Figura 1.1: Ejemplo de gráficos por computadora.

La triangulación es una de las maneras en las que se pueden estudiar éstas

[1]Ejemplo tomado de: http://es.wikipedia.org/wiki/M %C3 %A9todo_de_los_elementos_finitos

formas geométricas. La triangulación consiste en colocar puntos sobre alguna superficie y posteriormente unir todos esos puntos formando triángulos.

En este proyecto se ha creado un programa que nos permite conocer cuantas formas o figuras diferentes (no isomorfas), pueden generarse a partir de una cierta cantidad de triángulos unitarios ortogonales.

1.1. Descripciones.

Considere los puntos de coordenadas enteras sobre el plano a los que de ahora en adelante les llamaremos simplemente enteros. Un triángulo se dice ortogonal si tiene sus tres vértices enteros, uno de sus ángulos mide 90 grados y sus dos lados perpendiculares miden 1. En otras palabras, un triángulo ortogonal no es otra cosa que un cuadrado unitario partido por la mitad.

Ahora considere que tiene N triangulos ortogonales que no se traslapan y que forman un polígono P. Con ellos vamos a formar una gráfica $G = (V, E)$ como sigue: cada vértice de G es un triángulo ortogonal y cada vez que dos triángulos ortogonales compartan un lado entonces unimos los vértices correspondientes de G con una arista. Ahora diremos que G es una gráfica ortogonal y que P es un polígono ortogonal.

Es de gran interés para diversos investigadores responder a las siguientes preguntas:

1. ¿Cuántos polígonos ortogonales conexos no congruentes se pueden construir con N triángulos ortogonales? Por ejemplo, para $N = 1$ sólo se puede construir uno (Figura 1.2) y para $N = 2$ sólo se pueden construir tres (Figura 1.3).

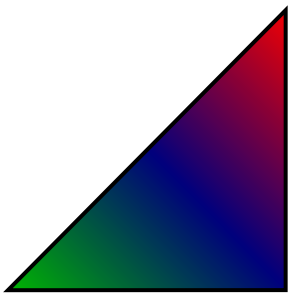


Figura 1.2: Para $N = 1$

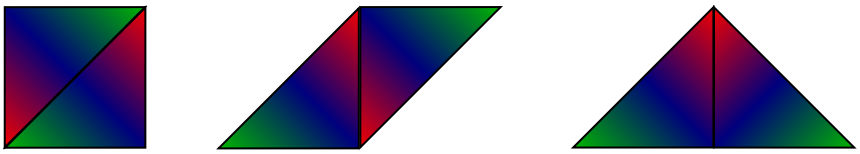


Figura 1.3: Para $N = 2$

2. ¿Cuántas gráficas ortogonales conexas no isomorfas se pueden construir con N triángulos ortogonales? Por ejemplo, para $N = 1$ sólo se puede construir una y para $N = 2$ también sólo se puede construir una.

Las respuestas a estas preguntas sólo se conocen para valores muy pequeños de N. El propósito de este proyecto es el de ayudar a responder al menos la primera de estas preguntas para valores un poco más grandes de N a través de un programa de cómputo.

1.2. Antecedentes.

Las triangulaciones ortogonales son un caso especial de las triangulaciones unimodulares (Figura 1.4), que son aquellas formadas por triángulos cuyos vértices son enteros y no contienen otros puntos enteros. Estos triángulos tienen la propiedad de que su área es siempre $\frac{1}{2}$.

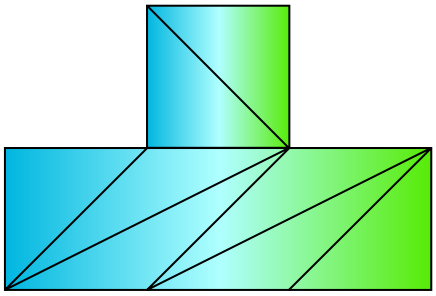


Figura 1.4: Ejemplo tiangulación unimodular

Las triangulaciones unimodulares han sido estudiadas por científicos de la computación y matemáticos.

Capítulo 2

Aspectos Técnicos del Proyecto.

2.1. Descripción Técnica.

Este proyecto fue desarrollado sobre el lenguaje de programación Java[1]. Consiste de un archivo de código fuente el cual contiene las clases necesarias para su buen funcionamiento.

Para poder ejecutar este programa es necesario tener instalada y configurada la JRE (Java Runtime Environment). Adicionalmente si se desea procesar la salida como documento, se necesitará instalar algún compilador de LaTeX[2] (Miktex[3] para Windows®) con el paquete Ti*k*z[4].

El funcionamiento de este programa es sobre línea de comandos, mediante la ejecución de la instrucción:

$ java proyectoterminal.Main <Numero> <Nombre>

Donde *<Numero>* es el número de triángulos del cual se quiere obtener el resultado del numero de figuras no isomorfas y *<Nombre>* es un nombre que se asignará a la carpeta de resultado y al archivo resultado.

La salida del programa es una carpeta que contiene un archivo con extensión *tex* el cual contiene en lenguaje LaTeX combinado con Ti*k*Z el número de figuras que pueden formarse con ese numero de triángulos y dichas figuras.

[1] http://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_Java

[2] Lenguaje para definición de textos: http://es.wikipedia.org/wiki/LaTeX

[3] Compilador LaTeX: http://www.miktex.org/

[4] Lenguaje para definicion de formas: http://wiki.contextgarden.net/TikZ

Adicionalmente se proporciona un script (.sh y .bat), con los cuales podrán llevar a cabo la compilación de los documentos a PDF (teniendo instalado el software pertinente).

2.2. Especificaciones Técnicas.

2.2.1. Representación lógica.

La representación de las figuras esta dada por 2 figuras básicas, un triángulo unitario, y un cuadrado unitario, el cual representa la union de 2 triangulos unitarios por medio de sus respectivas hipotenusas.

El algoritmo contempla 4 posibilidades de triangulo los cuales se muestran el la figura 2.1.

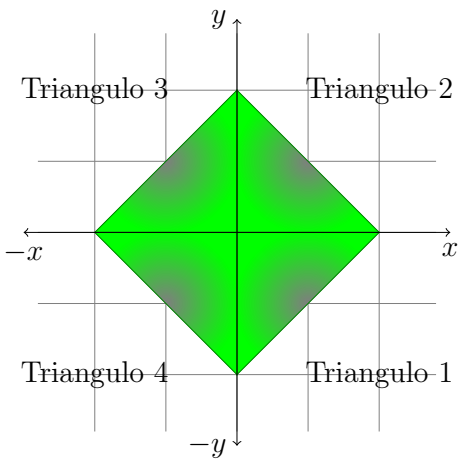


Figura 2.1: Triángulos que se pueden representar

Estos triángulos pueden generar cualquier combinación de figuras, a excepción de cuando 2 triángulos se unen por sus hipotenusas, por lo cual nace la necesidad de colocar una figura más, un cuadrado unitario, el cual corresponde a colocar 2 triángulos en alguna de las posiciones mostradas por la figura 2.2. Cabe mencionar que no importa en que posición se encuentre ésta es la misma figura por todos lados.

Cada figura tiene asociado un número (fig 2.3), y asú mismo cada figura final tiene asociada una matriz descriptora, que representa las posiciones de

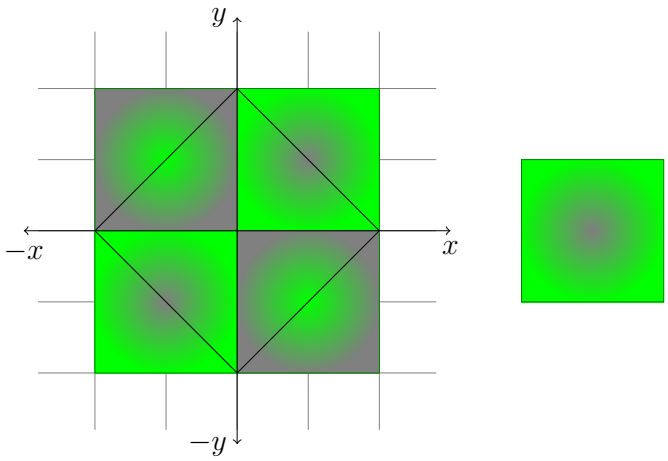


Figura 2.2: Cuadrado unitario.

cada una de las figuras básicas dentro del plano empezando por el origen, como se muestra en el siguiente ejemplo:

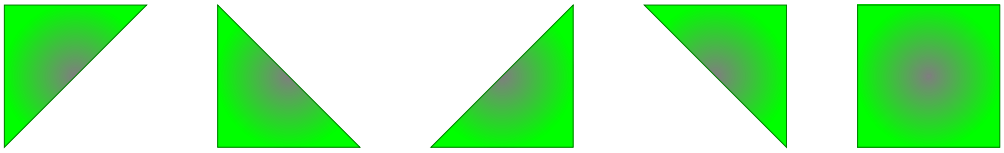
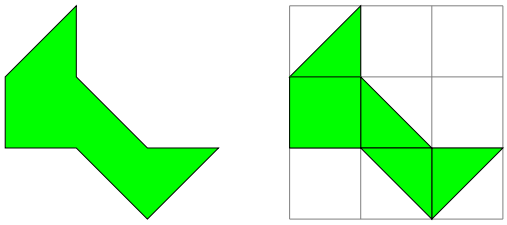


Figura 2.3: Asociación numérica de las figuras básicas.

A la figura de 6 triangulos que se muestra a continuación (Fig. 2.4), le correspondería la siguiente matriz, si nos damos cuenta esta matriz puede ser asociada únicamente a ésta figura.

Es preciso aclarar que aunque la figura del cuadrado (5) es una sola representación equivale a 2 triángulos unitarios que es la base que estamos tomando para el algoritmo.



3	0	0
5	2	0
0	4	1

Figura 2.4: Representación lógica de las figuras.

2.2.2. Estructura del programa.

El programa consiste de 3 clases:

Coordenada: esta clase solo define una coordenada en el plano (x, y)

Figura: Esta clase se encarga de contener la representación de las figuras, asi como sus coordenadas.

Main: Es la clase principal del programa, es la encargada de contener el codigo principal.

Clase *Coordenada***:**

Esta clase consta un constructor y tres métodos.
Constructores

Coordenada(*int x***,** *int y***):** Este constructor recibe dos números enteros que son las coordenadas que se desea representar. Solo se admiten enteros ya que las coordenadas serán utilizadas para acceder a las diferentes matrices descriptoras.

Métodos

getAbcisa(): Este método obtiene la correspondiente coordenada en el eje de las x.

getOrdenada (): Este método obtiene la correspondiente coordenada en el eje de las y.

toString(): Sirve para crear una cadena de caracteres imprimible que represente la coordenada..

Clase *Figura*:

Esta clase consta un constructor, cinco métodos de clase o estáticos y cinco métodos de instancia.

Constructores

Figura(*Coordenada punto*,*int representacion*, *Figura siguiente***):** Crea una figura en una coordenada especificada, la figura puede ser del tipo 1, 2, 3, 4 o 5, y adicionalmente se le pasa otra figura que será añadida a ésta. Si la figura es inicial en el parámetro *siguiente* puede pasársele un valor *null*[5].

Métodos de clase o estáticos

Init(*int maxx*, *int maxy***):** Sirve para inicializar la matriz asociada a la figura, en un tamaño máximo. Es necesario inicializar antes de utilizar el método *reduce*().

seCruzan(*Figura a*, *Figura b***):** Este método regresa un valor lógico (*true* o *false*) indicando si las figuras a y b se cruzan en algún punto.

sonIguales(*Figura a*, *Figura b***):** Indica con un valor lógico si las dos figuras son iguales en todo su recorrido, tanto en figuras como en coordenadas. Respeta el orden en el que las figuras fueron agrupadas.

sonMismaFigura(*Figura a*, *Figura b***):** Este método regresa un valor lógico (*true* o *false*) indicando si la figura *a* y la figura *b* son la misma figura, sin importar el orden de agrupamiento. Este método solo debe ser usada después de haber reducido ambas figuras, ya que trabaja a partir de las matrices descriptoras..

sumaFiguras(*Figura a*,*Figura b***):** Este método adiciona a la figura *a*, la figura *b* al final de su estructura, y regresa una *Figura* que corresponde a la figura *a*, ya con b al final.

Métodos de instancia

getTamanio(): Regresa el tamaño que tiene esta figura, en figuras unitarias (triángulos y cuadrados).

obtenTodas (): Una vez que la figura ha sido reducida este método obtiene todas las figuras que pueden ser sacadas de la original contándola, figuras rotadas, reflejadas y ambas.

[5]Este es un valor definido en el lenguaje Java y representa *sin valor*.

reduce(): Cuando una figura se genera, ésta puede estar fuera del origen, y/o tener elementos no validos dentro de su estructura, por esta razón no se le puede asignar una matriz descriptora. Cuando este método es invocado, traslada la figura al origen y elimina los elementos no validos generados de forma automática, además de asignarle una matriz descriptora, esta matriz tiene el tamaño exacto de la figura sin exceder sus dimensiones.

siguienteFigura (): Regresa la figura siguiente en el orden en el que fueron anidadas. Hay que tener claro que solo es posible ir en un sentido, si se pierde la referencia de la figura principal, no se podrá regresar a ella.

toStringTex (): Este método obtiene una cadena de caracteres que describe en Ti*k*Z cada una de las figuras, a este código solo hay que agregar las sentencias: $\backslash begin\{tikzpicture\}$ y $\backslash end\{tikzpicture\}$ al inicio y final del texto respectivamente y clocarlo dentro de algún archivo *.tex*, para que la figura sea dibujada.

Clase *Main***:**

Esta clase consta un constructor, un método de clase o estáticos y un método de instancia.

Constructores

Main (*int triangulos***):** Recibe el número de triángulos a partir del cual se quiere trabajar.

Métodos de clase o estáticos

main (*String[] args***):** Es el método principal, este es el método que se ejecutará al principio de la ejecución, recibe como parámetros los argumentos que se le pasen desde línea de comandos.

Métodos de instancia

triangulacion (*Coordenada punto***,** *int figura***,** *int restantes***):** Es el encargado de buscar todas las combinaciones que pueden ser creadas a partir de la figura que se le pasa. El parámetro *punto* indica la posición original de la primer figura (normalmente se le pasa (*restantes*, *restantes*) en la primera ejecución), *figura* es un valor entre 1, 2, 3, 4 y 5, que describen las figuras unitarias, es el valor de la siguiente figura a colocar si es que se puede y *restantes* indica cuantas figuras faltan por colocar.

2.3. Itinerario trabajo.

En la siguiente tabla se describe el modo y tiempo de trabajo:

Etapa	**Horas**	**Descripción**
Diseño e implementación del algoritmo para generar polígonos ortogonales conexos.	98	Este algoritmo fue diseñado haciendo uso de las técnicas de programación "Backtraking", "Programación dinámica", "Búsqueda a lo ancho" y "Búsqueda en profundidad".
Diseño e implementación del algoritmo que permite encontrar todas las formas que puede tomar una figura.	90	Este algoritmo hace uso de la técnica de "Programación dinámica", ya que en su primera versión se realizo con recursividad, pero el tiempo de ejecución era alto, así que realizando un análisis un poco mas profundo se encontró un algoritmo mucho mas sencillo que proporcionaba los mismos resultados, haciendo uso de las matrices descriptoras.
Diseño e implementación del algoritmo de desición (si una figura es igual que otra).	27	Este algoritmo fue sencillo realizarlo, ya que se tenían todas las figuras que se pueden generar a partir de un cierto numero de triángulos, y todas las formas que puede tomar cada figura, solo fue necesario comparar cada posible solución con las posibilidades que generaban todas las soluciones anteriores, sino es igual a ninguna entonces también es solución.
Pruebas y obtención del catálogo	45	Las pruebas fueron realizadas conforme se fueron generando los algoritmos, y al terminar de unirlos se realizaron pruebas de funcionamiento global, en las cuales se detectaron algunos errores y fueron corregidos de forma satisfactoria en cada caso.
Elaboración del reporte final	18	.

Cuadro 2.1: Itinerario de trabajo expresado en horas.

Capítulo 3

Conclusiones.

Se realizo la búsqueda de figuras hasta para 9 triángulos unitarios, cada una de las ejecuciones fueron realizadas 3 veces para llevar a cabo el calculo del tiempo promedio, estas ejecuciones fueron realizadas sobre una computadora con procesador Intel® Core™2 Duo, 1Gb de memoria RAM y sistema operativo Linux Ubuntu 8.10 desktop x64[1], obteniendo los resultados que se muestran en la siguiente tabla:

Numero de triángulos unitarios	**Numero de figuras no isomorfas que se pueden formar**	**Tiempo de ejecución promedio**
1	1	-
2	3	0.086 Segundos
3	4	0.1 Segundos
4	14	0.459 Segundos
5	30	2.023 Segundos
6	107	6.88 Segundos
7	318	1 Minuto 12.856 Segundos
8	1116	13 Minutos 1.184 Segundos
9	3728	2 Horas 59 Minutos 8.712 Segundos

Cuadro 3.1: Resultados.

[1]Para procesador de 64 bits de longitud de palabra.

Bibliografía

[1] Peter Shirley, "Fundamentals of Computer Graphics" , 2005.

[2] James D. Foley, "Introduction to Computer Graphics" , 1994.

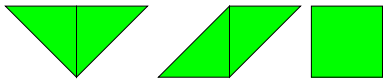
[3] Wikipedia, "Tangrama", http://es.wikipedia.org/wiki/Tangrama, *Wikipedia*, 19 de Noviembre del 2008.

[4] Jesús A. De Loera, "Triangulations: Structures and Algorithms", *http://www.math.ucdavis.edu/˜deloera/BOOK/Nov2008version.pdf*, Version preliminar - Noviembre 11 del 2008

[5] F. Hurtado and M. Noy. "Counting triangulations of almost-convex polygons.", 1997.

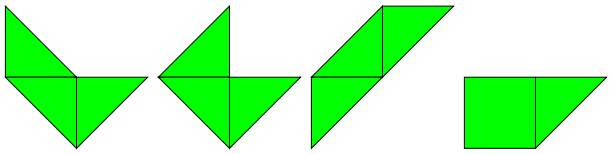
Apéndice A

N = 2



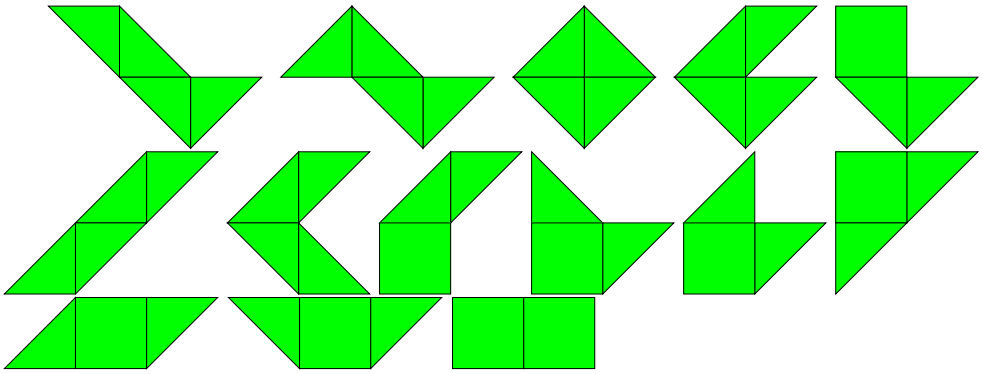
Apéndice B

N = 3



Apéndice C

N = 4



Apéndice D

N = 5