

UNIVERSIDAD AUTÓNOMA DE CIUDAD JUÁREZ

Instituto de Ingeniería y Tecnología

Departamento de Ingeniería Eléctrica y Computación



GENERACIÓN DE CIUDADES VIRTUALES EN 3D POR MEDIO DE
PROCEDIMIENTOS ALGORÍTMICOS

Reporte Técnico de Investigación presentado por:

Brenda Berenice Candia Rey 86355

Jesús Carlos Ruvalcaba Pérez 87670

Requisito para la obtención del título de

INGENIERO EN SISTEMAS COMPUTACIONALES

Profesor Responsable: Saúl González Campos

Mayo del 2012

Autorización de Impresión

Los abajo firmantes, miembros del comité evaluador autorizamos la impresión del proyecto de titulación

GENERACIÓN DE CIUDADES VIRTUALES EN 3D POR MEDIO DE PROCEDIMIENTOS ALGORÍTMICOS

Elaborado por los alumnos:

Brenda Berenice Candia Rey 86355

Jesús Carlos Ruvalcaba Pérez 87670

Fernando Estrada Saldaña
Profesor de la Materia

Saúl González Campos
Asesor Técnico

Declaración de Originalidad

Nosotros Brenda Berenice Candia Rey y Jesús Carlos Ruvalcaba Pérez declaramos que el material contenido en esta publicación fue generado con la revisión de los documentos que se mencionan en la sección de Referencias y que el Programa de Cómputo (Software) desarrollado es original y no ha sido copiado de ninguna otra fuente, ni ha sido usado para obtener otro título o reconocimiento en otra Institución de Educación Superior.

Brenda Berenice Candia Rey

Jesús Carlos Ruvalcaba Pérez

Dedicatoria

Este proyecto se lo quiero dedicar a mi mamá, quien me inspiró y me ayudó a ser la persona que soy. Su mayor deseo era verme graduar de la Universidad y estuvo a punto de lograrlo pero Dios ya le tenía preparado algo mejor. Esto es para ti mami.

Berenice Candia

Quiero dedicarle este proyecto a todo lo que nos mueve a seguir sobrepasando nuestros límites, a lo que nos recuerda lo pequeños que somos en el mundo. Quiero dedicárselo al poder.

Carlos Ruvalcaba

Agradecimientos

Primeramente que nada agradezco a Dios por permitirme llegar a este punto tan importante en mi vida. También quiero darle las gracias a mi papá y a mi hermana quienes me estuvieron alentando a seguir hasta estos momentos, los quiero mucho. Gracias a ti mami por darme todo lo necesario para ser una persona exitosa, siempre estarás en mi corazón.

Gracias a mi novio Raúl Mata quien prácticamente me acompañó en toda mi etapa de universidad y me dio ánimos para concluir de una manera exitosa como lo estoy haciendo en estos momentos. De corazón Gracias.

Por último quiero darle las gracias a mi compañero Carlos Ruvalcaba por su paciencia y apoyo en este proyecto, hicimos un buen equipo, muchas gracias.

Berenice Candia

Quiero agradecer a mi padre y madre, quienes primeramente me dieron la oportunidad de llegar a este mundo y vivir hasta estos momentos. Gracias por haberme apoyado de toda manera posible para poder estar en este instante escribiendo estos versos, por enseñarme a luchar por mis metas. Agradezco a mis hermanos por hacer de la vida más interesante y enseñarme el valor de la responsabilidad.

Agradezco a mis compañeros *maiceros* por darme la oportunidad de aprender a tomar cada aspecto de la vida como un reto, sutil y desafiante. Son la razón por la que sigo luchando y empujando hacia adelante para seguir creciendo. Muchas gracias.

Por último le doy gracias a mi compañera, de generación y de título, Berenice Candia, quien compartió su dedicación y tiempo a la realización de este proyecto tanto como yo, muchas gracias.

Carlos Ruvalcaba

Índice

Autorización de Impresión.....	iii
Declaración de Originalidad	iv
Dedicatoria	v
Agradecimientos	vi
Índice.....	vii
Lista de Figuras.....	ix
Lista de Tablas	xi
Introducción	1
Capítulo 1. Planteamiento del problema.....	2
1.1 Antecedentes	2
1.2 Definición del problema	6
1.3 Objetivos de la investigación	6
1.4 Preguntas de investigación.....	6
1.5 Justificación de la investigación	6
1.6 Delimitaciones y limitaciones de la investigación.....	7
Capítulo 2. Marco Teórico.....	8
2.1 Ciudades.....	8
2.2 Subdivisión iterada.....	11
Capítulo 3. Materiales y Métodos	14
3.1 Descripción del área de estudio	14
3.2 Materiales.....	14
3.3 Métodos.....	14
3.3.1 Metodología	15
3.3.2 Procedimientos.....	16
Capítulo 4. Resultados de la investigación	26

4.1 Presentación de resultados	26
4.2 Análisis e interpretación de resultados	30
Capítulo 5. Discusiones, conclusiones y recomendaciones	36
5.1 Con respecto a las preguntas de investigación.....	36
5.2 Con respecto al objetivo de la investigación.....	37
5.3 Recomendaciones para futuras investigaciones.....	37
Referencias.....	39
Apéndices.....	41
Anexo.....	51
Propuesta de investigación (Protocolo corto)	51

Lista de Figuras

Figura 1. En algún lugar de Manhattan virtual [1].....	2
Figura 2. Generación de primitivas que representan a los edificios [3].....	3
Figura 3. Se muestra como una ciudad va en crecimiento conforme pasan los años [4].	3
Figura 4. Paso a paso la generación de una ciudad [5].	4
Figura 5. Implementación de los campos tensoriales [6].....	4
Figura 6. Comparación de las asignaciones de diámetro pequeño para proporción de anchura (izquierda) y los más grandes (derecha) [7].	5
Figura 7. Utilización de gramáticas para la generación de las construcciones [8].	5
Figura 8. Tensor de estrés [9].	6
Figura 9. Plano irregular en el casco histórico de Granada. Barrio del Albaicín [10]...	9
Figura 10. Trazado ortogonal en el ensanche de Barcelona, área superior izquierda [15].	10
Figura 11. Plano de Moscú [17].	11
Figura 12. Posible resultado de la implementación de el algoritmo de subdivisión [7].	12
Figura 13. Algoritmo de subdivisión iterada [7].	12
Figura 14. Intersecciones solamente entre horizontales.	18
Figura 15. Ancho de las avenidas	19
Figura 16. Calles secundarias cuando los puntos inicial y final son correspondientes.	21
Figura 17. Calles secundarias cuando los puntos inicial y final no son correspondientes.	21
Figura 18. Zonas inválidas en 2 de las colonias.....	22
Figura 19. Se observa un desfase en las cuadras cuando alguno de sus lados presenta una inclinación.	23

Figura 20. Division de cuadras en lotes.	23
Figura 21. Resultado de las avenidas principales y las colonias.....	26
Figura 22. Resultado de calles secundarias y cuadras.	27
Figura 23. Resultado de calles secundarias y cuadras irregulares.	27
Figura 24. Cuadras coloreadas.	28
Figura 25. Resultado de los lotes.	28
Figura 26. Cuadras irregulares.	29
Figura 27. Vistas aéreas de las construcciones.	29
Figura 28. Resultados en cuanto a las zonas de construcción.....	29
Figura 29. Avenidas principales y colonias de Nueva York. [18]	30
Figura 30. Avenidas principales y colonias de la aplicación.	31
Figura 31. Calles secundarias y cuadras de Nueva York. [18]	31
Figura 32. Calles secundarias y cuadras de Ciudad Juárez. [18]	31
Figura 33. Calles secundarias y cuadras de la aplicación.	32
Figura 34. Distribución de lotes en El Paso, TX. [18]	32
Figura 35. Lotes con zona de construcción en la aplicación.....	33
Figura 36. Lotes en algunas cuadras de Cd. Juárez. [18].....	33
Figura 37. Edificios en el D.F. [18]	34
Figura 38. Construcción de 3 pisos en Guadalajara. [18]	34
Figura 39. Construcciones generadas en la aplicación.	35

Lista de Tablas

Tabla 1. Descripción de los conceptos básicos de la figura 13.....	13
Tabla 2. Descripción de las herramientas.	14
Tabla 3. Ejemplo de intersecciones.	18

Introducción

La elaboración de contenido virtual para proyectos gráficos, así como ambientes virtuales ó realidad virtual, siempre ha presentado desafíos y complejidades cuando se pretende crear ambientes muy extensos y detallados. La principal limitante es el tiempo, y por ende los recursos que éstos proyectos requieren cuando se intenta elaborar por medios convencionales. En éste proyecto, se describe una manera de resolver éste problema al elaborar una ciudad virtual por medio de algoritmos.

En el capítulo 1, se plantea la situación que se requiere resolver, se muestran algunos de los proyectos más sobresalientes anteriores a éste que pretendieron abarcar diferentes partes de la misma problemática e incluso diferentes formas de resolverla. Se plantean también el problema que presenta éste tipo de proyecto y los objetivos que debe alcanzar, delimitándolo hasta un cierto punto.

En el capítulo 2 se redacta el marco teórico sobre el que el proyecto se apoyará para lograr dar solución a los objetivos planteados y poder dar respuesta a las preguntas de investigación en conjunto con el desarrollo del mismo.

En el capítulo 3 se especifica el área de estudio, los materiales que el proyecto utilizará para ser llevado a cabo y los métodos que se siguieron y elaboraron para lograr llegar a obtener resultados satisfactorios en cuanto al alcance que el proyecto desea abarcar, esto es, llegar a alcanzar un nivel de realismo aceptable.

El capítulo 4 es la presentación de los resultados obtenidos y su comparación y análisis cualitativo en contra de imágenes de mapas de ciudades reales. Es esta sección la síntesis visual del capítulo 3.

Por último, en el capítulo 5 se discutirá el estado que el proyecto alcanzó y cómo, en proyectos futuros, puede mejorarse para alcanzar niveles de realismo más avanzados.

Capítulo 1. Planteamiento del problema

En esta sección se abordarán los puntos básicos que ayudarán a entender el problema que se tiene en la actualidad para llegar a crear una ciudad virtual por medio de métodos computacionales.

1.1 Antecedentes

En el 2001, Parish & Müller [1] presentaron el sistema de CityEngine. El proceso de modelación del sistema se enfoca principalmente en la utilización de *Lindenmayer systems (L-systems)* para generar el contenido de la ciudad virtual, básicamente consiste en una secuencia de pasos algorítmicos, los cuáles al final dan como resultado un modelo virtual en 3D de una ciudad.

En este sistema el usuario proporciona datos de entrada, como mapas geográficos y socio-estadísticos, que el sistema utiliza para proceder a generar el plano de calles que conformará a la ciudad virtual por medio de un *L-system* extendido, una vez obtenida la red de calles se procede a realizar la división de lotes, en los cuáles se generarán los modelos sólidos en 3D de las construcciones utilizando otro tipo de *L-system* para este propósito [1].



Figura 1. En algún lugar de Manhattan virtual [1].

El sistema de CityEngine es uno de los proyectos más maduros en la industria del modelado urbano, que hasta su versión del 2011 soporta funcionalidades como: importación de datos de Sistemas de Información Geográfica (SIG), formatos industriales estandarizados, núcleo de modelado basado en reglas, interface de modelación paramétrica, *wizard* para la generación de fachadas, patrones de redes de calles, soporte de *scripts* de Python, multiplataforma, entre otras [2].

Por otro lado, Greuter, Parker, Stewart & Leach [3], presentaron en su aplicación un enfoque diferente para generar ciudades virtuales en tiempo real, aparte de que tienen la característica de ser “pseudo-infinitas”. Esto se logró mediante la utilización de algoritmos que forman por extrusión las construcciones a partir de planos de pisos, estos planos se generan combinando iterativamente polígonos geométricos uno sobre otro. Además se hace uso de *Display Lists* y del *View frustum* para controlar el uso de recursos que la computadora utiliza al generar y visualizar la ciudad.

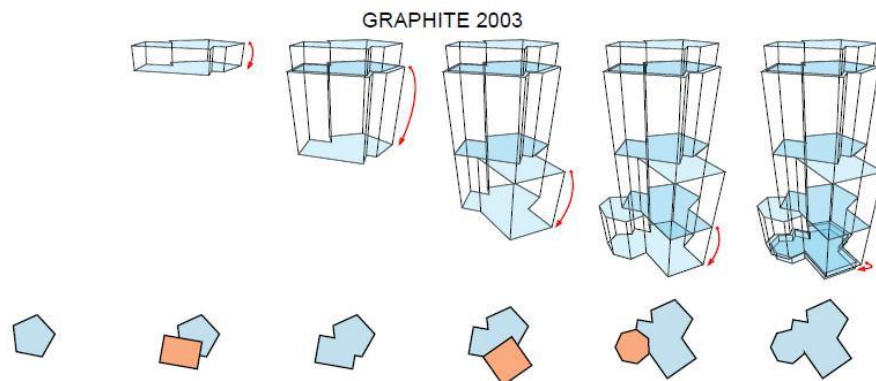


Figura 2. Generación de primitivas que representan a los edificios [3].

El sistema de simulación de Weber, Müller, Wonka & Gross [4], alcanza a simular el cambio de una ciudad conforme pasa el tiempo. Esto se hace posible gracias a que su sistema modela una ciudad virtual como un conjunto de objetos geométricos en lugar de como campos en una red que representan datos espaciales. Esto permite el simulado de la creación y modificación de nuevas redes de calles, de zonificación, de densidad poblacional y entre otras cosas.

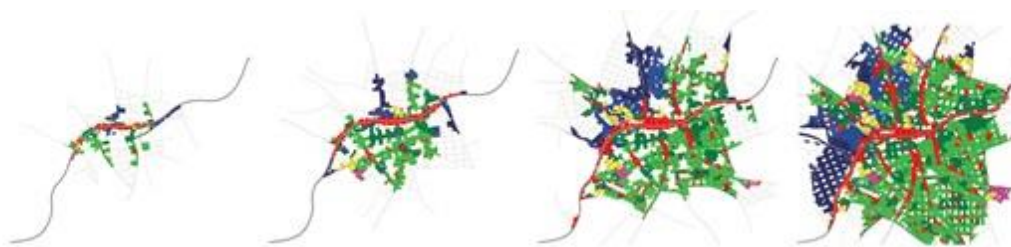


Figura 3. Se muestra como una ciudad va en crecimiento conforme pasan los años [4].

En el 2009, Groenewegen, Smelik, de Kraker & Bidarra [5] dan a conocer su aplicación de modelado de ciudades, enfocándose a auxiliar la creación de contenido para mundos virtuales utilizados en la educación, capacitación o entrenamiento. Su sistema permite generar una ciudad virtual en 3D basándose en la utilización de datos que tienen que ver con la división histórica de la ciudad y las características que definen

el uso de la tierra en cada sección de la ciudad, obteniendo de esta manera el estilo de patrones de calles y construcciones que se presentan en cada región o división.

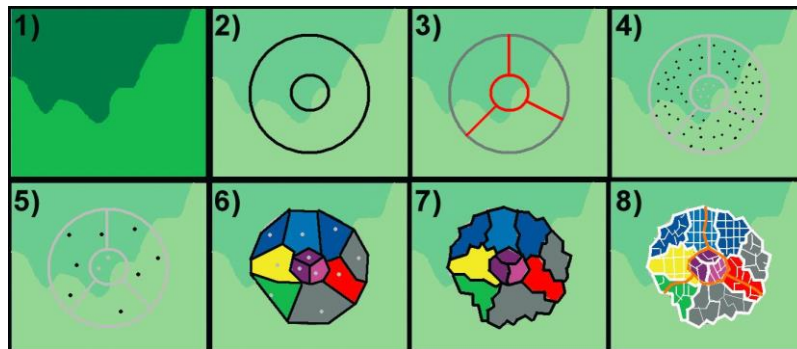


Figura 4. Paso a paso la generación de una ciudad [5].

Con un mayor enfoque a la generación de la red de calles, Esch, Chen, Wonka, Müller, & Zhang [6] presentan la forma de cómo generar redes de calles de manera interactiva y procedimental. Esto se hace mediante la utilización de campos tensoriales para generar un grafo que describirá a la red de calles, este grafo representa a la red como un conjunto de vértices y caminos, afirmando que generandola de esta manera es posible aprovechar la flexibilidad de los campos tensoriales para modificar y dar forma a la red de calles de una manera más personificada.



Figura 5. Implementación de los campos tensoriales [6].

Rudzicz & Verbrugge [7], presentan un método para generar planos de calles por medio de lo que llaman un algoritmo de subdivisión iterativa, enfocado en mantener la sencillez en cuanto a su descripción e implementación. Esta subdivisión es básicamente para obtener un mapa de calles. Se empieza con un polígono, por lo general grande, el cuál será del tamaño de la ciudad total, luego en cada iteración del algoritmo el área se divide en dos para obtener nuevos polígonos. A cada nuevo polígono se le hace una revisión detallada, esto es, observar si es muy grande o muy pequeño pasando del límite que se le asignó, con la finalidad de seguir subdividiendo.

Este tipo de subdivisión se hace cuantas veces sea posible partir un polígono, prácticamente es ir dividiendo en dos cada polígono que se va creando.

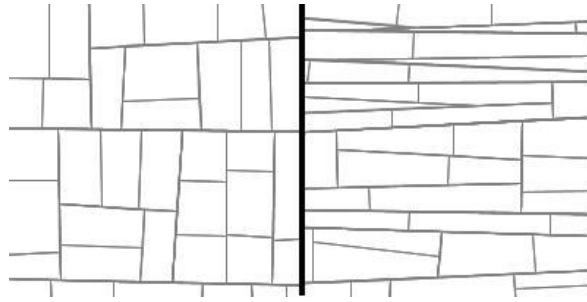


Figura 6. Comparación de las asignaciones de diámetro pequeño para proporción de anchura (izquierda) y los más grandes (derecha) [7].

En cuanto a la elaboración de las construcciones, Müller, Wonka, Haegler, Ulmer, & Gool [8] hacen uso de gramáticas para la generación de edificios y utilizan *CGA shape* en su aplicación, el cual es una extensión de gramáticas que hace uso de reglas de producción para generar las construcciones iterativamente. Esta gramática es una configuración de formas, cada forma posible que puede tomar un edificio está representada por un símbolo y un conjunto de atributos, donde la gramática es utilizada para generar el plano de cada piso que describe a la construcción. Finalmente por medio de extrusión se genera la edificación obedeciendo las reglas de producción de la misma gramática.



Figura 7. Utilización de gramáticas para la generación de las construcciones [8].

Para llegar a la realización de todo esto es necesario conocer más a detalle los campos tensoriales descritos por Zhang, Hays y Turk [9], estos se dividen en dos enfoques, el clásico y el moderno, el primero habla de la visualización de los tensores como arreglos multidimensionales, esto es, vectores de una dimensión y matrices de dos dimensiones. El segundo enfoque habla de visualizar los tensores inicialmente como objetos abstractos, usando un concepto multi-lineal.

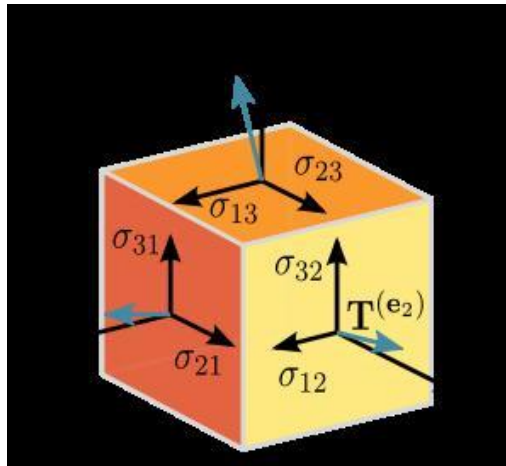


Figura 8. Tensor de estrés [9].

1.2 Definición del problema

La creación de grandes ambientes virtuales para la industria del entretenimiento toma mucho tiempo y esfuerzo si se realiza utilizando medios tradicionales como modeladores gráficos o grupos artísticos (modelación a mano). Además, la implementación de los modelos resulta compleja en cuanto a la programación requerida.

1.3 Objetivos de la investigación

Implementar un prototipo de aplicación que genere ciudades virtuales en 3D, utilizando una selección de técnicas y métodos procedimentales propuestos para lograr un nivel básico de realismo.

1.4 Preguntas de investigación

- ¿Cómo se genera una ciudad virtual por medio de algoritmos?
- ¿Cómo se obtiene el patrón ortogonal que se observa en algunas ciudades?
- ¿Cómo se delimitan los espacios de los lotes?
- ¿De qué forma se generan modelos tridimensionales que asemejen construcciones?

1.5 Justificación de la investigación

En diversos sectores de la industria, sobre todo en la del entretenimiento, existe la necesidad de poder crear contenido ambiental virtual (como una ciudad) minimizando la utilización de datos de entrada o el trabajo enteramente manual y creativo de diseño, lo cual consume tiempo y es costoso. Considerando lo anterior, se considera que es de gran utilidad la implementación de un procedimiento computacional que automatice en

buena medida este proceso y que el resultado tenga un nivel aceptable de realismo e incluso pueda ser personificado hasta cierto punto.

1.6 Delimitaciones y limitaciones de la investigación

El alcance de este proyecto es lograr la implementación de una metodología la cual en general consta de cinco fases. Delimitando el proyecto a las tres primeras para generar una ciudad virtual, éstas se enlistan a continuación:

- 1) Red de calles y carreteras.
- 2) Distribución de lotes.
- 3) Generación de construcciones.
- 4) Texturización.
- 5) Sistemas misceláneos (vegetación, personas, clima, tráfico).

Otro punto es que no se toma en cuenta las alturas del terreno, puesto que para hacerlo sería necesario utilizar e implementar otros métodos, de modo que sólo se trabajará sobre un plano.

Por otro lado, los polígonos que representan a las cuadras y lotes serán siempre cuadriláteros, la aplicación siempre generará el mapa de una ciudad en un área cuadrangular especificada.

Debido a que tomaría más tiempo del disponible, no se incluye un método que tome en cuenta restricciones geográficas, como montañas, ríos o lagos en donde regularmente no es posible construir.

Una limitación del proyecto es que debido al método seleccionado para crear los polígonos de la ciudad, no es posible crear curvas al momento de generar las calles y los lotes. En todo caso tendría que modificarse el método para lograr crearlas ó utilizar algún otro.

Capítulo 2. Marco Teórico

En esta sección se abordarán todas aquellas definiciones importantes así como los acontecimientos que se utilizarán en la investigación. Dando así la base fundamental para el desarrollo del proyecto.

2.1 Ciudades

Se entiende por ciudad a un área urbana la cual cuenta con una alta densidad de población y en su mayoría predominan la industria y los servicios. [15]

2.1.1 Morfología de las ciudades

El concepto de morfología urbana hace referencia a la forma y distribución en el espacio de los edificios urbanos. Existen diferentes formas de percibirla: según Golany [14] con el simple hecho de caminar sobre las calles uno puede darse una idea de la morfología, en cambio Gómez [13] dice que no es suficiente solo la vista para entender la morfología de la ciudad completa. Ya que es claro que lo que se observa es solo una mínima parte de toda la ciudad.

Entrando más a fondo y para entender mejor la morfología de la ciudad se encuentra el plano urbano, éste viene siendo una representación de la ciudad en todo su conjunto [13], es decir, un mapa en donde se puede observar con más detalle el trazado de las calles, la forma que toman las manzanas y la base de las construcciones.

2.1.2 Elementos de la morfología urbana

Como primer elemento se encuentra la disposición de las calles en donde se muestra cuál es el trazado de la misma, de igual manera se puede observar en qué zonas las calles son más amplias o más angostas, si son curvas o son rectas [10]. Lo importante aquí es entender que si un edificio se derriba se puede construir de diferente manera, en cambio las calles son más difíciles de modificar.

Como segundo elemento se tiene el tipo de edificios que predomina en la ciudad, siendo la visualización directa la mejor forma de analizar este elemento [10], ya que de esta manera se puede observar tanto la forma en que están diseñados como el material que utilizaron en sus construcciones, su altura, la decoración, etc. Aquellas construcciones que perduran durante siglos por lo regular son los monumentos y las catedrales las cuales generalmente se ubican en la zona centro de las ciudades.

Como último elemento, pero no menos importante, se encuentran las funciones de las calles y edificios [10,16] ya que la forma y el trazado de los mismos en una zona

urbana están en relación con las funciones que se desarrollan en ella, esto es, distinguir entre la función que tiene una zona residencial a la función que tiene una zona industrial.

Las funciones que predominan en cada zona de la ciudad influyen mucho en la morfología urbana.

2.1.3 Tipología de planos urbanos

El plano urbano, como se vio en puntos anteriores, es solamente el mapa de una localidad expresado a escala grande, el trazado de las calles permite distinguir diferentes formas geométricas las cuales suelen ser representadas por diferentes épocas, ya que la manera de organizar el espacio urbano ha cambiado a lo largo del tiempo según las necesidades, los gustos, los medios de transporte, etc. [12]

Algo que se debe tener en cuenta es que en la mayoría de las grandes ciudades actuales lo que se da es una acumulación de diferentes tipologías de planos [10], esto es, según las zonas de la ciudad es como va a depender su época de construcción y la función que predomine en ellas.

2.1.3.1 Plano desordenado o irregular

Como su nombre lo dice, un plano irregular es aquel en el que las calles no tienen un orden fijo, por lo general suelen ser en forma de curva y estrechas creando la sensación de estar en un laberinto cuando se pasa por ellas [10].

La causa de esto es que no se siguió una planificación previa, esto es, las casas y los edificios se construyeron sin seguir un orden. Todo esto da como resultado un trazado irregular. En la actualidad se puede observar este tipo de planos en la mayor parte de los centros históricos de las ciudades [16].

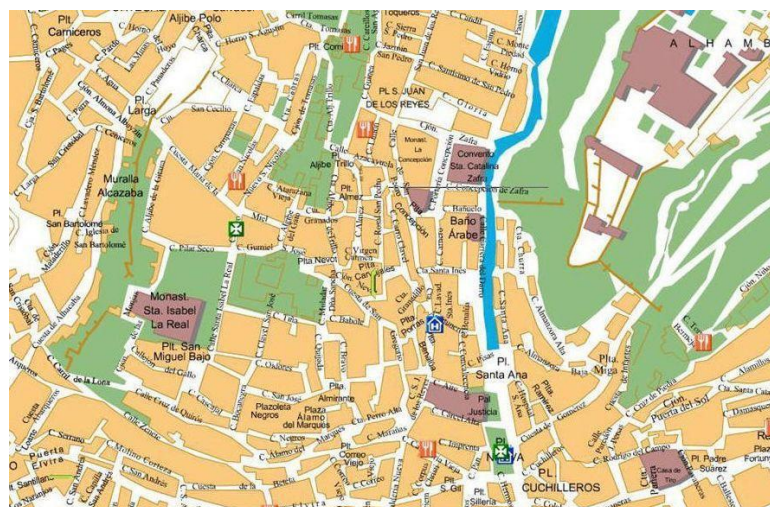


Figura 9. Plano irregular en el casco histórico de Granada. Barrio del Albaicín [10].

2.1.3.2 Plano ortogonal

Ortogonal es una palabra muy técnica para definir un tipo de plano urbano puesto que también significa reticular, en cuadrícula o en damero [15]. Ahora bien, un plano ortogonal es aquel en donde predominan las líneas rectas en el trazado de las calles, cortándose perpendicularmente y formando cuadrículas, creando la sensación de un orden [10].



Figura 10. Trazado ortogonal en el ensanche de Barcelona, área superior izquierda [15].

Este tipo de planificación se remonta a la época de los antiguos griegos, pero en la actualidad es muy usada cuando de crecimiento de ciudades se trata. Rodolfo Castagna [11] dice que este tipo de planos no es el más conveniente ya que dificulta la circulación en la ciudad, aumentando el tráfico vehicular debido a la necesidad de organizar los cruces con los semáforos.

2.1.3.3 Plano radiocéntrico

Este tipo de plano está organizado en torno a un punto central de la ciudad [17], un ejemplo sería una plaza y en base a ésta parten las calles principales en forma de radios.

El plano radiocéntrico suele darse en ciudades medievales ya que las calles giran en torno a un centro importante [17] como lo es un castillo, un monasterio, una iglesia, un mercado, etc. En la actualidad es difícil encontrar este tipo de planos debido a que la forma irregular predomina en estos tiempos.

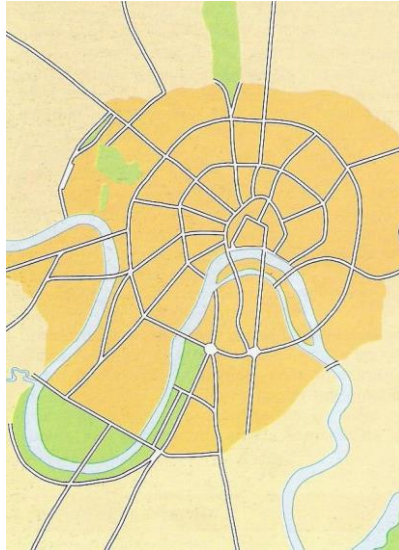


Figura 11. Plano de Moscú [17].

2.1.4 Categorías del suelo

El espacio urbano está formado siempre por la transformación del espacio rural. A este proceso se le denomina urbanización, formando de esta manera dos categorías de suelo, la primera se refiere a los espacios públicos, estos se caracterizan por la continuidad, es decir, son la parte vacía del espacio urbano.

Como segunda categoría se encuentran los espacios parcelados, la característica de éstos es la compartimentación, esto es lo lleno de los espacios urbanos [10].

2.1.5 Asentamientos urbanos

Los asentamientos urbanos se pueden clasificar en dos grandes grupos: el primero son los simples, los cuales constan de ciudades compuestas por su centro y sus barrios, además de incluir un área suburbana, estos asentamientos están separados de su entorno rural.

Como segunda clasificación se encuentran los asentamientos compuestos, los cuales son aglomeraciones urbanas formadas por estructuras simples [17].

2.2 Subdivisión iterada

Dentro de la bibliografía investigada, se encontró una forma de poder representar y crear planos urbanos de un modo que resultó sencillo de entender e implementar para los autores de este documento. Este método es el algoritmo llamado Subdivisión Iterada.

Este algoritmo comienza tomando un área inicial generalmente grande, la cual es dividida por una línea escogida según las restricciones que especifique la aplicación. El

proceso se va repitiendo en cada nueva área resultante de la división anterior hasta que se alcanza a obtener polígonos que cumplan con los tamaños especificados por el usuario o ya no sea posible hacer más divisiones. Es posible manipular el resultado de la red final mediante el uso de datos de entrada, como por ejemplo si se usan mapas de densidad poblacional [7].

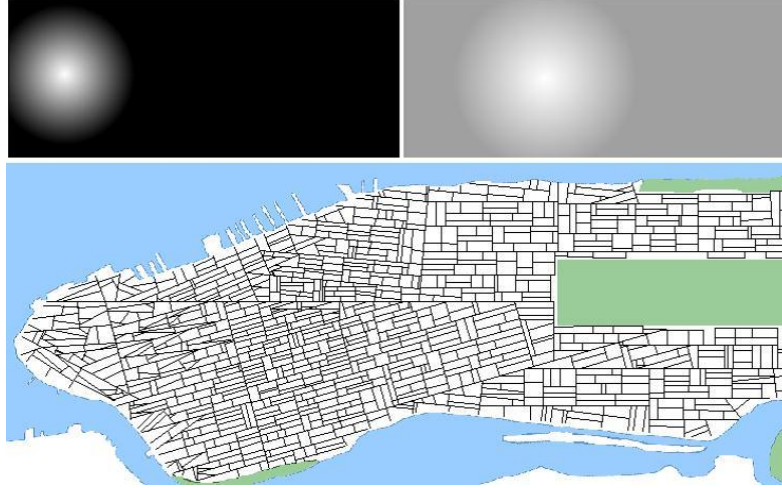


Figura 12. Posible resultado de la implementación de el algoritmo de subdivisión [7].

La relativa sencillez que presenta implementar la división de polígonos por medio de este método es la razón por la cuál se decidió implementar la idea de éste algoritmo.

El algoritmo del cuál está basando el proyecto es el siguiente:

Algorithm 1: Iterated Subdivision

Input: Polygon P , A_{min} , A_{max}
Output: S_{allot} , S_{roads}

```

1  $S_{oversized} \leftarrow P$ 
2  $S_{roads} \leftarrow \emptyset$ 
3  $S_{allot} \leftarrow \emptyset$ 
4 repeat
5    $P_{working} \leftarrow S_{oversized}.pop()$ 
6    $L_{bisect} \leftarrow P_{working}.getAcceptableBisector()$ 
7    $\{p_1, p_2\} \leftarrow P_{working}.bisect(L_{bisect})$ 
8    $S_{roads} \leftarrow S_{roads} \cup L_{bisect}$ 
9   if  $p_1.area > A_{max}$  then
10     $S_{oversized} \leftarrow S_{oversized} \cup p_1$ 
11  else
12     $S_{allot} \leftarrow S_{allot} \cup p_1$ 
13  end
14  if  $p_2.area > A_{max}$  then
15     $S_{oversized} \leftarrow S_{oversized} \cup p_2$ 
16  else
17     $S_{allot} \leftarrow S_{allot} \cup p_2$ 
18  end
19 until  $|S_{oversized}| == 0$ 

```

Figura 13. Algoritmo de subdivisión iterada [7].

La siguiente tabla especifica el significado de cada término del algoritmo:

P	Representa un polígono, en el caso del proyecto un polígono es definido mediante 4 vértices.
S_{allot}	Lista de polígonos que tienen un área aceptable.
S_{roads}	Lista de líneas que representan las calles, las cuales dividen a los polígonos.
S_{oversized}	Lista de polígonos que se pasan de los límites de áreas y que aún se siguen dividiendo.
A_{min}, A_{max}	Son los límites de las áreas que pueden tener los polígonos.
P_{working}	Es el polígono sobre el cuál se hacen las operaciones en la iteración actual.
L_{bisect}	Línea que dividió al polígono correctamente, ésta se agrega a S _{roads} .
P₁, P₂	Son los polígonos que resultan de la bisección de P _{working} por medio de bisect(). Los que cumplan con los límites de las áreas se agregan a S _{allot} y los que no se agregan a S _{oversized} .
Pop ()	Hace referencia a que se pasa a P _{working} el último elemento que se agregó a S _{oversized} y después es borrado de la lista.
getAcceptable Bisector()	Hace referencia a que se hacen los cálculos necesarios sobre P _{working} para obtener una línea L _{bisect} que divida al polígono de forma aceptable. Para decir que sí es aceptable se debe obtener la proporción que hay entre el <i>diámetro</i> y el <i>ancho</i> del polígono (debe ser convexo), en donde si la proporción es muy grande el polígono resulta muy alargado y por lo tanto inaceptable. Entre más se acerque la proporción a 1 el polígono se asemejará mas a un cuadrado y por lo tanto es más aceptable.

Tabla 1. Descripción de los conceptos básicos de la figura 13.

Capítulo 3. Materiales y Métodos

Este capítulo contendrá todos y cada uno de los pasos del desarrollo plasmando ideas, esfuerzos, dificultades y logros con los que se enfrentó al momento de realizar el proyecto.

3.1 Descripción del área de estudio

Este proyecto es un desarrollo tecnológico ya que se está experimentando con tecnología prácticamente reciente, la cual tiene poco más de 4 años en ser creada a partir del momento en que se empezó a escribir este documento. El proyecto está basado en distintos algoritmos que darán como resultado la creación de la ciudad virtual en 3D.

3.2 Materiales

Herramientas	Características
Laptop	Con procesador: Athlon X2 QL62 de 2.00 GHz. Memoria RAM: 4.00 GB. Tarjeta de video: NVIDIA GeForce 9600M GT.
OpenGL 2.1	Interface y controlador encargado de la visualización del proyecto.
OpenTK	Librería que permite la implementación de OpenGL en Visual C#.
Visual C# 2010.	A criterio de los autores de este documento se decidió este ambiente de desarrollo puesto que es con el que más familiaridad se tiene, lo cual permite agilizar el desarrollo de la aplicación.
Algoritmo <i>Iterated Subdivision</i> (ItSub)	Algoritmo principal en el que está basado el proyecto.
Fotografías reales de ciudades	Modo de comparación de resultados obtenidos de Google Maps.

Tabla 2. Descripción de las herramientas.

3.3 Métodos

En este punto se comienza a plasmar todo el desarrollo del proyecto, iniciando con la metodología y finalizando con los procedimientos.

3.3.1 Metodología

Como primer punto de la metodología que se siguió, fue necesario comenzar especificando cuáles de los métodos investigados iba a ser posible implementar para lograr obtener los puntos planteados en la sección 1.6, los cuales se presentan a continuación:

Mapa de calles

Para obtener el mapa de calles, que finalmente es la estructura principal que representa a la ciudad, se decidió utilizar el algoritmo de ItSub propuesto en el método de Rudzicz & Verbrugge [7]. La principal razón fue la sencillez y practicidad de entendimiento. Los resultados que se observaron en el documento parecieron bastante aceptables para lo que éste proyecto quería alcanzar.

Lotes

Para lograr obtener los lotes, los cuales representan el área donde habrá construcciones, se observó que era posible seguir utilizando el mismo método de subdivisión que ya se utilizaba para las calles, puesto que el problema para obtenerlos resultaba muy similar, la diferencia principal radicó en que la subdivisión para generar los lotes debía ser más especializada.

Construcciones

Para realizar las construcciones en un inicio se optó por utilizar la forma en que se hacen las edificios en el método de Greuter et al [3], puesto que el concepto de su implementación resulta sencillo de entender. Se intentó tomar la misma idea pero sólo fue posible implementar una forma no muy compleja de obtenerlas.

Como segundo punto, ya una vez que se identificaron los métodos a utilizar, se tuvo que especificar cuáles herramientas harían posible la implementación y la verificación de resultados para el proyecto.

En tercer punto, para llevar a cabo la visualización de la aplicación se optó por utilizar las librerías de OpenGL, la razón principal fue que se tuvo gran facilidad para encontrar la documentación necesaria para poder utilizarlas. Del mismo modo, al inicio del proyecto se identificó la librería de OpenTK, que permite implementar OpenGL dentro de Visual C#, lo cual ultimadamente fue lo que condujo a la decisión de usar éste último como herramienta de programación, aparte de que es el ambiente de programación con el que más familiaridad se tiene.

Como último punto se utilizaron fotografías aéreas de secciones de ciudades reales para poder comparar cualitativamente los resultados obtenidos de la aplicación. Las fotografías se seleccionaron conforme a las delimitaciones del proyecto especificadas en la sección 1.6.

3.3.2 Procedimientos

En esta sección se presentarán los procedimientos que se siguieron a lo largo del proyecto para que éste se llevara a cabo.

3.3.2.1 Requisitos del sistema

Primeramente, para lograr que funcione el programa el usuario debe cerciorarse de contar con una tarjeta gráfica que soporte al menos OpenGL 2.1. En caso de no contar con Visual Studio 2010 instalado, se debe instalar el paquete redistribuible *.Net Framework 4.0*.

3.3.2.2 Desarrollo de la lógica del proyecto

Para facilitar el flujo de la redacción, las secciones del código a las que se haga referencia en esta sección serán partes del código fuente perteneciente a la aplicación de este proyecto, éstas estarán disponibles en la sección de apéndices.

Avenidas y colonias

De los métodos para generar mapas de ciudades que se discutieron en la sección 1.1, se había identificado como principal candidato a implementar el presentado por Esch et al [6], mediante el uso de campos tensoriales. La principal razón de ello fue que se observaba que obtenía resultados con una complejidad bastante aceptable cuando se comparaban con fotografías aéreas de ciudades, aparte de que daba la posibilidad de modificar el mapa ya una vez generado.

Debido a la falta de tiempo necesario para lograr entenderlo e implementarlo, se tuvo que optar por buscar alguna otra forma de obtener el mapa de calles. Esto nos llevó a contemplar el algoritmo de ItSub para su implementación. A pesar de que por la naturaleza del método se obtenían resultados menos complejos que en el anteriormente mencionado, su implementación resultaba mucho más práctica y entendible. Así que se decidió utilizarlo.

Así pues, cada avenida es representada por una línea, la cual a su vez es definida por dos puntos (x, y): el inicial y el final. Para la mayoría de los cálculos en donde es necesario obtener divisiones, intersecciones, áreas y demás, se utilizaron las ecuaciones de las rectas que representan a cada avenida involucrada, esto junto con el conjunto de

fórmulas ya definidas para obtener todos esos cálculos (véase apéndice 1). Cabe mencionar que a las rectas se les permite cierto grado de inclinación, esto para permitir más irregularidad a las avenidas.

En un primer intento en donde se crearon las avenidas con ItSub, efectivamente se consiguieron resultados al ir subdividiendo un polígono inicial, de modo que al final resultara una estructura parcelada que de hecho asemeja una ciudad. Puesto que lo principal a considerar en el algoritmo es la forma en que se hace la línea que divide al polígono, el resultado de cada subdivisión depende totalmente de ello. Más adelante en esta sección se describirá en mayor detalle este proceso.

Fue entonces que se observó que aunque se obtenía una red de calles algo convincente, la aleatoriedad con la que se creaba la línea divisoria hacía también que las calles fueran totalmente al azar, perdiendo en ciertas ocasiones un sentido general de dirección, es decir que las calles no llegaban a dar a ningún lado. El problema se acentúa más cuando se crean las calles o avenidas principales, ya que éstas podían resultar muy cortas, de modo que no atravesaban la mayor parte de la ciudad. En la realidad, una calle o avenida es principal precisamente porque conecta con una gran proporción de la ciudad, a diferencia de las calles secundarias.

Para solucionar el problema, se decidió hacer una división inicial para obtener las avenidas, asegurando que estas tuvieran cierta regularidad en cuanto a siempre cubrir todo el largo y ancho de la ciudad.

El procedimiento para lograrlo fue crear avenidas a lo largo de un eje vertical y un eje horizontal, cuántas avenidas se generarían a lo largo, y cuántas a lo ancho del área que representa la ciudad (véase el Apéndice 2): *NCallesHor* y *NCallesVer* será el número de calles horizontales y verticales, respectivamente. Luego fue necesario especificar qué tanta separación habría entre cada avenida siguiente, *TmpSeparaciones[0]* representa la separación entre cada avenida creada a partir de la horizontal, de igual modo *TmpSeparaciones[1]* a partir de la vertical.

Se utilizó un acumulador para la separación, *TmpAvance*, para poder ir moviendo horizontal o verticalmente cada nueva avenida que se crea, conservando cierto grado de regularidad entre cada separación. Luego en el Apéndice 3 se define para cada avenida horizontal su ecuación, como la pendiente es a fin de cuentas la inclinación que la recta tendrá, se le dio directamente un valor aleatorio. Teniendo ya el valor en *pendiente*, se calcula la ordenada al origen en *b_nxt* para completar la ecuación.

Después, se procede a calcular los puntos inicial y final de la recta. Fue en este paso donde surgió un problema relacionado a cuando había intersecciones entre rectas horizontales o intersecciones entre rectas verticales (figura 14), de forma que los polígonos que se formaban a partir de las intersecciones podían contener un número de vértices diferente a cuatro.

Puesto que era necesario, una vez obtenidas las avenidas, calcular qué vértices definían a cada polígono a partir de las intersecciones, el cálculo se tornaba muy complejo al momento de definir qué vértices pertenecían a qué polígono.

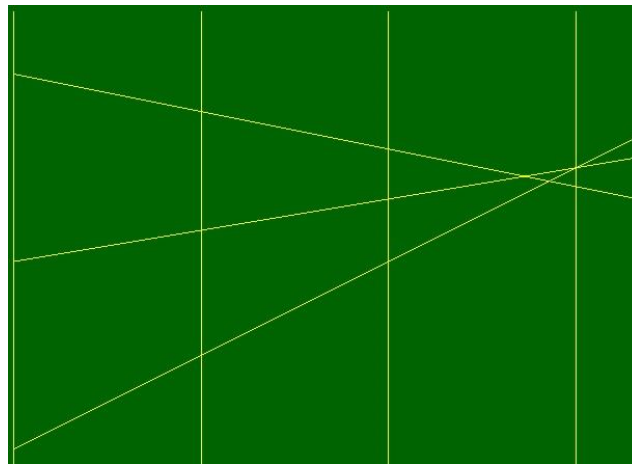


Figura 14. Intersecciones solamente entre horizontales.

De modo que hubo la necesidad de establecer que los polígonos generados sólo fuesen de cuatro vértices, de esta forma, sólo comparando las intersecciones de cada avenida horizontal con las verticales se podían organizar los vértices en una estructura de datos que facilitara la detección de qué vértices pertenecen a qué polígono.

En la tabla 3, se ilustra la forma en que los vértices se acomodan y se facilita dicha detección, las celdas interiores representan intersecciones entre horizontales y verticales. Cada H representa las rectas creadas a lo largo de una horizontal y cada V representa las rectas creadas a lo largo de una vertical:

	V0	V1	V2	V3
H0	0,0	0,1	0,2	0,3
H1	1,0	1,1	1,2	1,3
H2	2,0	2,1	2,2	2,3

Tabla 3. Ejemplo de intersecciones.

En una estructura donde sólo pueden haber cuadriláteros sin intersecciones de verticales con verticales ni horizontales con horizontales, se puede deducir que un

polígono se forma por (0,0), (0,1), (1,0), (1,1). El siguiente se forma por (0,1), (0,2), (1,1), (1,2). Otro, por ejemplo, se forma por (1,2), (1,3), (2,2), (2,3).

La definición de los polígonos se consigue entonces asegurando que no haya el tipo de intersecciones complejas como en la figura 14, aparte de que estén acomodadas en una estructura de modo que cada recta siguiente sea la vecina inmediata. Se utilizó *QuickSort* para realizar este acomodo, véase el apéndice 4. Luego al hacer pasadas en la estructura de datos, entre intersecciones de dos horizontales y dos verticales vecinas se consiguen los puntos que definirán al polígono. En este proyecto se denominarán como Colonias a estos polígonos.

El proceso para calcular y obtener las rectas, tomando en cuenta todas las consideraciones anteriores, está implementado en el Apéndice 5. Para las calles que se crean a lo largo de la horizontal izquierda y de un modo similar para las que se crean a lo largo de la vertical inferior. Se utilizó el algoritmo de *Quicksort* para estructurar y acomodar los puntos de las intersecciones y así poder obtener las Colonias, se puede observar en el Apéndice 6.

Una vez obtenidas las colonias se tuvo que pensar en cómo hacerle para que su área no se extendiera hasta la línea que representa a cada avenida, dándole de esta forma cierta distancia del área de la colonia a la línea de la avenida se le permite a ésta crear la ilusión de ser una de las calles principales. Al momento de obtener la colonia, tomando en cuenta esta distancia, solamente cambiando su color se logra distinguir y le da a la avenida la apariencia que se mencionó anteriormente.

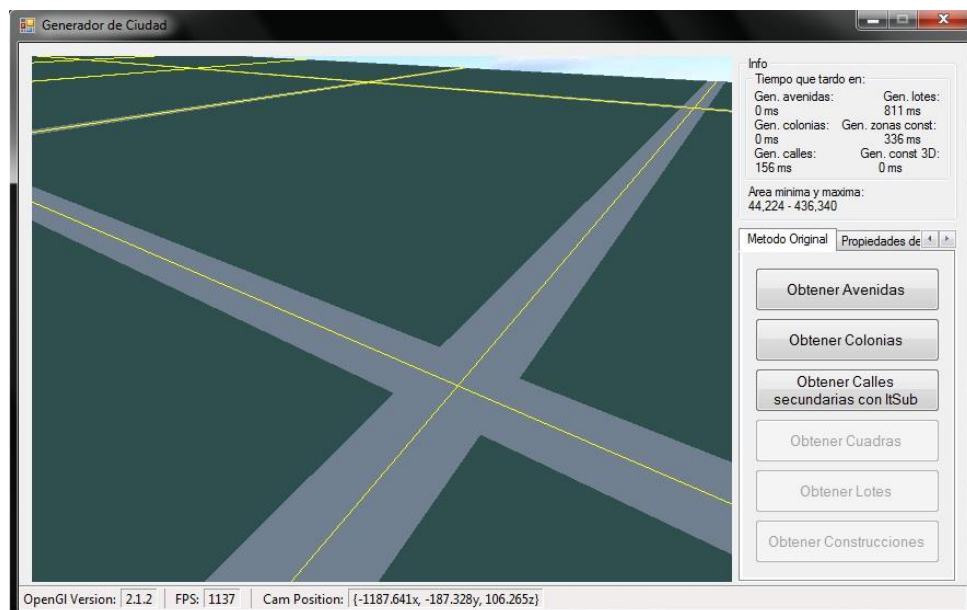


Figura 15. Ancho de las avenidas

Para llevarlo a cabo se tomaron en cuenta los cuatro vértices de las intersecciones de las avenidas que conforman a la colonia, a cada uno de estos puntos de les sumó o restó una distancia para obtener los resultados mostrados en la Figura 15. La implementación de esto está en el Apéndice 6, en la parte de *ObtenerAreasDeColonias()*.

El método que se utilizó es incorrecto cuando las avenidas presentan alguna inclinación, puesto que no se toma en cuenta ésta cuando se suma o resta la distancia a los vértices de la colonia.

Ya que se aseguró que las avenidas se pudieran crear con cierta regularidad en cuanto a su forma, se puede entonces aplicar el algoritmo ItSub a cada colonia resultante para obtener las calles secundarias. La aleatoriedad con la que se habló al principio sobre el resultado de las calles que se obtienen con ItSub es en este caso una ventaja, pues es lo que comúnmente se observa en la realidad.

Calles secundarias y cuadras

El punto principal del proceso de ItSub es, como ya se mencionó antes, la creación de la línea que subdividirá a un polígono, de modo que el proyecto tiene un fuerte enfoque en ello. El algoritmo se utiliza iterativamente en cada colonia, utilizando la función el Apéndice 8, para ir haciendo cada división y subdivisión, tomando en cuenta *Amin*, que delimita el área mínima de una colonia, y *Rmax* que delimita la proporción máxima entre diámetro y ancho que puede tener el polígono

Ya una vez llegado a este punto se hace la división llamando a *getBisectorLine()* (Apéndice 9), empezando en una línea inicial y terminando en el extremo no adyacente a ésta, de esta forma se asegura que siempre resulten cuadriláteros de las subdivisiones. El punto inicial de la línea divisoria se decide al partir la línea inicial en dos o tres partes aleatoriamente: si se divide en dos solo puede elegirse el punto medio, si se divide en tres puede elegirse la primera o segunda o tercera parte, y finalmente se guarda ese punto en $(X1, Y1)$, el punto final se obtiene de una forma similar pero se utiliza el extremo no adyacente anteriormente mencionado y se guarda en $(X2, Y2)$.

Aquí mismo puede decidirse (con *boIrregular*) si el punto inicial es correspondiente al punto final, esto es, si el punto inicial se encuentra por ejemplo en la primera o tercera parte de la línea inicial, el punto final se encontrará también en la primera o tercera parte del extremo no adyacente; ó si no son correspondientes, esto es, el punto

inicial puede encontrarse por ejemplo en la mitad de la línea inicial y el punto final puede encontrarse en la segunda tercer parte del extremo no adyacente.

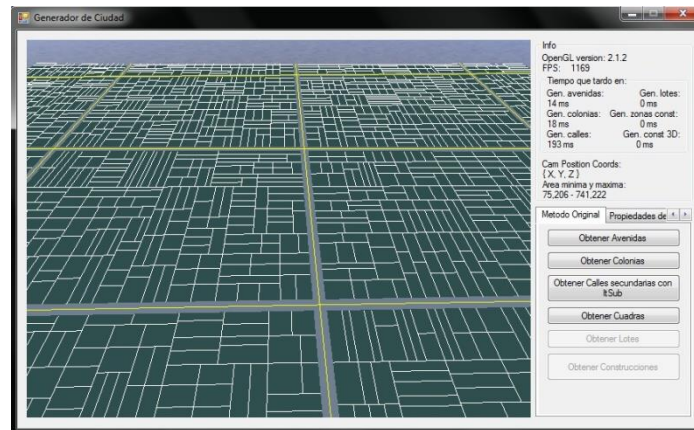


Figura 16. Calles secundarias cuando los puntos inicial y final son correspondientes.

Cuando se opta por hacer las calles como en la figura 16 la aplicación presenta mas dificultades al tratar de crear las calles secundarias, en algunas ocasiones se pueden presentar resultados no muy convincentes puesto que éstos repercuten a la forma que toman las edificaciones.

Ya obtenida la línea divisoria, se retorna a *getAcceptableBisector(AreaMin, Rmax, contador, bIrregular)*, en el Apéndice 8, y es entonces guardada en *BisectorLine*, ésta es la que se utiliza para definir los vertices que conformarán a los dos polígonos que resultan al hacer la división con esa línea, estos se guardan en *PoligonosBisectados[0]* y *PoligonosBisectados[1]*. Si el área de los dos polígonos resultantes es mayor o igual al *AreaMin* entonces la línea divisoria es aceptable y se guarda en *PoligonosBisectados[2]*, en caso contrario el proceso de obtener la línea divisoria se vuelve a repetir hasta obtener una aceptable.

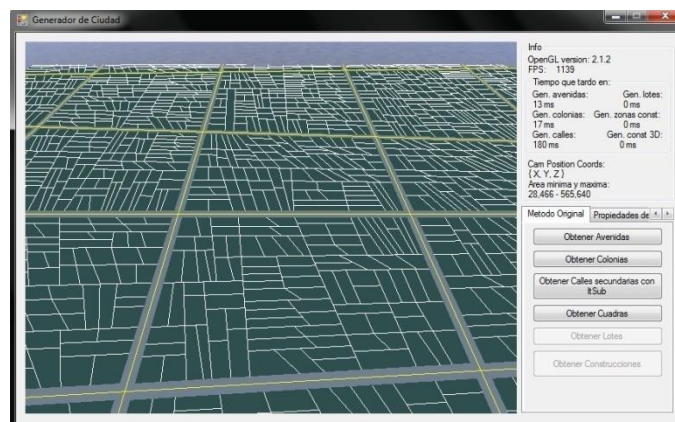


Figura 17. Calles secundarias cuando los puntos inicial y final no son correspondientes.

Finalmente, *PoligonosBisectados[]* es retornado a y se guarda en *DatosBiseccion* (véase Apéndice 7), después se calcula la proporción diámetro:ancho del polígono, se guarda en *Ratio* y es éste el que se utiliza para comparar que no sea mayor que *Rmax*, si no lo es entonces se procede a guardar la línea divisoria en *ListaShapeRoads* ya que ésta es una calle, luego se verifica que las áreas de los polígonos resultantes de la división no sobrepasen a *Amax*, en cuyo caso aquel que no sobrepase este valor es guardado en *ListaShapeAllots*, de lo contrario se guarda en *ListaShapePOversized* y se repite todo este proceso hasta que *ListaShapePOversized* esté vacío.

Ya una vez que está vacío significa que ya se obtuvieron tanto las calles como las cuadras, éstas últimas se obtienen sencillamente recorriendo *ListaShapeAllots*. Este proceso se observa en el Apéndice 10.

Pueden presentarse ocasiones en las que la colonia no logró ser totalmente subdividida por algún error en la aplicación del algoritmo, en donde no fue posible obtener cuadras en una o varias zonas de la colonia, en este caso se clasifican como inválidas y se visualiza en color negro.

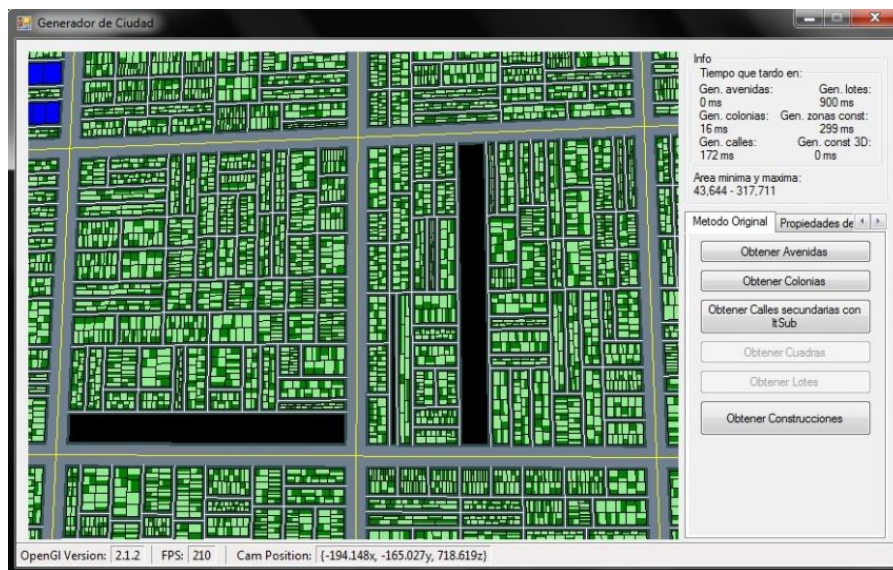


Figura 18. Zonas inválidas en 2 de las colonias.

De un mismo modo que en las avenidas, se tomó en cuenta una distancia para lograr esta vez darle a las calles secundarias un ancho realista, presentando las mismas dificultades (figura 19).

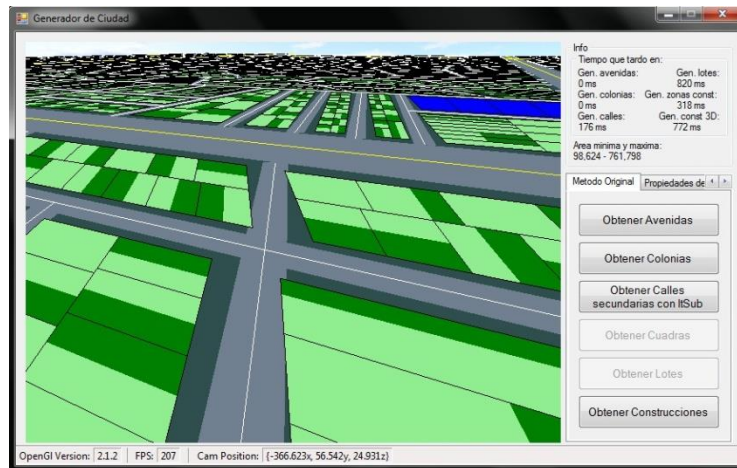


Figura 19. Se observa un desfase en las cuadras cuando alguno de sus lados presenta una inclinación.

Cabe mencionar que a diferencia de la forma en que se obtuvieron las avenidas, el aplicar ItSub permite obtener de una sola vez las líneas que representarán calles y los polígonos que definirán en este caso a las cuadras, evitando la complejidad de calcular intersecciones entre rectas y realizar el acomodo de las calles mencionado en las avenidas.

Lotes

Puesto que la generación de lotes sigue siendo un problema de división de polígonos, se observa inmediatamente que es posible seguir utilizando el algoritmo de ItSub, solo que esta vez un poco más especializado.

Una consideración que se tomó en cuenta es que los lotes, puesto que representarán zonas de construcción, siempre deben dar a una calle para evitar tener lotes encerrados. Otra consideración, para asegurar que los lotes siempre tuvieran construcciones que dieran a uno y otro lado de las calles adyacentes al lote, es que las divisiones se decidieron hacer siempre empezando y terminando en los lados más largos de cada polígono. Estas consideraciones pueden ser mejor observadas en la figura 20.

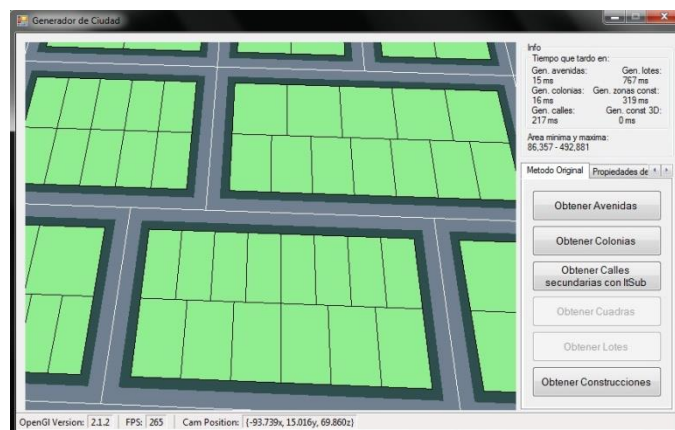


Figura 20. Division de cuadras en lotes.

Para lograr la implementación, primero fue necesario especificar el tipo de edificios que se construirán en cada lote, en el Apéndice 11 se observa la implementación.

Para hacerlo se decidió clasificar las cuadras en zonas residenciales o zonas comerciales, en donde las residenciales contienen construcciones más pequeñas que las comerciales. Se tomó en cuenta el área total de la cuadra, si ésta cae sobre un rango menor o igual a *A_{max}* y mayor que una cantidad especificada se considerará como zona comercial, de lo contrario, como es menor será residencial.

Una vez clasificada cada cuadra se realiza de nuevo el proceso de dividir los polígonos de una manera similar a las calles secundarias (similar al Apéndice 8). En esta función especializada, para las zonas residenciales, se hará una primera división a lo largo para obtener lotes de un lado y de otro de la calle, todas las demás divisiones subsiguientes se harán a lo ancho, cubriendo con esto las consideraciones de las que se habló al inicio. El parámetro *n_{lotes}* indica, por separado, cuantos lotes se desean obtener en cada lado de la primera división.

Para las zonas comerciales, puesto que contendrán construcciones más grandes, se omitió la primera división que se hizo en las zonas residenciales y utilizando sólo las divisiones subsiguientes, esto para asegurar construcciones más grandes y hacer la diferencia entre casas y edificios más obvia. Se sigue utilizando *n_{lotes}* para indicar cuantos lotes comerciales habrá en cada cuadra.

Ya que se definieron los lotes de cada zona, se decidió tomar en cuenta que en las zonas residenciales puede haber lotes en donde una construcción sólo ocupa una parte del lote y no toda el área, tratando de simular de esta manera que en algunas residencias hay espacios libres, como por ejemplo patios. En los lotes para los edificios se omite esto y se utiliza toda el área del lote para construir, obteniendo así edificios más grandes.

Para lograrlo simplemente se volvió a utilizar el algoritmo de ItSub, sólo que esta vez a nivel lote. El algoritmo hace una subdivisión en donde el polígono que se dividirá se puede partir en dos o tres partes, luego aleatoriamente se decide cuál de las dos divisiones será zona libre y cuál zona de construcción, de este modo, por ejemplo, si se partió en 3 partes la zona de construcción puede medir dos tercios y la zona libre un tercio o viceversa. También se tomó en cuenta que se presentan ocasiones en las que hay lotes sin construir, en cuyas ocasiones se omite hacer la subdivisión.

En este punto se intentó darle un espacio a las zonas de construcción de los lotes comerciales para simular que pudiese haber un tipo de callejones entre cada edificio dándole de esta manera más realismo a estos. Esta operación no se logró debido a que no se encontró un método sencillo para llevarlo a cabo.

Ya con todo esto se tienen las bases para poder empezar así a generar las construcciones como cuerpos en 3D.

Construcciones

Al inicio del proyecto se tenía planeado implementar el método de [3] para hacer los cuerpos sólidos de los edificios, en donde se le daba bastante variedad a la forma que podían tomar los cuerpos conforme se creaban. Para mayor detalle se puede observar la figura 2. Por cuestión de falta de tiempo al momento en que se llegó a esta etapa de la implementación no fue posible intentar aplicar este método.

La forma en que se decidió hacer la implementación fue otra más sencilla en la cual, teniendo ya la base de la construcción, se hizo una serie de extrusiones para lograr darle altura, de este modo se pueden hacer cuerpos que asemejan edificaciones con resultados más sencillos.

El método consiste en realizar extrusiones, esto es tomar la base que se obtuvo anteriormente y darle una altura a todos los vértices que la conforman, de este modo se consigue generar un piso de la construcción, ya dependiendo de si ésta tendrá más de un piso se repetirá este método tantas veces como pisos vaya a tener, siempre tomando en la siguiente iteración los vértices del techo anteriormente formado.

La forma en que se crearon las construcciones puede observarse en el Apéndice 12. De este modo se logró alcanzar lo mínimo esperado en el proyecto.

Capítulo 4. Resultados de la investigación

En esta sección se mostrarán y explicarán a detalle los resultados obtenidos en este proyecto, así como también se hará una comparación entre imágenes de ciudades reales y las que genera el proyecto.

4.1 Presentación de resultados

Avenidas y Colonias

En la figura 21 se puede observar el resultado final de las avenidas principales y de las colonias, notándose la regularidad a la que se refería en el procedimiento quedando tal vez demasiado cuadrangular.

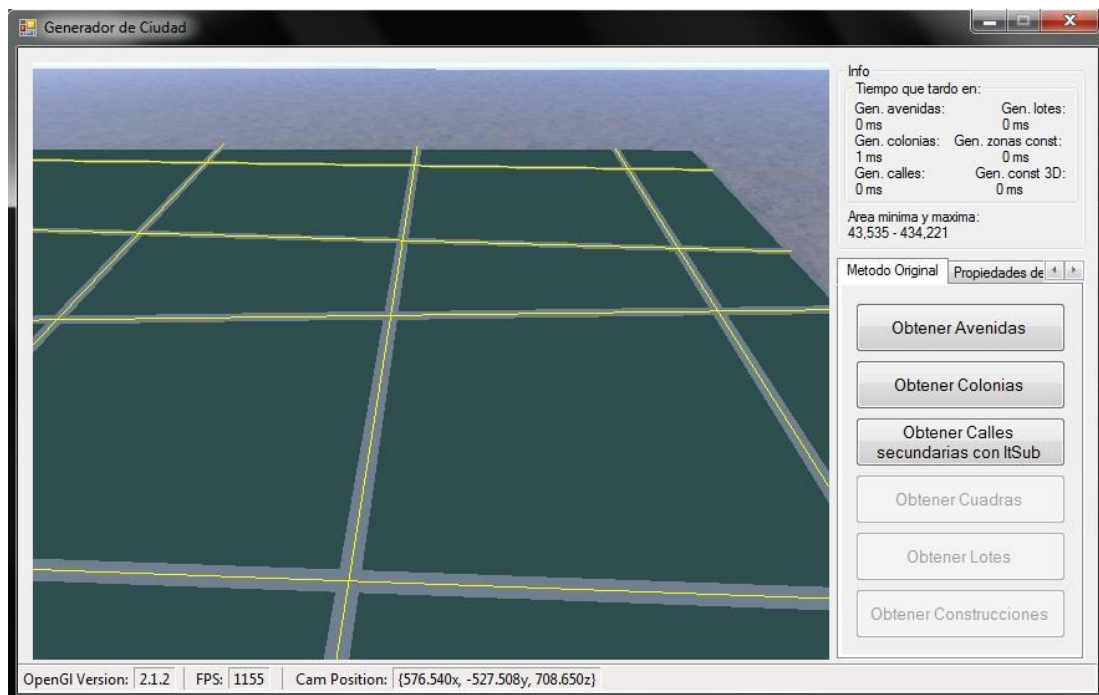


Figura 21. Resultado de las avenidas principales y las colonias.

Aunque esta forma de generarlas permitió lograr la implementación es bastante notable que necesite de más complejidad para alcanzar resultados más realistas a este nivel.

Calles secundarias y Cuadras

Por lo que se puede observar en esta fase de la generación resulta sorprendente, al menos para los autores de este proyecto, el nivel de realismo que se alcanzó con la implementación del algoritmo de ItSub, lo que le hacía falta a la generación de las avenidas es ofuscado cuando se llega a esta fase. Esto se observa con más detalle en la figura 22.

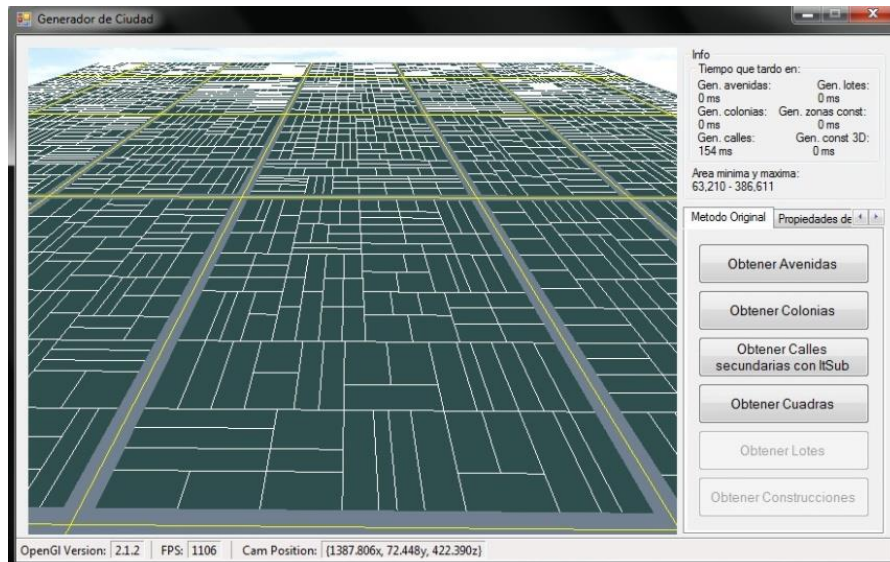


Figura 22. Resultado de calles secundarias y cuadras.

Otro resultado que se obtiene en este mismo proceso al cambiar el parámetro de regularidad en la aplicación se observa en la figura 23.

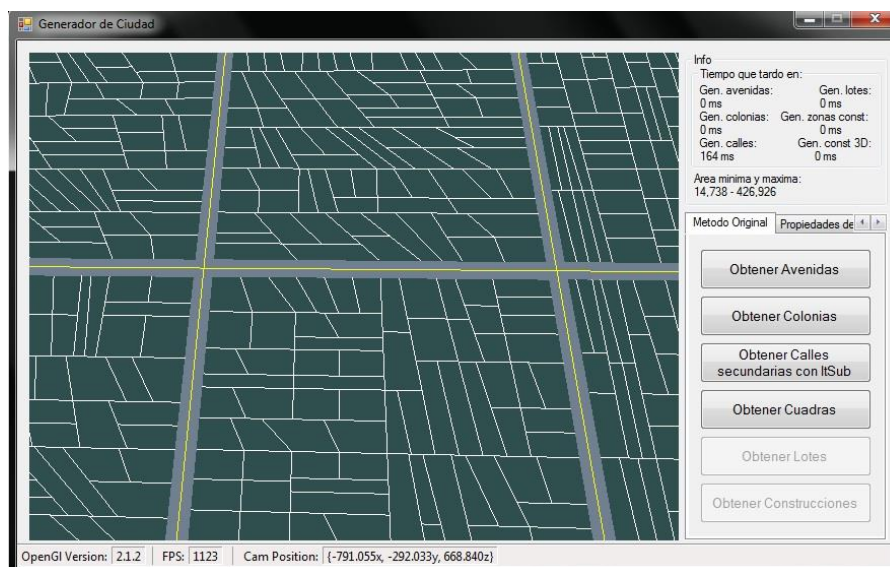


Figura 23. Resultado de calles secundarias y cuadras irregulares.

En este caso las irregularidades se presentan de una forma bastante descontrolada quitándole realismo en vez de aumentárselo, a diferencia de la figura 22.

En la figura 24 se observa por medio de colores la razón de utilizar R_{max} , en la sección 3.3, para tratar de evitar que las cuadras tuvieran proporciones no adecuadas. El color azul indica una razón mínima en donde la proporción del área y el largo es cercana a 1, los siguientes colores pasando por el amarillo, el naranja y el rojo indican que la proporción se aleja cada vez más de 1, indicando que son más largos que anchos.

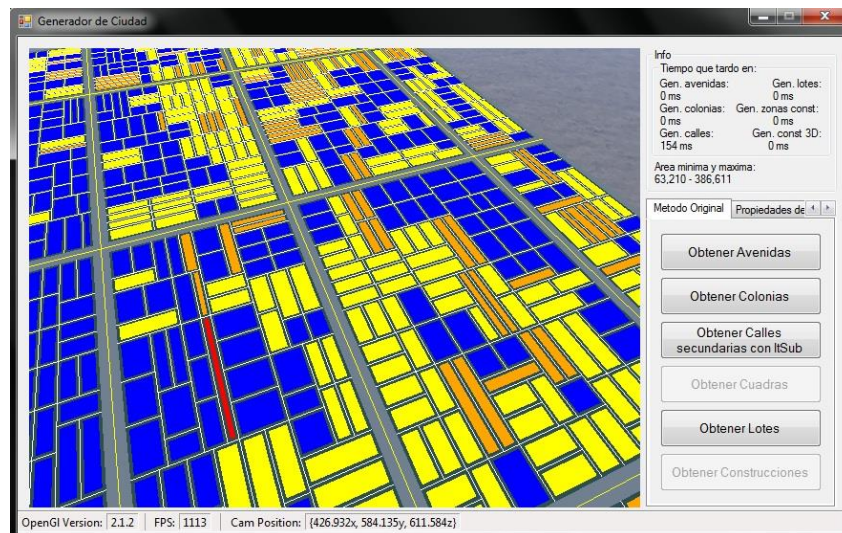


Figura 24. Cuadras coloreadas.

Lotes

En la figura 25 se observa lo que se obtiene al intentar resolver el problema de la división de lotes utilizando el acercamiento descrito en el procedimiento.

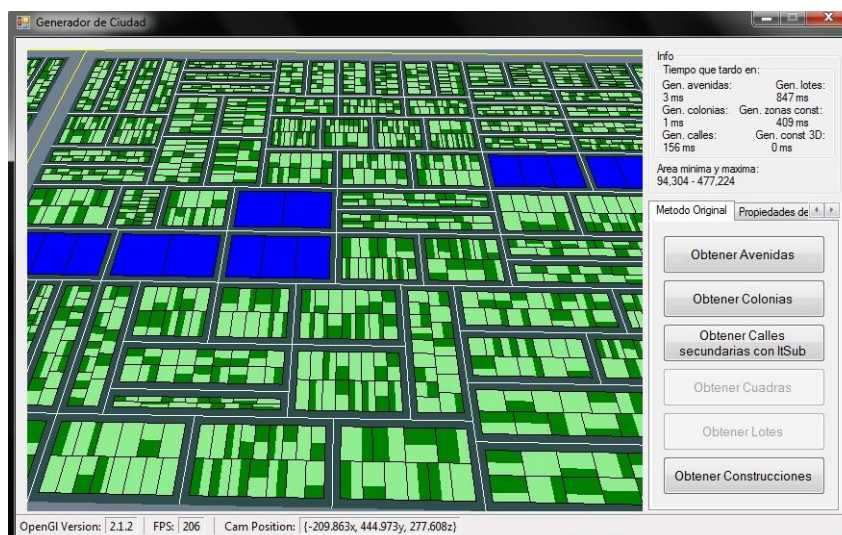


Figura 25. Resultado de los lotes.

Los lotes están delimitados por contornos negros, en este caso el color verde oscuro representa el área del lote en la que se construirá, el color verde claro representa el área libre de construcción de cada lote. Se puede observar que en algunos lotes no se construye nada y se quedan vacíos.

Las áreas azules representan las cuadras que fueron clasificadas como zonas comerciales, en las cuales se construirán edificios. Puede observarse que en estos casos se utiliza toda el área del lote para construir el edificio. En la figura 26 se observan como resultan los lotes en esta aplicación cuando las cuadras son irregulares.

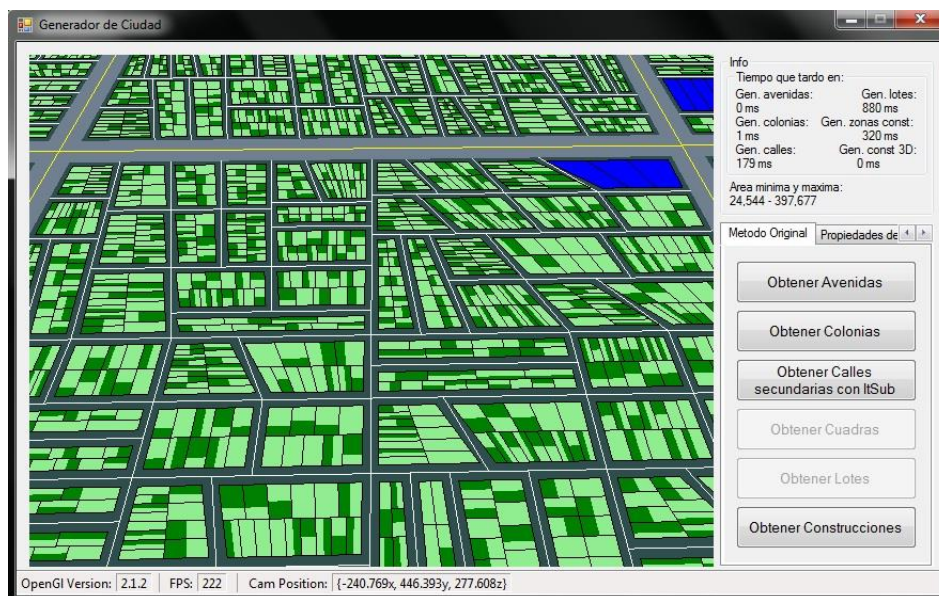


Figura 26. Cuadras irregulares.

Construcciones

En seguida se muestra en las figuras los resultados que se obtuvieron en la aplicación en cuanto a las construcciones.

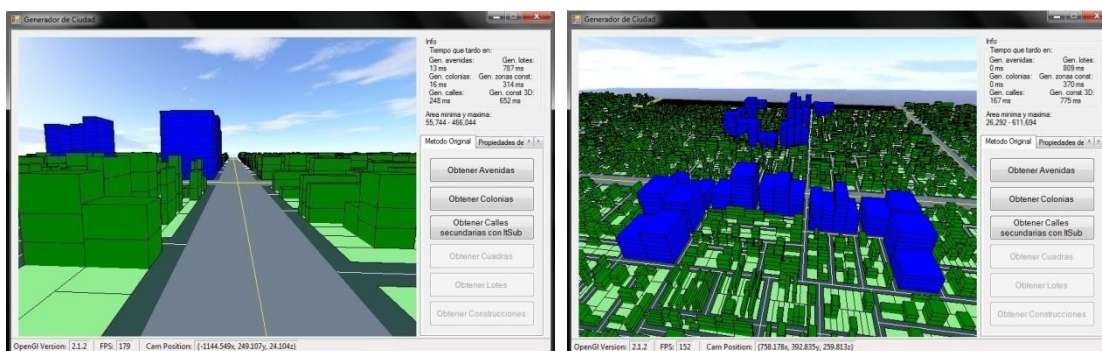


Figura 27. Vistas aéreas de las construcciones.

En ambas imágenes se alcanza a apreciar las separaciones entre las construcciones y sus diferencias en tamaños, se observa de igual modo que la morfología de la ciudad en cuanto a calles y construcciones es congruente.

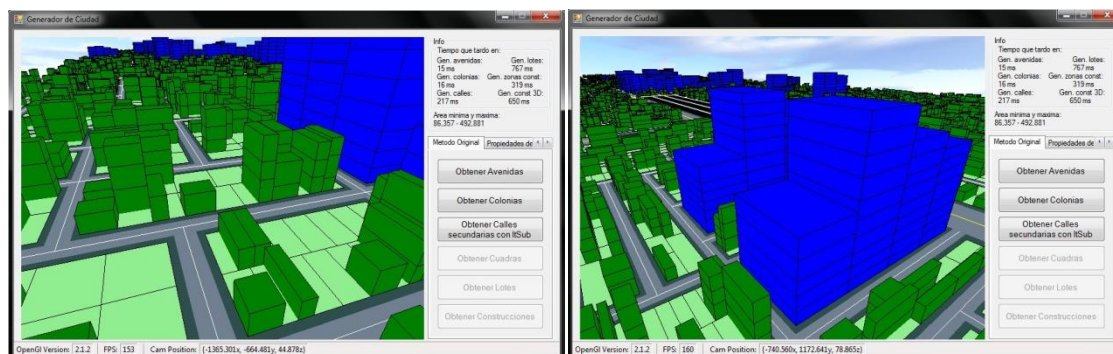


Figura 28. Resultados en cuanto a las zonas de construcción.

En las imágenes de la figura 28 se nota como afecta el hecho de tomar en cuenta si alguna zona de cada lote se va a dejar sin construir, en las zonas residenciales (a la izquierda, de color verde) se alcanza a observar que a parte de su construcción cuentan con un área libre que podría llegar a ser, a fin de cuentas, un patio o un frente.

En cambio en las zonas comerciales (a la derecha, de color azul) no se utilizó este espaciado, tratando de hacer que los edificios fueran más grandes.

4.2 Análisis e interpretación de resultados

Ahora se harán análisis cualitativos para comparar si los resultados obtenidos se acercan a la realidad, como fue planteado en el objetivo de la investigación. Iniciando con la imagen 29 donde se observa una parte de la ciudad de Nueva York que presenta características muy notorias de un plano ortogonal, éste es el que actualmente la aplicación genera con mayor precisión, pues en cierta forma es el más sencillo de obtener.

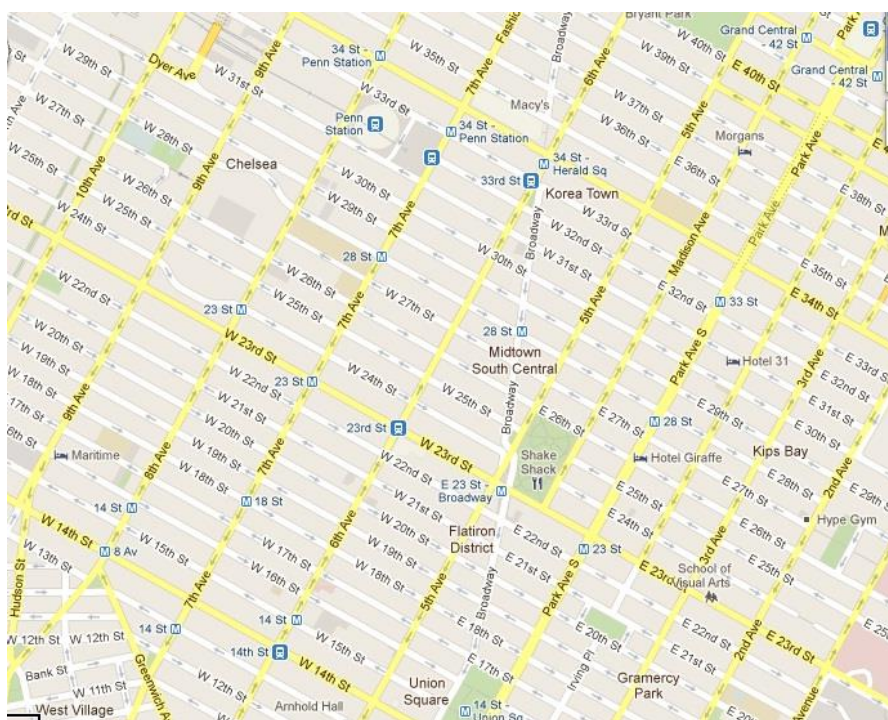


Figura 29. Avenidas principales y colonias de Nueva York. [18]

La siguiente imagen muestra características muy similares en cuanto al patrón que pueden tomar las avenidas y colonias comparándolas con la figura 29. En la imagen izquierda de la figura 30, se observa que las avenidas cuentan con más regularidad, pero en cambio también es posible alejarse un poco de la misma sin perder demasiada ortogonalidad, como puede verse en la imagen de la derecha.

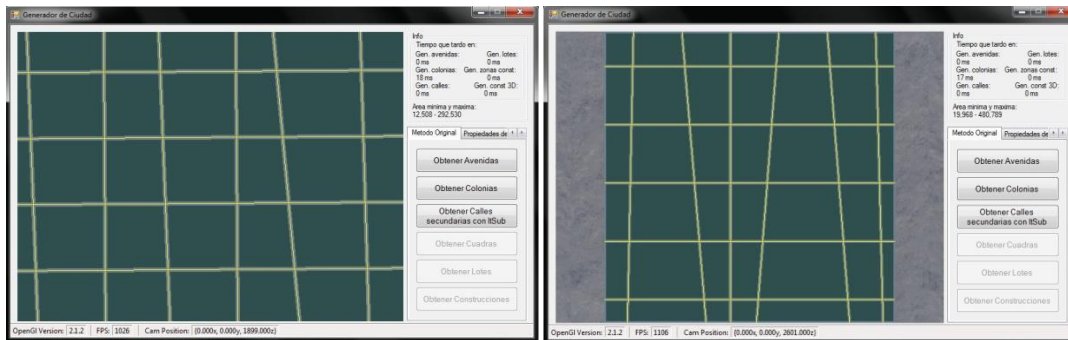


Figura 30. Avenidas principales y colonias de la aplicación.

En el estado actual de la aplicación no es posible obtener el plano radiocéntrico por razones de limitaciones del proyecto, que se mencionaron en la sección 1.6.

Las imágenes 31 y 32 muestran que se sigue conservando el mismo orden que se observa en los planos ortogonales aún a nivel calles secundarias. No siempre se presenta un patrón similar entre ciudades que al parecer son regulares. La figura 32 muestra un poco más de irregularidad a nivel calles secundarias que la figura 31.



Figura 31. Calles secundarias y cuadras de Nueva York. [18]

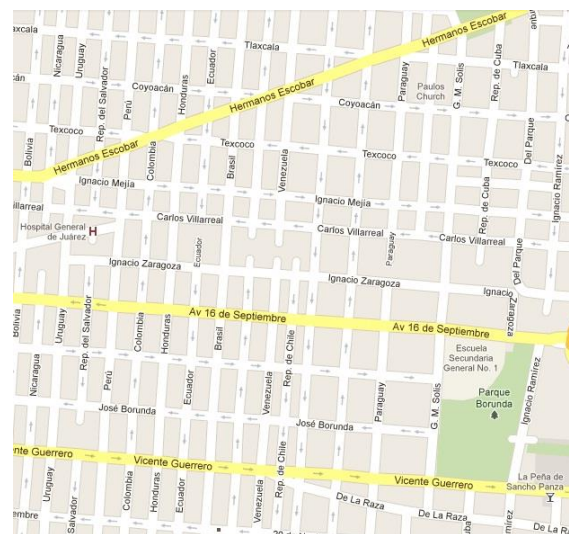


Figura 32. Calles secundarias y cuadras de Ciudad Juárez. [18]

La aplicación genera patrones más parecidos a la figura 32, independientemente de si las avenidas presentan algún nivel de irregularidad. Cabe notarse que en ésta figura aparece una zona como área verde, la cual no está definida en la aplicación, aunque si se consideraron áreas en donde no habrá ninguna construcción a nivel lotes. En todo caso, las zonas inválidas que se mencionaron en la figura 18 podrían llegar a considerarse como zonas libres, aunque su origen aún resulta incierto.

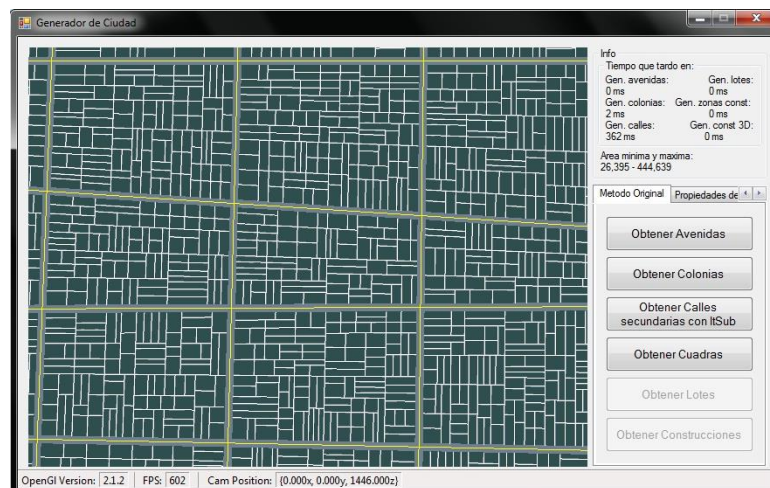


Figura 33. Calles secundarias y cuadras de la aplicación.

La figura 34 da un ejemplo claro de cómo se puede observar en la ciudad de El Paso, la distribución de los lotes de una manera muy similar a la figura 25, la cual es generada por la aplicación. De igual manera se puede ver la forma en que está dividida cada cuadra, partiéndose regularmente por la mitad y el número de lotes es diferente en ambos lados de la división. La aplicación refleja ésta característica en la figura 20.

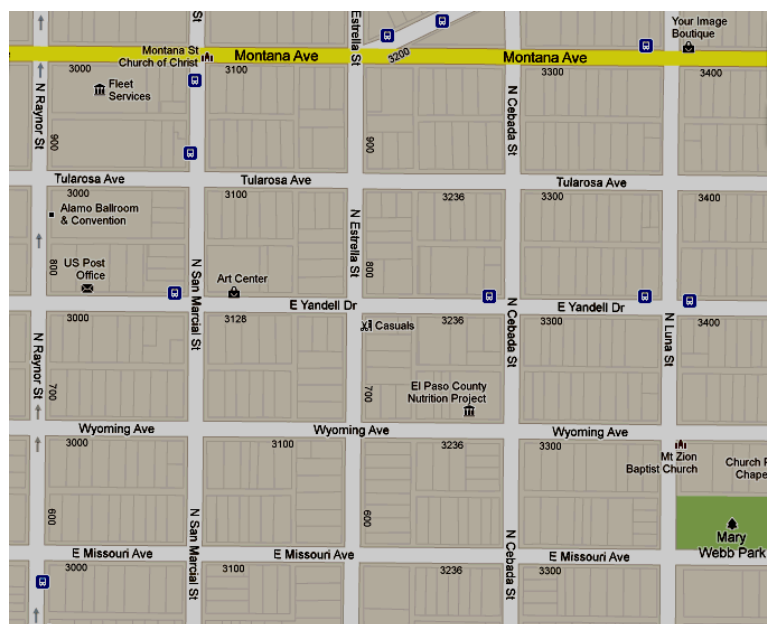


Figura 34. Distribución de lotes en El Paso, TX. [18]

Sin embargo en la aplicación del proyecto se puede apreciar esta misma distribución de una forma muy similar, en la imagen 35 la manera en que se dividió la zona del lote es la siguiente: de color verde oscuro es el lugar donde se va a hacer la construcción, y la parte de color verde claro será el espacio libre que queda del lote.

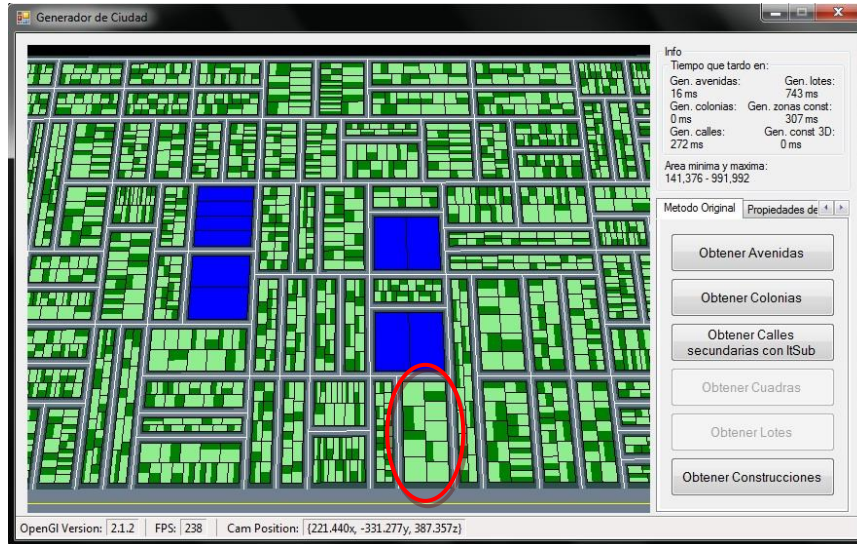


Figura 35. Lotes con zona de construcción en la aplicación.

Puede observarse también que en algunos de los lotes, como por ejemplo los que están encerrados en el óvalo rojo, no hay zonas de construcción y son simplemente lotes vacíos. La siguiente figura muestra este hecho visto desde una toma satelital.



Figura 36. Lotes en algunas cuadras de Cd. Juárez. [18]

Cabe mencionar que, puesto que en la aplicación la forma de elegir las zonas de construcción se realiza aleatoriamente, se alcanza a apreciar en cierto modo una falta de orden. En la figura 35 es notable este hecho, ya que no en todas las cuadras se sigue el mismo patrón para las zonas de construcción.

Para comparar los resultados obtenidos al generar las construcciones con otras reales, se debe tomar en cuenta que la aplicación realiza modelos muy básicos de construcción. Más sin embargo en la figura 37 se puede observar que incluso en la realidad este tipo de estructuras es común encontrarlas.



Figura 37. Edificios en el D.F. [18]

La figura 38 muestra que existen construcciones de tipo comercial o residencial que por lo general son de uno a tres pisos, y en este caso no se observa un patrón general en cuanto a si todas las construcciones de esa zona tendrán todos cierta cantidad de pisos.



Figura 38. Construcción de 3 pisos en Guadalajara. [18]

Por último, la figura 39 muestra cierta congruencia a las figuras 37 y 38 en cuanto a la decisión que se tomó de que en las zonas residenciales sólo hubiera posibilidad de tener construcciones de hasta tres pisos, mientras que a los edificios se les permite más altura.

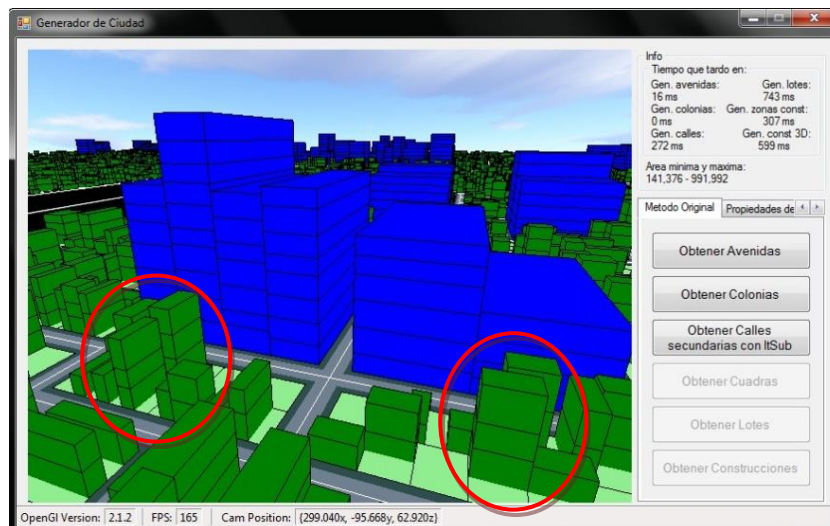


Figura 39. Construcciones generadas en la aplicación.

Los resultados obtenidos en la aplicación fueron más que satisfactorios para lo que se esperaba alcanzar en la investigación, abriendo incluso las puertas para posibles trabajos futuros y mejoras de la misma.

Capítulo 5. Discusiones, conclusiones y recomendaciones

Por último en esta sección se comprobará si las preguntas de la investigación fueron contestadas satisfactoriamente y qué tanto abarcaron las respuestas, así como si se cumplió el objetivo de la investigación y hasta qué punto se alcanzó a llegar. Finalmente se hablará de qué mejoras se podrían hacer en futuros proyectos para resolver aquello que no fue posible en éste.

5.1 Con respecto a las preguntas de investigación

- ¿Cómo se genera una ciudad virtual por medio de algoritmos?

Para poder generar algorítmicamente una ciudad se puede utilizar la subdivisión de polígonos, es necesario incluir en el proceso de las divisiones las consideraciones necesarias para lograr obtener formas y patrones propios de una ciudad, estas deben ser controladas para lograr personalizar el resultado final. Una de las consideraciones más importantes es proporcionar al método la forma en que se creará cada línea divisoria, seguido del tamaño que finalmente podrá tener cada polígono resultante.

- ¿Cómo se obtiene el patrón ortogonal que se observa en algunas ciudades?

Una forma de lograrlo es asegurar que las líneas divisorias que crea el algoritmo para cada subdivisión eviten tener algún tipo de inclinación a lo largo de cada iteración. Puede llegar a permitirse un mínimo de inclinación sin que se pierda mucho el patrón.

- ¿Cómo se delimitan los espacios de los lotes?

Esto se controla mediante la imposición de un rango de tamaños que puede tener cada lote generado mediante el algoritmo. El simple hecho de observar una imagen satelital de una cuadra da una idea de la proporción del área total que cada lote ocupa.

- ¿De qué forma se generan modelos tridimensionales que asemejen construcciones?

La extrusión de planos en 2D para convertirlos a 3D es una de las formas más sencillas de lograrlo, sólo se crea otro plano del mismo tamaño y forma pero con diferente altura y se unen para crear un piso. Todos los pisos adicionales se crean de la misma manera pero ahora tomando el último plano creado como base para el siguiente. Una forma más compleja es, como lo realiza

5.2 Con respecto al objetivo de la investigación

Puesto que se logró realizar una aplicación como prototipo que abarca las tres primeras etapas mencionadas en la sección 1.6, se puede concluir que este aspecto del objetivo fue alcanzado. En cuanto al nivel básico de realismo que se quería alcanzar se puede decir que al menos las primeras dos etapas son satisfactorias, donde en la tercera el tiempo no permitió cubrir más de un nivel básico.

Dada la naturaleza del método que se decidió utilizar (sección 2.2), el nivel de personalización al momento de crear la ciudad no pudo abarcar mucho, puesto que esto requiere de métodos matemáticos más avanzados que permitan la detección y modificación de polígonos ya una vez generados, además de requerir más tiempo para su implementación y comprobación.

Lo que sí se alcanzó respecto a este punto es que la aplicación permite modificar algunos parámetros que cambian ya sea el tamaño de las calles, si éstas presentan un patrón ortogonal o irregular a nivel ciudad, qué tan grande o pequeñas pueden ser las cuadras y el tamaño del área total que tendrá la ciudad. Todo esto modifica el resultado final al generar una ciudad.

5.3 Recomendaciones para futuras investigaciones

Como primera instancia un punto importante que puede hacer mucha diferencia en cuanto al realismo, es que en un trabajo futuro se defina la manera en que se puedan crear polígonos de cualquier forma resolviendo las complejidades que se presentaron en los procedimientos, que no estén limitados solamente a cuatro vértices como lo están en esta implementación. Esto permitiría generar un mayor número de posibles formas que pueden tomar ya sean las calles y avenidas, así como también los lotes y cuadras.

Otro punto importante sería dar la posibilidad de trabajar con curvas en la implementación, de ésta manera podría abarcarse desde todo un nuevo patrón morfológico en la ciudad (el radiocéntrico), hasta detalles más a nivel local como glorietas o simplemente que las calles puedan tener curvatura. Este punto podría resolverse si se lograra implementar el método de Esch et al [6].

En cuanto a la implementación, se puede mejorar el modo en que se generan los lotes y construcciones tomando en cuenta datos extra que ayuden a definir cómo serán generados, en lugar de valerse de la aleatoriedad como actualmente se logró implementar. Algunos datos pueden ser por ejemplo de densidad poblacional, de modo que en zonas con mayor densidad se crean más lotes con construcciones más grandes y

en las de menor haya lotes más individuales y zonas vacías. Un modo de resolverse es implementar estos datos como mapas de bits, en donde, al sobreponer el mapa de bits en el de la ciudad, se verifica según la posición en la que se encuentre la cuadra, si ésta está en una zona de alta o en una de baja densidad, y así finalmente decidir cómo resultarán los lotes y construcciones. La transición entre zonas puede ser continua o discreta.

Otros datos extra pueden ser puntos de interés, como el centro de la ciudad o alguna zona comercial importante, esto mejoraría la manera en que se crean las avenidas al indicar el lugar en donde empieza y termina alguna avenida, asegurando que estas siempre conecten con los puntos de interés. La verificación se hace de la misma manera mencionada anteriormente, utilizando mapas de bits para indicar las posiciones y poder crear las líneas que representan a cada avenida.

En otra instancia se puede incrementar la complejidad que tendrá la forma de la ciudad total, buscando asemejar más a como son las ciudades en la realidad donde no siempre tienen figuras regulares, si no que presentan más bien formas irregulares debido principalmente a restricciones geográficas o patrones de crecimiento poblacional.

En cuanto a las restricciones geográficas, esto se puede lograr implementando un método que tome en cuenta las alturas del terreno en el que se creará la ciudad, se puede utilizar un *heightmap* (mapa de alturas) que viene siendo una imagen en escala de grises donde cada coordenada de la imagen representa una elevación según el tono de gris presente. La elevación es entonces utilizada como límite para saber si esa coordenada será excluida o no del área total de la ciudad.

El implementar éstas mejoras al proyecto daría como resultado una mejor base sobre la cual poder entonces avanzar a las siguientes fases de la generación de ciudades, que sería la texturización y los sistemas misceláneos de complejidad más avanzada (sistema de tráfico, vegetación, clima, etc.).

Referencias

- [1] Y. I. Parish and P. Müller, "Procedural modeling of cities." in *SIGGRAPH '01 Proceeding of the 28th annual conference on Computer graphics and interactive techniques.*, ACM New York, NY, USA, 2001, pp. 301-308.
- [2] Esri. (2011, Noviembre) esri. Understanding our world. [Online]. "<http://www.esri.com/software/c-ityengine/features.html>"
- [3] Stefan Greuter, Jeremy Parker, Nigel Stewart, and Geoff Leach, "Real-time procedural generation of pseudo infinite cities." in *Proceeding of the 1st international conference on Computer graphics and interactive techniques.*, Australasia and South East Asia., 2003, pp. 87-ff.
- [4] Brasil Weber, Pascal Müller, Peter Wonka, and Markus Gross, "Interactive Geometrics Simulation of 4D Cities.", *Computer Graphics Forum*, vol. 28, no. 2, pp. 481-492, April 2009.
- [5] S.A. Groenewegen, R.M. Smelik, K.J. Kraker, and R. Bidarra, *Procedural City Layout Generation Based on Urban Land Use Models.*: EUROGRAPHICS 2009, 2009.
- [6] G. Esch, G. Chen, P. Wonka, P. Müller, and E. Zhang, "Interactive Procedural Street Modeling,," *ACM Transaction on Graphics (TOG).*, vol. 27, no. 3, 2008.
- [7] N. Rudzincz and C. Verbrugge, "An iterated subdivision algorithm for procedural road plan generation." in *Game-On NA.*, 2008.
- [8] P. Müller, P. Wonka, S. Haegler, A. Ulmer, and L. V. Gool, "Procedural modeling of buildings." in *SIGGRAPH Internacional Conference on Computer Graphics and Interactive Techniques.*, ACM New York, NY, USA., 2006, pp. 614-623.
- [9] Eugene Zhang, James Hays, and Greg Turk, "Interactive Tensor Field Design and Visualization on Surfaces", *Transactions on Visualization and Computer Graphics*, vol. 13, no. 1, pp. 94-106, January-February 2007.
- [10] Juli Esteban Noguera, *Elementos de ordenación urbana*. Barcelona: Edicions UPC, Mayo 1998.
- [11] Rodolfo E. Castagna, *Ciudad: Recidesia Total*. Argentina: Ediciones Nueva Vision, 1984.
- [12] Elvira Maycote P. and Fernando Lozada I., *Gestión Urbana ¿Más estado o más*

mercado? Mexico: Universidad Autónoma de Ciudad Juárez, 2010.

- [13] Domingo Gómez O., *Ordenación territorial*. México: Ediciones Mundi-Prensa, 2008.
- [14] Gideon Golany, *Planificación de nuevas ciudades, principios y prácticas*. México D.F.: Limusa, 1985.
- [15] G. P., *Diccionario de Geografía*. Madrid, 1991.
- [16] J. Bosque, *Sistemas de Información Geográfica*, 2nd ed. Madrid, Rialp, 1997.
- [17] Max Weber, *La dominación no legítima (Tipología de las ciudades)*. México: Economía y Sociedad, 1964.
- [18] Google, Google Maps. [Online]. “maps.google.com”.

Apéndices

Apéndice 1. Ecuaciones de Algebra Lineal.

Ecuación de la recta, no vertical:

$$y = mx + b$$

Ecuación de la recta, vertical:

$$x = a$$

Ecuación de la pendiente dados dos puntos, cuando $x_2 - x_1$ es diferente a 0:

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

Intersección entre dos rectas en un mismo plano, en donde para cada recta b es la ordenada al origen y m es la pendiente:

$$i. \quad m_1x + b_1 = m_2x + b_2$$

$$ii. \quad m_1x - m_2x = b_2 - b_1$$

$$iii. \quad x(m_1 - m_2) = b_2 - b_1$$

$$iv. \quad x = \frac{b_2 - b_1}{m_1 - m_2}$$

Distancia entre dos puntos:

$$d = \sqrt{(y_2 - y_1)^2 + (x_2 - x_1)^2}$$

Área de un cuadrilátero en dos dimensiones, dados los puntos P1, P2, P3, P4:

$$\begin{aligned} \text{área} = & |(P1x * P2y - P1y * P2x) + (P2x * P3y - P2y * P3x) \\ & + (P3x * P4y - P3y * P4x) + (P4x * P1y - P4y * P1x)| \end{aligned}$$

Apéndice 2. Inicialización de variables

```
#region Inicializacion de variables
shapeBaseGlobal = new Shapes();
ListaShapeAvs = new List<Shapes>[2] { new List<Shapes>(), new List<Shapes>() };
int NCallesHor, NCallesVer;
float[] TmpSeparaciones = new float[2];
NCallesHor = RndNum2.Next(3, 7);
NCallesVer = RndNum2.Next(3, 7);
TmpSeparaciones[0] = ((Lims[3] - Lims[2]) / NCallesHor);
TmpSeparaciones[1] = ((Lims[1] - Lims[0]) / NCallesVer);
float TempAvance = TmpSeparaciones[0] * (0.6f);
float pendiente;
int rndnum = RndNum2.Next(3);
#endregion
```

Apéndice 3. Definición de la recta implementada

```
#region Definicion de la recta que representa la calle
rndnum = RndNum2.Next(3);
switch (rndnum) // Decide la direccion de la calle, cada uno con probabilidad 1/3
{
    // direccion = pendiente
    case 0: // Se inclina hacia arriba
        pendiente = (1.0f / (float)RndNum2.Next(12, 57));
    }
}
```

```

        break;
    case 1:          // Se inclina hacia abajo
        pendiente = (-1.0f / (float)RndNum2.Next(12, 57));
        break;
    default:
        pendiente = 0;          // No tiene inclinacion
        break;
}
float b_nxt = (Lims[2] + TempAvance) - (pendiente * Lims[0]);
#endregion

```

Apéndice 4. Pseudocódigo de QuickSort

```

funcion quicksort('arreglo')
Si longitud('arreglo') ≤ 1
    --retornar 'arreglo' // un 'arreglo' de uno o cero elementos ya está acomodado
Seleccionar y remover un valor pivote 'pivote' de 'arreglo'
crear dos listas 'menores' y 'mayores'
Para cada 'x' en 'arreglo'
    --Si 'x' ≤ 'pivote' entonces agregar 'x' a 'menores'
    --Si no agregar 'x' a 'mayores'
// la función concatenar concatena las listas de mayor a menor
// hace cada vez dos llamadas recursivas
retornar concatenar(quicksort('menores'), 'pivote', quicksort('mayores'))

```

Apéndice 5. Calculo de los puntos de intersección

```

#region Calculo de los puntos
if (ListaShapeAvs[0].Count != 0)
{
    float x_intersect, y_intersect;
    // Se verifica si la recta que se creará es paralela o no a la anterior, de ser así se tiene que evitar que
    // intersecten.
    if (pendiente - ListaShapeAvs[0][ListaShapeAvs[0].Count - 1].m != 0) // Son paralelas si la resta
    // de las pendientes da 0, no se intersectarian
    {
        do
        {
            x_intersect = (b_nxt - ListaShapeAvs[0][ListaShapeAvs[0].Count - 1].b) / (pendiente -
            ListaShapeAvs[0][ListaShapeAvs[0].Count - 1].m);
            y_intersect = (pendiente * x_intersect) + b_nxt;
            if (x_intersect > Lims[1] || x_intersect < Lims[0]) // No hay interseccion dentro del area
            global?
            {
                if ((float)Math.Round(pendiente * Lims[1], 0) + b_nxt < Lims[3]) // La Y en (X = Limite
                superior X Global) es menor al (Limite superior Y Global)?
                {
                    ListaShapeAvs[0].Add(new Carretera(Lims, TempAvance, pendiente));
                    TempAvance += (TmpSeparaciones[0]);
                }
            }
            else // Si no, modificar la pendiente
                pendiente -= 0.010f;
        }
        else // Hubo interseccion, modificar la pendiente
            pendiente += 0.006f;
    } while (!(x_intersect > Lims[1] || x_intersect < Lims[0]) || !((pendiente * Lims[1]) + b_nxt <
    Lims[3]));
}
else // Como son paralelas, solo se verifica que no salga del limite global Lims[]

```

```

    {
        ListaShapeAvs[0].Add(new Carretera(Lims, TempAvance, pendiente));
        if (ListaShapeAvs[0][ListaShapeAvs[0].Count - 1].Vertices[0].Y < Lims[3] &&
            ListaShapeAvs[0][ListaShapeAvs[0].Count - 1].Vertices[1].Y < Lims[3])
            TempAvance += (TmpSeparaciones[0]);
        else
            ListaShapeAvs[0].RemoveAt(ListaShapeAvs[0].Count - 1);
    }
}
else // La primera avenida solo puede intersectar con el limite global Lims[]
{
    ListaShapeAvs[0].Add(new Carretera(Lims, TempAvance, pendiente));
    if (ListaShapeAvs[0][ListaShapeAvs[0].Count - 1].Vertices[0].Y > Lims[2] &&
        ListaShapeAvs[0][ListaShapeAvs[0].Count - 1].Vertices[1].Y > Lims[2])
        TempAvance += (TmpSeparaciones[0]);
    else
        ListaShapeAvs[0].RemoveAt(ListaShapeAvs[0].Count - 1);
}
#endregion

```

Apéndice 6. Procesos para obtener las Colonias

```

private void ObtenerColonias()
{
    ...
    ObtenerIntersecciones();
    for (int i = 0; i < ListaIntersecciones.Length; i++)
    {
        ListaIntersecSorteada[i] = QuickSort_XAsc(ListaIntersecciones[i]);
        ListaIntersecSorteada[i].TrimExcess();
    }
    ObtenerAreasDeColonias();
    ...
}

ObtenerIntersecciones()
{
    int LineasHor = tmpListaLineas[0].Count, LineasVer = tmpListaLineas[1].Count;
    ListaIntersecciones = new List<Vector3>[LineasHor];
    ListaIntersecSorteada = new List<Vector3>[LineasHor];
    int n_linea = 0;
    if (TotalAvenidas > 0)
    {
        for (int i = 0; i < LineasHor; i++)
        {
            ListaIntersecciones[n_linea] = new List<Vector3>();
            #region Chequeo de calles horizontales-verticales
            for (int j = 0; j < LineasVer; j++)
            {
                float Xintersect, Yintersect;
                if (!tmpListaLineas[1][j].IsVertical)
                {
                    #region Interseccion entre no vertical - no vertical
                    // m1x + b1 = m2x + b2
                    // m1x - m2x = b2 - b1 --> x(m1-m2) = b2 - b1
                    // x = (b2-b1) / (m1-m2)
                    Xintersect = (tmpListaLineas[1][j].b - tmpListaLineas[0][i].b) / (tmpListaLineas[0][i].m -
                    tmpListaLineas[1][j].m);
                    Xintersect = (float)Math.Round(Xintersect, 2);

```

```

        // y = mx + b
        Yintersect = (tmpListaLineas[0][i].m * Xintersect) + tmpListaLineas[0][i].b;
        Yintersect = (float)Math.Round(Yintersect, 2);
        if (Xintersect >= LimiteInfX &&
            Xintersect <= LimiteSupX &&
            Yintersect >= LimiteInfY &&
            Yintersect <= LimiteSupY)
            ListaIntersecciones[n_linea].Add(new Vector3(Xintersect, Yintersect, 1.0f));
        #endregion
    }
    else
    {
        #region Interseccion entre no vertical - vertical
        Xintersect = tmpListaLineas[1][j].Vertices[0].X;
        Xintersect = (float)Math.Round(Xintersect, 2);
        Yintersect = (tmpListaLineas[0][i].m * Xintersect) + tmpListaLineas[0][i].b;
        Yintersect = (float)Math.Round(Yintersect, 2);
        if (Xintersect >= LimiteInfX &&
            Xintersect <= LimiteSupX &&
            Yintersect >= LimiteInfY &&
            Yintersect <= LimiteSupY)
            ListaIntersecciones[n_linea].Add(new Vector3(Xintersect, Yintersect, 1.0f));
        #endregion
    }
}
#endregion
}
#region
...Chequeo de calles horizontales-horizontales...
#endregion

if (ListaIntersecciones[n_linea].Count > 0) // Hubo intersecciones?
    n_linea++;
}
}
}
private void ObtenerAreasDeColonias()
{
    ListaShapeColonias = new List<Shapes>();
    Vector3[] tmpVectors;
    for (int i = 0; i < ListaIntersecSorteada.Length; i++)
        for (int j = 0; j < ListaIntersecSorteada[i].Count; j++)
        {
            if ((i + 1) >= ListaIntersecSorteada.Length)
                break;
            if ((j + 1) >= ListaIntersecSorteada[i].Count || (j + 1) >= ListaIntersecSorteada[i + 1].Count)
                break;
            tmpVectors = new Vector3[4];
            tmpVectors[0] = new Vector3(ListaIntersecSorteada[i][j].X + offsetAnchoAvenida,
                ListaIntersecSorteada[i][j].Y + offsetAnchoAvenida, ListaIntersecSorteada[i][j].Z);
            tmpVectors[1] = new Vector3(ListaIntersecSorteada[i + 1][j].X + offsetAnchoAvenida,
                ListaIntersecSorteada[i + 1][j].Y - offsetAnchoAvenida, ListaIntersecSorteada[i + 1][j].Z);
            tmpVectors[2] = new Vector3(ListaIntersecSorteada[i + 1][j + 1].X - offsetAnchoAvenida,
                ListaIntersecSorteada[i + 1][j + 1].Y - offsetAnchoAvenida, ListaIntersecSorteada[i + 1][j + 1].Z);
            tmpVectors[3] = new Vector3(ListaIntersecSorteada[i][j + 1].X - offsetAnchoAvenida,
                ListaIntersecSorteada[i][j + 1].Y + offsetAnchoAvenida, ListaIntersecSorteada[i][j + 1].Z);
            ListaShapeColonias.Add(new Parcela(tmpVectors));
        }
    TotalColonias = ListaShapeColonias.Count;
    ...
}

```

```
}
```

Apéndice 7. Procedimientos para generar las calles secundarias

```
GenerarCalles_ItSub()
{
    ...
    for (int i = 0; i < TotalColonias; i++)
    {
        ListaShapePOversized.Add(new Parcela(ListaShapeColonias[i].Vertices, true));
        do
        {
            float Ratio;
            int iteraciones = 0; // Controla que el proceso no quede ciclado
            do
            {
                iteraciones++;
                DatosBiseccion = new List<Vector3>[3] { new List<Vector3>(4), new List<Vector3>(4),
new List<Vector3>(2) };
                DatosBiseccion = ListaShapePOversized[0].getAcceptableBisector(Amin, Rmax, iteraciones,
!isCiudadOrtogonal);
                Ratio = ListaShapePOversized[0].PolygonDiameterWidth_Ratio();
            } while (Ratio > Rmax && iteraciones < 50);
            if (iteraciones < 50) // Si se pudo biseccionar el poligono
            {
                ListaShapeRoads.Add(new Carretera(DatosBiseccion[2].ToArray(), Color.White));
                if (AreaCuadrilatero(DatosBiseccion[0]) > Amax)
                    ListaShapePOversized.Add(new Parcela(DatosBiseccion[0].ToArray(), true));
                else
                {
                    Shapes tmpShape = new Parcela(DatosBiseccion[0].ToArray(), true);
                    float tmpratio = tmpShape.PolygonDiameterWidth_Ratio();
                    ListaShapeAllots.Add(new Parcela(DatosBiseccion[0].ToArray(), false, tmpratio));
                }
                if (AreaCuadrilatero(DatosBiseccion[1]) > Amax)
                    ListaShapePOversized.Add(new Parcela(DatosBiseccion[1].ToArray(), true));
                else
                {
                    Shapes tmpShape = new Parcela(DatosBiseccion[1].ToArray(), true);
                    float tmpratio = tmpShape.PolygonDiameterWidth_Ratio();
                    ListaShapeAllots.Add(new Parcela(DatosBiseccion[1].ToArray(), false, tmpratio));
                }
            }
            else // No se pudo biseccionar, se agrega así como está a la lista de Cuadras
                ListaShapeAllots.Add(new Parcela(ListaShapePOversized[0].Vertices, false, 20));
            ListaShapePOversized.RemoveAt(0);
        } while (ListaShapePOversized.Count > 0);
    }
    ...
}
```

Apéndice 8. Implementación de ItSub – Subdivision de polígonos

```
public List<Vector3>[] getAcceptableBisector(float AreaMin, float Rmax = 16, int contador = 0, bool
bIrregular = false)
{
    List<Vector3>[] PoligonosBisectados;
    Vector3[] BisectorLine;
```

```

// Decide aleatoriamente si hacer la division a lo largo o a lo ancho
switch (rand.Next(2))
{
    case 0:
        while (true)
        {
            contador++; // Contador que sirve para controlar que no se cicle indefinidamente el proceso
            BisectorLine = new Vector3[2];
            PoligonosBisectados = new List<Vector3>[3] { new List<Vector3>(4), new List<Vector3>(4),
new List<Vector3>(2) };

            BisectorLine = getBisectorLine(true, blrregular);

            PoligonosBisectados[0].Add(new Vector3(Vertices[0]));
            PoligonosBisectados[0].Add(new Vector3(Vertices[1]));
            PoligonosBisectados[0].Add(new Vector3(BisectorLine[1]));
            PoligonosBisectados[0].Add(new Vector3(BisectorLine[0]));

            PoligonosBisectados[1].Add(new Vector3(BisectorLine[0]));
            PoligonosBisectados[1].Add(new Vector3(BisectorLine[1]));
            PoligonosBisectados[1].Add(new Vector3(Vertices[2]));
            PoligonosBisectados[1].Add(new Vector3(Vertices[3]));

            if (PolygonDiameterWidth_Ratio(PoligonosBisectados[0].ToArray()) > Rmax
                || PolygonDiameterWidth_Ratio(PoligonosBisectados[1].ToArray()) > Rmax)
                if (contador < 100)
                    goto case 1;

            float Area_1 = AreaCuadrilatero(PoligonosBisectados[0]);
            float Area_2 = AreaCuadrilatero(PoligonosBisectados[1]);

            if ((Area_1 >= AreaMin && Area_2 >= AreaMin) || contador >= 100)
            {
                PoligonosBisectados[2].Add(new Vector3(BisectorLine[0]));
                PoligonosBisectados[2].Add(new Vector3(BisectorLine[1]));
                return PoligonosBisectados;
            }
            if (contador == 50)
                goto case 1;
        }
    case 1:
        while (true)
        {
            contador++; // Contador que sirve para controlar que no se cicle indefinidamente el proceso
            BisectorLine = new Vector3[2];
            PoligonosBisectados = new List<Vector3>[3] { new List<Vector3>(4), new
List<Vector3>(4), new List<Vector3>(2) };

            BisectorLine = getBisectorLine(false, blrregular);

            PoligonosBisectados[0].Add(new Vector3(Vertices[0]));
            PoligonosBisectados[0].Add(new Vector3(BisectorLine[0]));
            PoligonosBisectados[0].Add(new Vector3(BisectorLine[1]));
            PoligonosBisectados[0].Add(new Vector3(Vertices[3]));

            PoligonosBisectados[1].Add(new Vector3(BisectorLine[0]));
            PoligonosBisectados[1].Add(new Vector3(Vertices[1]));
            PoligonosBisectados[1].Add(new Vector3(Vertices[2]));
            PoligonosBisectados[1].Add(new Vector3(BisectorLine[1]));

```

```

        if (PolygonDiameterWidth_Ratio(PoligonosBisectados[0].ToArray()) > Rmax
            || PolygonDiameterWidth_Ratio(PoligonosBisectados[1].ToArray()) > Rmax)
            if (contador < 100)
                goto case 0;

        float Area_1 = AreaCuadrilatero(PoligonosBisectados[0]);
        float Area_2 = AreaCuadrilatero(PoligonosBisectados[1]);

        if ((Area_1 >= AreaMin && Area_2 >= AreaMin) || contador >= 100)
        {
            PoligonosBisectados[2].Add(new Vector3(BisectorLine[0]));
            PoligonosBisectados[2].Add(new Vector3(BisectorLine[1]));
            return PoligonosBisectados;
        }
        if (contador == 50)
            goto case 0;
    }
}
}

```

Apéndice 9. Implementación de ItSub – Como obtener la línea divisoria

```

private Vector3[] getBisectorLine(bool IsVertical, bool boIrregular)
{
    float X1, X2, Y1, Y2;
    Vector3[] carretera = new Vector3[2] { new Vector3(), new Vector3() };
    int divisor, maxmult, mult;
    divisor = rand.Next(2, 4);
    maxmult = divisor - 1;
    if (maxmult > 1)
        mult = rand.Next(1, maxmult);
    else
        mult = 1;
    if (IsVertical)
    {
        #region Calculos de puntos mas sencillos
        X1 = (Vertices[3].X - (Vertices[3].X - Vertices[0].X) / divisor) * mult;
        Y1 = (Vertices[3].Y - (Vertices[3].Y - Vertices[0].Y) / divisor) * mult;
        if (boIrregular) // boIrregular especifica si el punto final de la linea bisectora corresponde al
        punto inicial
        {
            divisor = rand.Next(2, 3);
            maxmult = divisor - 1;
            if (maxmult > 1)
                mult = rand.Next(1, maxmult);
            else
                mult = 1;
        }
        X2 = (Vertices[2].X - (Vertices[2].X - Vertices[1].X) / divisor) * mult;
        Y2 = (Vertices[2].Y - (Vertices[2].Y - Vertices[1].Y) / divisor) * mult;
        carretera[0].X = X1;
        carretera[0].Y = Y1;
        carretera[0].Z = 1.0f;
        carretera[1].X = X2;
        carretera[1].Y = Y2;
        carretera[1].Z = 1.0f;
        #endregion
    }
    else
    {

```

```

    #region Calculos de puntos mas sencillos
    X1 = (Vertices[1].X - (Vertices[1].X - Vertices[0].X) / divisor) * mult;
    Y1 = (Vertices[1].Y - (Vertices[1].Y - Vertices[0].Y) / divisor) * mult;
    X2 = (Vertices[2].X - (Vertices[2].X - Vertices[3].X) / divisor) * mult;
    Y2 = (Vertices[2].Y - (Vertices[2].Y - Vertices[3].Y) / divisor) * mult;
    carretera[0].X = X1;
    carretera[0].Y = Y1;
    carretera[0].Z = 1.0f;
    carretera[1].X = X2;
    carretera[1].Y = Y2;
    carretera[1].Z = 1.0f;
    #endregion
}
return carretera;
}

```

Apéndice 10. Definición de los vértices de las Cuadras

```

private void GenerarCuadras()
{
    ...
    TotalCuadras = ListaShapeAllots.Count;
    for (int i = 0; i < TotalCuadras; i++)
    {
        ListaShapeAllots[i].Vertices[0].X += (offsetAnchoCalle / 2);
        ListaShapeAllots[i].Vertices[0].Y += (offsetAnchoCalle / 2);

        ListaShapeAllots[i].Vertices[1].X += (offsetAnchoCalle / 2);
        ListaShapeAllots[i].Vertices[1].Y -= (offsetAnchoCalle / 2);

        ListaShapeAllots[i].Vertices[2].X -= (offsetAnchoCalle / 2);
        ListaShapeAllots[i].Vertices[2].Y -= (offsetAnchoCalle / 2);

        ListaShapeAllots[i].Vertices[3].X -= (offsetAnchoCalle / 2);
        ListaShapeAllots[i].Vertices[3].Y += (offsetAnchoCalle / 2);
    }
    ...
}

```

Apéndice 11. Generación de los Lotes

```

private void GenerarLotes()
{
    ...
    // Categorizacion de cuadras por proporcion Diametro:Ancho
    for (int i = 0; i < TotalCuadras; i++)
    {
        float area = AreaCuadrilatero(ListaShapeAllots[i].Vertices);
        float DiamWid_Ratio = ListaShapeAllots[i].PolygonDiameterWidth_Ratio();
        if (area > Amax && DiamWid_Ratio > Rmax) // Area invalida muy grande, hacer lote valdío
            ListaShapeAllots[i].TipoDeLote = 0;
        else if (area <= Amax && area > 3500 && DiamWid_Ratio < 7) // Area grande, sera para edificios
            ListaShapeAllots[i].TipoDeLote = 1;
        else // (area <= 3000) Area media-chica, o muy alargada. Puede ser residencial o areas verdes
            ListaShapeAllots[i].TipoDeLote = 2;
    }

    // Subdivision de las cuadras para obtener lotes (ItSub), segun el tipo de lote.
}

```

```

for (int i = 0; i < TotalCuadras; i++)
{
    int caso = ListaShapeAllots[i].TipoDeLote;
    switch (caso)
    {
        case 0: // Areas invalidas
            ListaShapeLotes.Add(new Parcela(ListaShapeAllots[i].Vertices, false, Color.Black, caso,
false));
            break;
        case 1: // Edificios
            List<Vector3> resultz = ListaShapeAllots[i].CalcularLotes_Edificios();
            for (int j = 0; j < resultz.Count; j += 4)
            {
                ListaShapeLotes.Add(new Parcela(resultz.GetRange(j, 4).ToArray(), false,
Color.CornflowerBlue, caso, false));
                ListaShapeLotes_Contornos.Add(new Parcela(resultz.GetRange(j, 4).ToArray(), false,
Color.Black, caso, true));
            }
            break;
        case 2: // Residencias
            List<Vector3> results = ListaShapeAllots[i].CalcularLotes_Residencias();
            for (int j = 0; j < results.Count; j += 4)
            {
                ListaShapeLotes.Add(new Parcela(results.GetRange(j, 4).ToArray(), false,
Color.LightGreen, caso, false));
                ListaShapeLotes_Contornos.Add(new Parcela(results.GetRange(j, 4).ToArray(), false,
Color.Black, caso, true));
            }
            break;
        default: throw new Exception("Error: tipo de Lote desconocido. Salio TipoDeLote < 0 o
TipoDeLote > 2");
    }
}
...
}

```

Apéndice 12. Constructor de la función que genera las edificaciones.

```

public Construcccion3D(Vector3[] Vertices_Base, int Tipo_deLote)
{
    NumeroDePisos = 0;
    Vector3[] SiguientePiso;
    ListaVertices = new List<Vector3>();
    ListaColores = new List<int>();
    ListaIndices = new List<int>();
    ListaVertices.AddRange(Vertices_Base);
    switch (Tipo_deLote)
    {
        case 0: // Zona invalida, con genera modelo 3D
            break;
        case 1: // Modelos 3D para los edificios
            NumeroDePisos = rand.Next(4, 11);
            for (int i = 0; i < NumeroDePisos; i++)
            {
                SiguientePiso = new Vector3[4];
                float AlturaAdd = (float)rand.Next(5, 9);
                for (int j = 0; j < 4; j++)
                {
                    SiguientePiso[j] = ListaVertices[(i * 4) + j];
                }
            }
        }
    }
}

```

```

        SiguientePiso[j].Z += AlturaAdd;
    }
    ListaVertices.AddRange(SiguientePiso);
}
break;
case 2: // Modelos 3D para las residencias
    NumeroDePisos = rand.Next(1, 4);
    for (int i = 0; i < NumeroDePisos; i++)
    {
        SiguientePiso = new Vector3[4];
        float AlturaAdd = (float)rand.Next(5, 9);
        for (int j = 0; j < 4; j++)
        {
            SiguientePiso[j] = ListaVertices[(i * 4) + j];
            SiguientePiso[j].Z += AlturaAdd;
        }
        ListaVertices.AddRange(SiguientePiso);
    }
    break;
}
Vertices = ListaVertices.ToArray();
switch (Tipo_deLote)
{
    case 0: break;
    case 1:
        foreach (Vector3 vec in Vertices) // Color de los edificios
            ListaColores.Add(ColorToRgba32(Color.Blue));
        break;
    case 2:
        foreach (Vector3 vec in Vertices) // Color de las residencias
            ListaColores.Add(ColorToRgba32(Color.Green));
        break;
}
Colors = ListaColores.ToArray();
}

```

Anexo

Propuesta de investigación (Protocolo corto)

Universidad Autónoma de Ciudad Juárez
Instituto de Ingeniería y Tecnología



Protocolo de Investigación

Versión Corta

Propuesta del tema para la materia de Proyecto de Titulación

Nombres: Brenda Berenice Candia Rey	Matrícula: 86355
Programa Académico: Ingeniería en Sistemas Computacionales	
Departamento: Ingeniería Eléctrica y Computación	

Título:

Generación de ciudades virtuales en 3D por medio de procedimientos algorítmicos.

Objetivo:

Implementar un prototipo de ciudad virtual en 3D, utilizando una selección de las técnicas y métodos procedimentales propuestos para lograr un nivel básico de realismo. El alcance de este proyecto es lograr la implementación de las tres primeras fases, de un total de cinco, de la metodología investigada para generar una ciudad, según se lista a continuación:

1. Red de calles y carreteras.
2. Distribución de lotes.
3. Generación de construcciones.
4. Texturización.
5. Sistemas misceláneos (vegetación, personas, clima, tráfico).

Descripción del Problema:

La creación de grandes ambientes virtuales para la industria del entretenimiento toma mucho tiempo y esfuerzo si se realiza utilizando medios tradicionales como modeladores gráficos o grupos artísticos (modelación a mano). Además, la implementación de los modelos resulta compleja en cuanto a la programación requerida.

Solución Propuesta:

Se puede generar contenido gráfico para modelar una ciudad virtual en 3D mediante la utilización de técnicas procedimentales que, dependiendo de su complejidad, dan la posibilidad de hacerlo en tiempo real y con un nivel aceptable de realismo. En el presente proyecto se pretende revisar la literatura especializada para implementar algunas de las técnicas propuestas y que completen las tres primeras fases de la

metodología.

Justificación:

En diversos sectores de la industria, sobre todo en la del entretenimiento, existe la necesidad de poder crear contenido ambiental (como una ciudad) minimizando la utilización de datos reales de entrada o el trabajo enteramente manual y creativo de diseño, lo cual consume tiempo y es costoso. Considerando lo anterior, creemos que es de gran utilidad la implementación de un procedimiento computacional que automatice en buena medida este proceso y que el resultado tenga un nivel aceptable de realismo e incluso pueda ser personificado hasta cierto punto.

Metodología Propuesta:

1. Identificar las técnicas procedimentales que aparte de generar el contenido deseado pueda éste ser hasta cierto punto modificable.
2. Precisar qué herramientas de programación facilitan la implementación de estas técnicas para poder crear un prototipo funcional a fin de utilizarlas.
3. Implementar en un prototipo de aplicación las técnicas seleccionadas en la investigación previa.
4. Definir medidas cuantitativas y/o cualitativas para estimar la eficiencia del prototipo.
5. Comparación de resultados variando la complejidad del modelo urbano generado y las características del hardware utilizado.

Observaciones del evaluador:

☐ Aprobado ☐ Aprobado con condición ☐ Rechazado

Fecha de terminación del proyecto: Mayo del 2012

Nombre del asesor responsable: Saúl González Campos

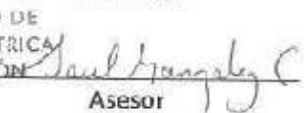

Alumno

UACJ


Evaluador


Evaluador


Maestro de la materia


Asesor

DEPARTAMENTO DE
INGENIERÍA ELÉCTRICA
Y COMPUTACIÓN

Protocolo de Investigación

Versión Corta

Propuesta del tema para la materia de Proyecto de Titulación

Nombres: Jesús Carlos Ruvalcaba Pérez	Matrícula: 87670
Programa Académico: Ingeniería en Sistemas Computacionales	
Departamento: Ingeniería Eléctrica y Computación	

Título:

Generación de ciudades virtuales en 3D por medio de procedimientos algorítmicos.

Objetivo:

Implementar un prototipo de ciudad virtual en 3D, utilizando una selección de las técnicas y métodos procedimentales propuestos para lograr un nivel básico de realismo. El alcance de este proyecto es lograr la implementación de las tres primeras fases, de un total de cinco, de la metodología investigada para generar una ciudad, según se lista a continuación:

1. Red de calles y carreteras.
2. Distribución de lotes.
3. Generación de construcciones.
4. Texturización.
5. Sistemas misceláneos (vegetación, personas, clima, tráfico).

Descripción del Problema:

La creación de grandes ambientes virtuales para la industria del entretenimiento toma mucho tiempo y esfuerzo si se realiza utilizando medios tradicionales como modeladores gráficos o grupos artísticos (modelación a mano). Además, la implementación de los modelos resulta compleja en cuanto a la programación requerida.

Solución Propuesta:

Se puede generar contenido gráfico para modelar una ciudad virtual en 3D mediante la utilización de técnicas procedimentales que, dependiendo de su complejidad, dan la posibilidad de hacerlo en tiempo real y con un nivel aceptable de realismo. En el presente proyecto se pretende revisar la literatura especializada para implementar algunas de las técnicas propuestas y que completen las tres primeras fases de la

metodología.

Justificación:

En diversos sectores de la industria, sobre todo en la del entretenimiento, existe la necesidad de poder crear contenido ambiental (como una ciudad) minimizando la utilización de datos reales de entrada o el trabajo enteramente manual y creativo de diseño, lo cual consume tiempo y es costoso. Considerando lo anterior, creemos que es de gran utilidad la implementación de un procedimiento computacional que automatice en buena medida este proceso y que el resultado tenga un nivel aceptable de realismo e incluso pueda ser personificado hasta cierto punto.

Metodología Propuesta:

1. Identificar las técnicas procedimentales que aparte de generar el contenido deseado pueda éste ser hasta cierto punto modificable.
2. Precisar qué herramientas de programación facilitan la implementación de estas técnicas para poder crear un prototipo funcional a fin de utilizarlas.
3. Implementar en un prototipo de aplicación las técnicas seleccionadas en la investigación previa.
4. Definir medidas cuantitativas y/o cualitativas para estimar la eficiencia del prototipo.
5. Comparación de resultados variando la complejidad del modelo urbano generado y las características del hardware utilizado.

Observaciones del evaluador:

☐ Aprobado ☐ Aprobado con condición ☐ Rechazado

Fecha de terminación del proyecto: Mayo del 2012

Nombre del asesor responsable: Saul González Campos

Carlos Baulcaba
Alumno

Evaluador
[Signature]
Evaluador
DEPARTAMENTO DE
INGENIERÍA ELÉCTRICA
Y COMPUTACIÓN
Maestro de la materia
Asesor
Saul González Campos