Universidad Autónoma Metropolitana
Unidad Azcapotzalco

División de Ciencias Básicas e Ingeniería

Licenciatura en Ingeniería en Computación

Reporte Final

# Síntesis procedural aproximada de modelos urbanos 3D a partir de imágenes de mapa virtual de Google maps.

Jessika Cecilia Hernández García 208336164

Trimestre 2013 Primavera

21 de agosto de 2013

Asesor: Dr. Risto Fermin Rangel Kuoppa

Profesor Titular

Departamento de Sistemas

# Tabla de contenido

## AGRADECIMIENTOS

*Antes que nada a Dios, por darme tantas bendiciones en cada paso que doy y por darme la oportunidad de terminar esta etapa de mi vida.*

*A mis papás por siempre esta apoyándome en cada decisión que tomo y por su consejo.*

*A mis hermanos que siempre están haciéndome reír y están en cada momento que los necesito.*

*Y gracias mis familiares que han estado a mi lado apoyándome en cada paso que doy.*

*Gracias al profesor Marco Antonio Gutiérrez Villegas, por facilitarme las herramientas para mi formación académica.*

*Gracias al profesor Risto, por guiarme en la elaboración de este proyecto y por los conocimientos transmitidos a lo largo de la carrera.*

*A mi mejor amiga Nayeli que siempre ha estado conmigo en las buenas y en las malas, gracias por esos consejos que me das y saber escuchar.*

*A mi mejor amiga de la carrera Marisol que siempre ha estado en cada momento especial y gracias a ella crecí como persona, disfrutando momentos felices y superando los malos momentos juntas.*

*A mi amigo Bernardo que durante la carrera me brindo su apoyo y me enseño a que siempre investigando se pueden resolver las cosas.*

*Y a todos los amigos que encontré durante la carrera, gracias a ellos fue una gran experiencia esta etapa*.

# RESUMEN

En este documento se describe la síntesis procedual aproximada de modelos urbanos 3D a partir de imágenes de mapa virtual de Google maps. La síntesis procedual es una técnica que genera objetos similares durante la ejecución del programa en este caso se utiliza para la generación de los edificios en el mapa.

Se desarrolla el modelo urbano 3D que toma como referencia el procesamiento digital de imágenes para poder quitar ruido que no pertenece al mapa (letras, señales, flechas, entre otros); y posteriormente reconocer donde se encuentran las manzanas urbanas en la distribución del mapa. Al saber donde se encuentran las manzanas en los bordes se genera las estructuras de los edificios en 3D.

Durante el procesamiento de imágenes se utilizan la técnica del Filtro Gaussiano encargada de difuminar los pixeles en la imagen para reducir los defectos, esta técnica toma en cuenta los pixeles vecinos. La otra técnica utilizada es el algoritmo de Canny Edge que detecta los bordes en la imagen para reducir la cantidad de datos a procesar y obtener los límites de la estructura del objeto.

La aplicación se realizó en el IDE Visual Studio 2010 Profesional Express Edition usando lenguaje de programación de C#, empleando las bibliotecas libres de XNA, como API de desarrollo DirectX y las bibliotecas para trabajar imágenes de .NET; con el objetivo de hacer el renderizado del mapa (visualización del mapa y las manzanas en 3D).

## OBJETIVO GENERAL

Sintetiza automáticamente modelos 3D urbanos aproximados, a partir de imágenes tomadas del servicio de Google maps [1][1].

## OBJETIVOS PARTICULARES

- ✓ Reduce el ruido en la imagen de mapa virtual de Google maps utilizando un filtro Gaussiano [2].
- ✓ Detecta los bordes en la imagen mediante el algoritmo de Canny Edge [3].
- ✓ Elimina señales que no pertenecían al mapa.
- ✓ Genera el modelo aproximado 3D de los edificios según su distribución del mapa.

[1] Google es una marca registrada; se reproduce en este documento con el único objetivo de servir a la formación académica.

# INTRODUCCIÓN

La aplicación sintetizará automáticamente modelos 3D urbanos aproximados a partir de imágenes tomadas de Google maps consiste básicamente en los siguientes pasos:

- El usuario de la aplicación previamente guarda la imagen en un formato .jpg para posteriormente cargarla a la aplicación.

- Se carga la imagen guardada a la aplicación, la cual posteriormente saldrá en la aplicación.

- Se convierte la imagen en escala de grises y se aplica el Filtro gaussiano.

- Posteriormente se quita las imágenes no pertenecientes al mapa, deacuerdo a los atributos que se determinan en la base de conocimientos (consiste en atributos de las letras, señales, flechas no pertenecientes al mapa).

- Se detectan los bordes en el mapa con ayuda del algoritmo de Canny Edye, quedando la imagen del mapa binarizada (blanco y negro).

- Determina en donde se encuentran las manzanas urbanas, calles y los parques en la distribución en el mapa.

- En el borde de las manzanas urbanas se colocan los edificios en 3D.

# JUSTIFICACIÓN

La mayoría de las empresas como *Google, Apple* [6]*, Microsoft, Samsung* o *Nokia*[2], hoy en día, implementan en sus dispositivos móviles software de mapas y navegación en sus aplicaciones, e incluso servicios como imágenes de satélite, información de tráfico, de los transportes públicos, entre otros; esto se debe principalmente a la demanda del mercado.

*Google maps* en su versión 5.0 utiliza para la aplicación un "renderizado"[3] dinámico el cual permite mostrar las edificaciones en 3D cuando se hace zoom a nivel de la calle.

Debido a las claras tendencias de las tecnologías presentes en el mercado con respecto al desarrollo del software cartográfico, es conveniente desarrollar una aplicación en este ámbito, demostrando que el alumno está preparado para incorporarse al mercado laboral, siendo este el objetivo último del proyecto terminal, según la legislación universitaria.

La aplicación va dirigida a todos aquellos interesados en conocer una distribución en el mapa; a los cuales sería de utilidad una herramienta capaz de mostrar una aproximación visual 3D de un entorno específico.

Dado que en la mayoría de las aplicaciones existentes no se utilizan filtros gaussianos y el algoritmo de Canny Edge para el procesamiento digital de imágenes se utiliza estos para la obtención de mejores resultados en la calidad de la imagen y detección de bordes respectivamente, y después se realiza el renderizado de las imágenes obtenidas.

[2] Marcas registradas; se reproducen en este documento con el único objetivo de servir a la formación académica.
[3] Proceso de generar una imagen a partir de un modelo.

# TRABAJOS RELACIONADOS

## Referencias Internas

*Software para el cálculo de intensidad de señal en ambiente 3D* [7]. En el proyecto se representa una imagen 3D de señales de frecuencia con base en su propagación, utilizando dos puntos para determinar las alturas. El proyecto utiliza un sólo punto de referencia.

*Algoritmo para acoplamiento automático por similitudes de imágenes digitales* [8]. El proyecto se encarga de acoplar las imágenes de entrada, tomando en cuenta su clasificación. El proyecto actual, además de clasificar los elementos de la imagen de entrada, genera la imagen en 3D de los edificios.

*Sistema de reconocimiento del alfabeto dactológico utilizando procesamiento de imágenes* [9]. El proyecto reconoce objetos determinados por una clasificación previa en la imagen, además de segmentar la imagen, para reconocer el alfabeto dactológico. El proyecto actual se usa para el reconocimiento de los objetos de un mapa.

## Referencias Externas

*Image-based Street-side City Modeling* [10]. El proyecto realiza una segmentación de la imagen para generar imágenes 3D de las calles; la imagen es tomada a nivel del suelo para hacer una imagen realista de la calle. El proyecto actual toma la imagen de un mapa.

*GIMP*[4] [11]. Software dedicado a la edición de imágenes digitales para retoques fotográficos; algunas herramientas con las que cuenta son filtros. El proyecto analiza la imagen de entrada para, posteriormente, transformarla.

*Reconocimiento automático mediante patrones biométricos de huella dactilar* [12]. Este proyecto reconoce patrones en la imagen como el proyecto actual. Sin embargo, el reconocimiento es biométrico (proporcionado por una imagen digital).

[4] GNU Image Manipulation Program.

## DESARROLLO DEL PROYECTO

Se puede observar en la Imagen 1 los pasos fundamentales del procesamiento digital de imágenes [4]  y posteriormente, se hace una breve descripción de cada uno de ellos.

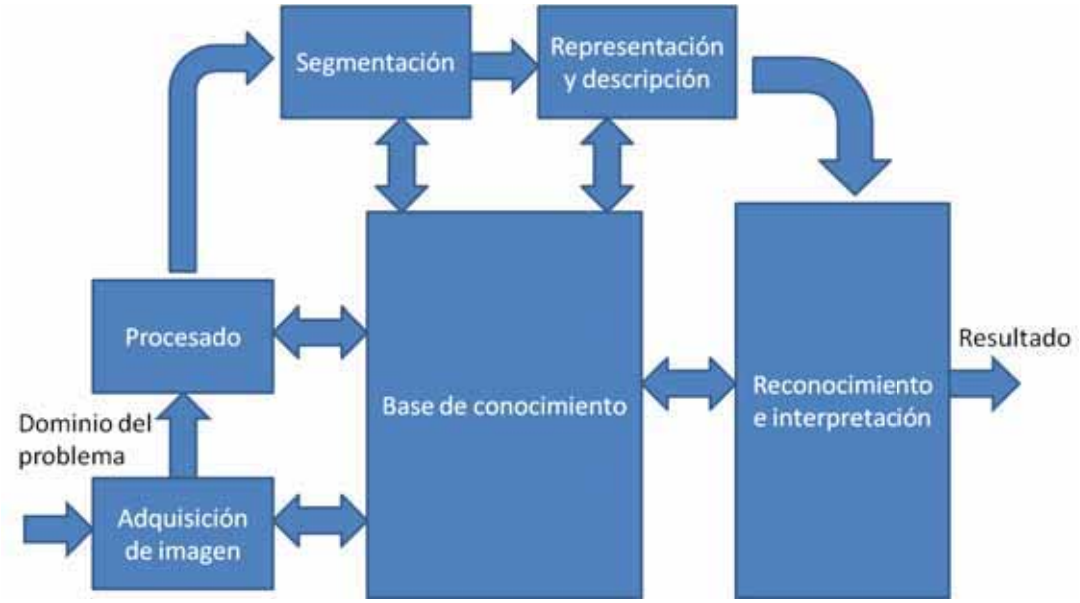



Imagen 1. Etapas fundamentales del procesamiento digital de imágenes [4].

A continuación se describe cada una de las etapas fundamentales del procesamiento digital de imágenes, la especificación técnica y lo que realiza el programa en cada etapa.

### Adquisición de la imagen

En el programa se carga una imagen de Google maps como entrada en formato JPEG de un tamaño máximo de 640x480 pixeles con 24 bits de profundidad (color verdadero). La imagen es guardada previamente en la computadora.

La imagen obtenida de Google maps se toma a una distancia de 100 m de altura y es una imagen tomada del satélite.


Imagen 2. Características del mapa.

Al inicio de la aplicación se tiene la siguiente imagen, en donde el usuario de la aplicación tiene la opción de cargar la imagen en el botón “CARGAR IMAGEN”.


Imagen 3. Inicio de la Aplicación

Al dar click en la el botón de “CARGAR IMAGEN”, da la opción de elegir la imagen del mapa (guardada previamente).

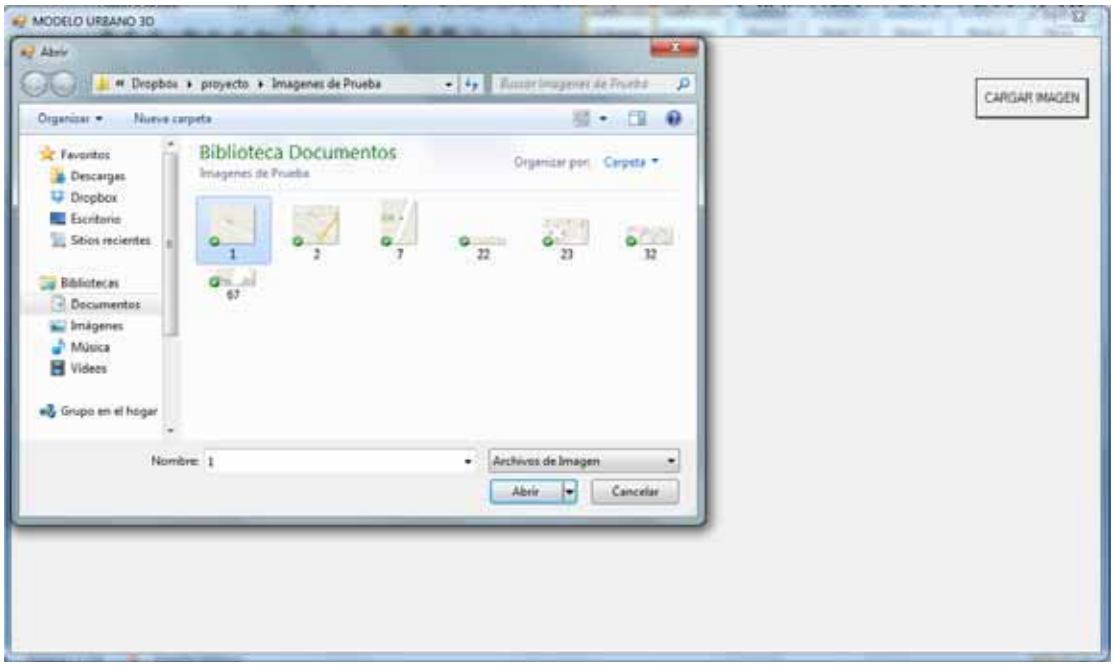



Imagen 4. Elección de la imagen del mapa.

Eligiendo la opción de abrir se carga la imagen a la aplicación, como lo muestra la siguiente imagen.


Imagen 5. Mapa cargado a la aplicación.

## Preprocesamiento

Su función es mejorar la imagen para ser utilizada en etapas posteriores.

En esta etapa se convirtió la imagen a escala de grises, posteriormente se utilizó el filtro gausiano para difuminar los pixeles en la imagen reduciendo los defectos generados por el ruido. Esta técnica aproxima el color del pixel a la distribución gaussiana, tomando en cuenta a los pixeles vecinos. En la siguiente imagen se puede observar lo que se realiza en este módulo.

Imagen 6. Transformación del mapa original aplicándole filtro gaussiano.

La matriz que se utiliza para realizar el filtro gausiano es la siguiente:

$$\frac{1}{306}\begin{bmatrix}1 & 4 & 7 & 4 & 1\\ 4 & 18 & 30 & 18 & 4\\ 7 & 30 & 50 & 30 & 7\\ 4 & 18 & 30 & 18 & 4\\ 1 & 4 & 7 & 4 & 1\end{bmatrix}$$

Posteriormente se quita el ruido que no pertenece al mapa. Esto se logra al consultar la base de conocimientos en la que se encuentran los atributos de las imágenes que se desean quitan para saber si se encuentran en el mapa. Como lo muestra la siguiente imagen:

Imagen 7. Mapa sin señales no pertenecientes al mapa.

Se observa en la imagen anterior que ya no se encuentran las letras, letras y los señalamientos.

El siguiente paso es aplicar el algoritmo de Canny Edge, el cual detecta los bordes de la imagen. Esto se hace para reducir la cantidad de datos a procesar y obtener los límites de la estructura del objeto como se observa en la imagen 8. Obteniendo así una imagen binarizada, es decir, que es representada por dos tonos de color (blanco y negro), donde el fondo es de un solo color y el fenómeno de interés (la manzana urbana, la calle, el área verde) está definido por el conjunto de pixeles en blanco en la imagen resultante.

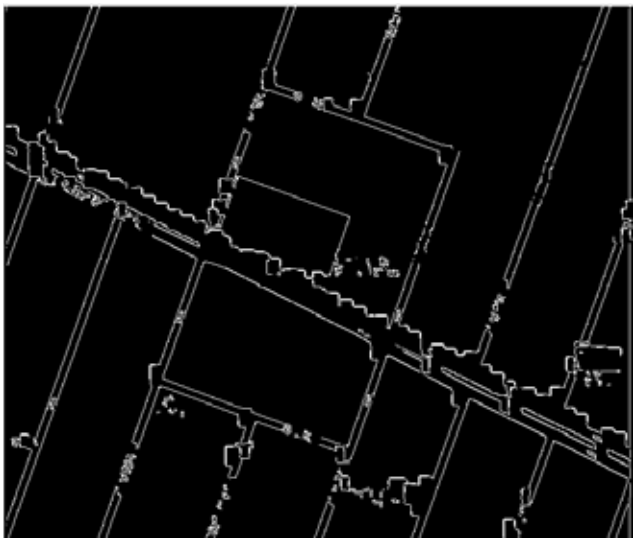


Imagen 8. Aplicación del algoritmo de Canny Edge al mapa (sin señales).

## Representación y descripción

En este proceso se determinan los atributos que diferencian a cada objeto, que pueden ser determinados por su color o forma.

En este proceso se seleccionaron imágenes de las letras, señales, símbolos, flechas que no pertenecen al mapa, de dichas imágenes se tomaron atributos como el ancho, la altura, la mediana, la desviación estándar y la variancia; para lo que se creó un programa aparte llamado “Propiedades” encargado de obtener los atributos de cada imagen.

## Base de conocimientos

Es la descripción del conocimiento adquirido. Ésta se encuentra en una base de datos. Guía a cada proceso a la operación correspondiente, además controla la interacción de cada proceso.

La base de conocimientos fue adquirida gracias al programa de “Propiedades” (mencionado en la etapa anterior) el cual al extraer los atributos de cada imagen lo guardo en un archivo .txt llamado “DatosRuido”.

## Segmentación

Consiste en partir la imagen de entrada en las partes u objetos que la constituyen. Este proceso agrupa pixeles que corresponden al mismo objeto. La salida serán los datos en pixel que constituirán el contorno de la región del objeto.

En la imagen 9 se puede observar cómo se segmento la imagen calles (color rojo) y manzanas urbanas (color verde).

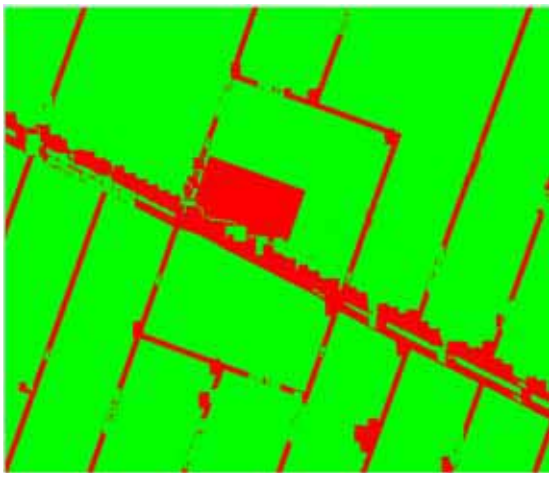


Imagen 9. Segmentación de la imagen en calles y manzanas.

## Reconocimiento e interpretación

El proceso se encarga de clasificar los objetos que tiene la imagen y, posteriormente asignarles un significado al conjunto de elementos reconocidos en la imagen.

La imagen (sus pixeles y sus coordenadas, representan las manzanas urbanas y las calles), el reconocimiento y la interpretación se obtienen en la etapa de preprocesamiento.

En esta etapa se generarán estructuras de diferentes tamaños 3D de los edificios en donde se encuentran las manzanas urbanas, y la calle no tendrá transformación. Como lo muestra el resultado de la aplicación en la imagen 10.

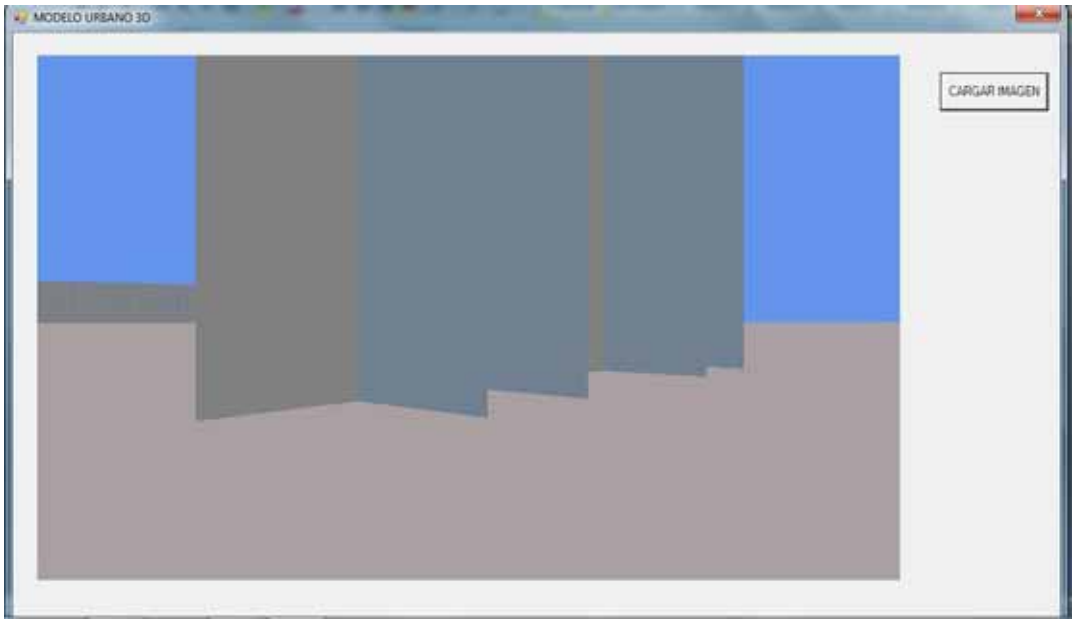



Imagen 10. Modelo urbano 3D (vista 1).

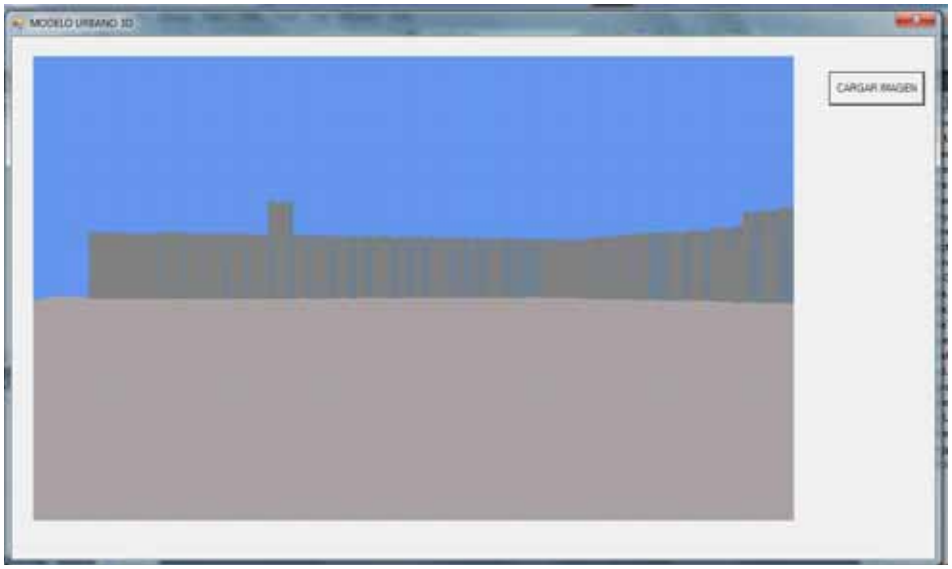



Imagen 11. Modelo urbano 3D (vista 2).

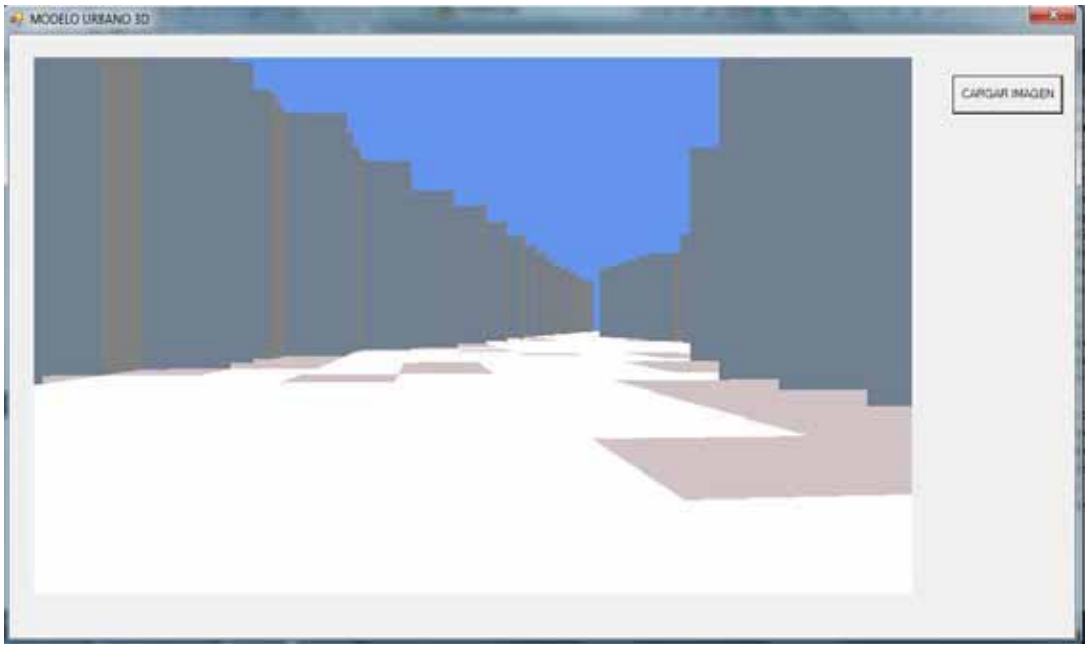
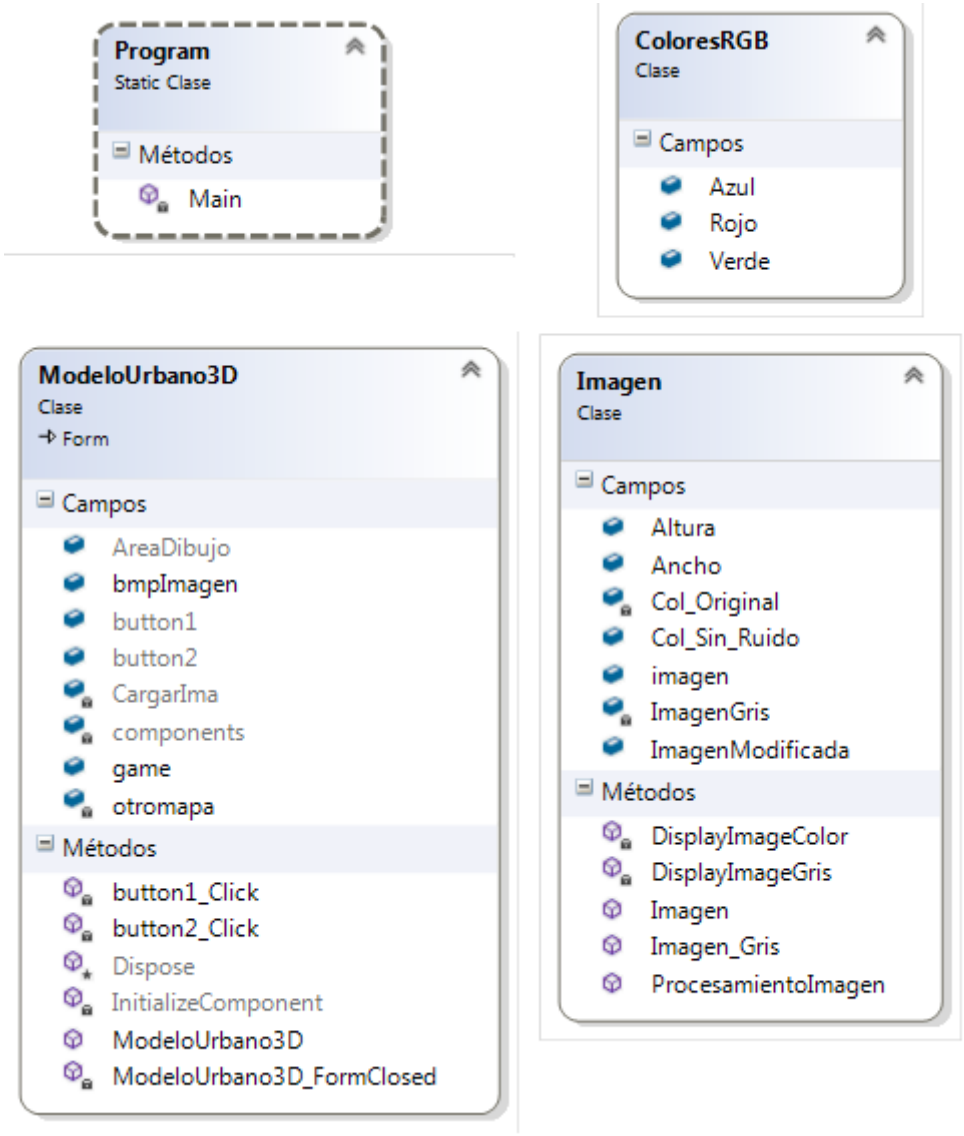
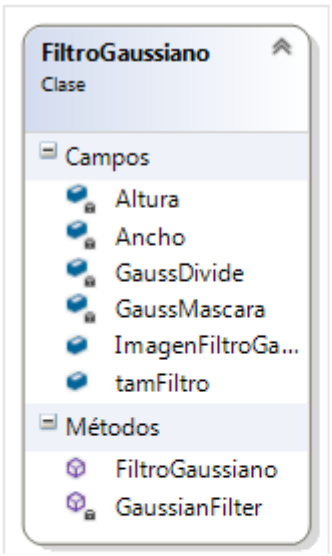


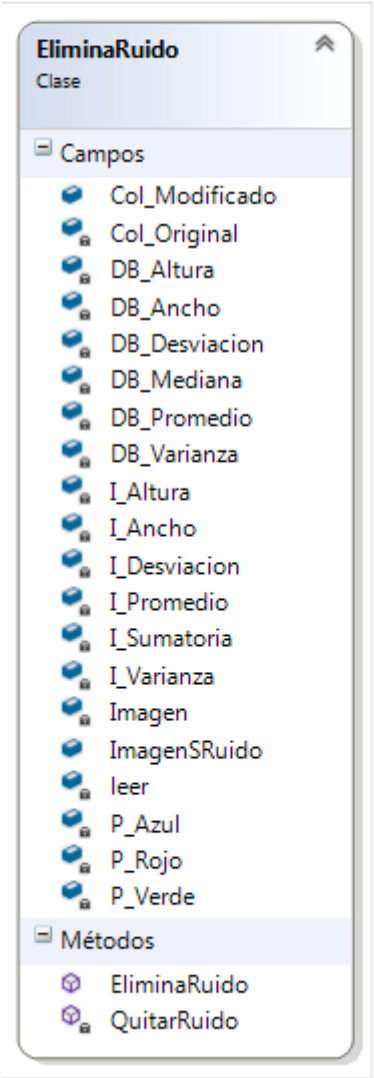

Imagen 12. Modelo urbano 3D (vista 3).

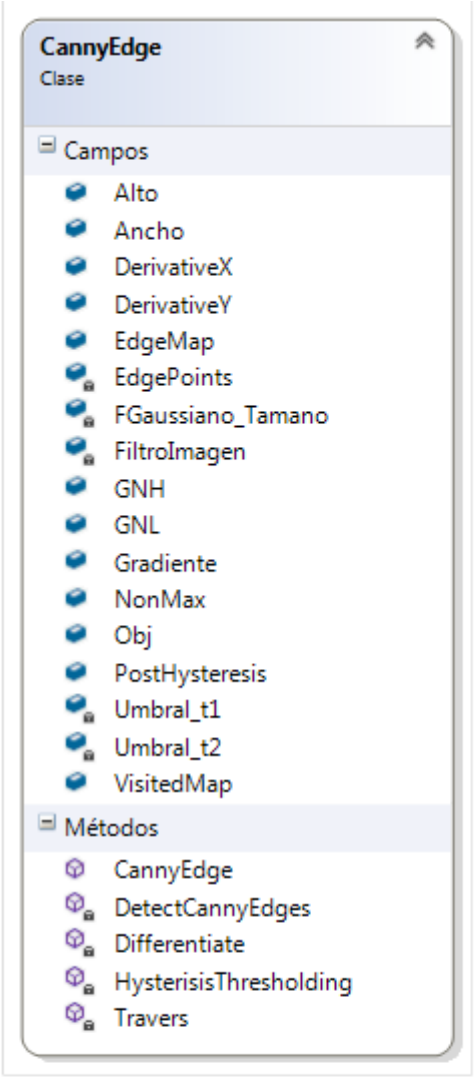
Para poderse mover dentro del modelo urbano es a través de las teclas del movimiento del cursor permitiendo mover al usuario hacia delante, atrás, derecha e izquierda, sin embargo no se podrá atravesar los edificios.

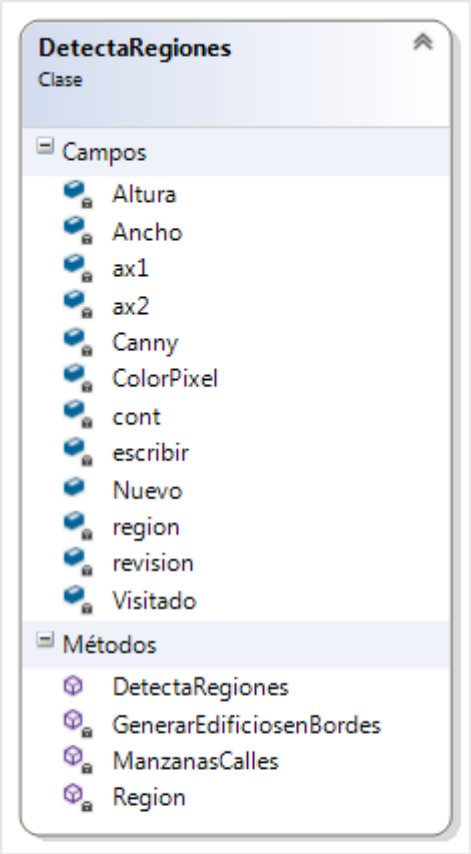
## DIAGRAMA DE CLASES

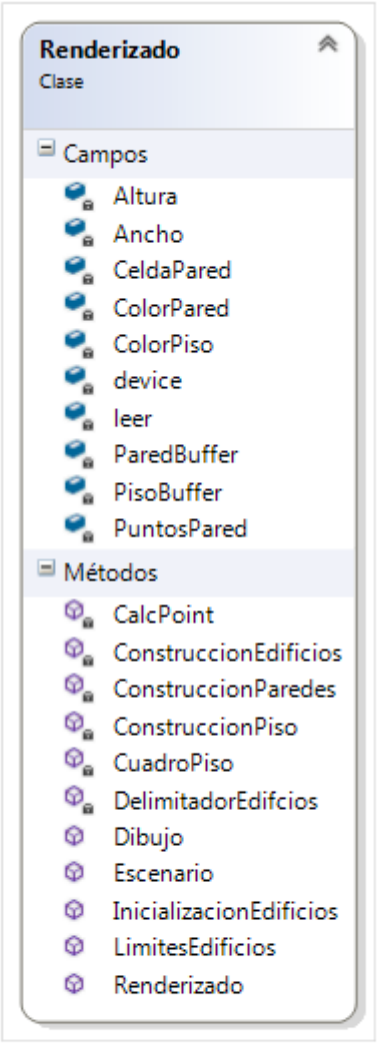


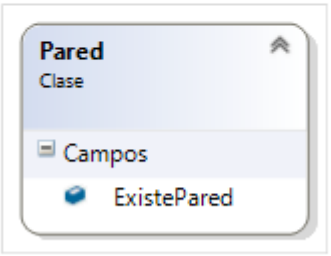

FiltroGaussiano
Clase
Campos
Altura
Ancho
GaussDivide
GaussMascara
ImagenFiltroGa...
tamFiltro
Métodos
FiltroGaussiano
GaussianFilter

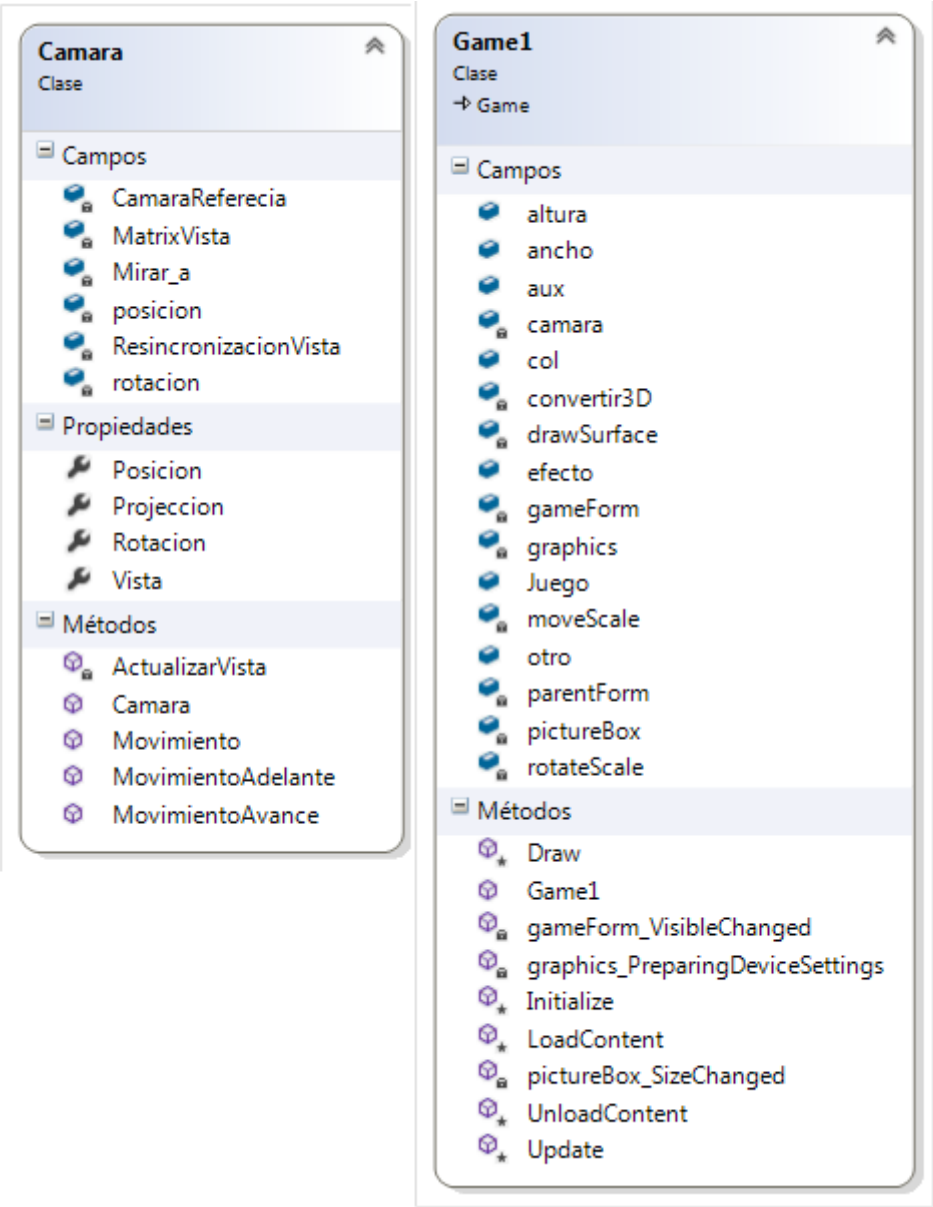

EliminaRuido
Clase
Campos
Col_Modificado
Col_Original
DB_Altura
DB_Ancho
DB_Desviacion
DB_Mediana
DB_Promedio
DB_Varianza
I_Altura
I_Ancho
I_Desviacion
I_Promedio
I_Sumatoria
I_Varianza
Imagen
ImagenSRuido
leer
P_Azul
P_Rojo
P_Verde
Métodos
EliminaRuido
QuitarRuido


CannyEdge
Clase
Campos
Alto
Ancho
DerivativeX
DerivativeY
EdgeMap
EdgePoints
FGaussiano_Tamano
FiltroImagen
GNH
GNL
Gradiente
NonMax
Obj
PostHysteresis
Umbral_t1
Umbral_t2
VisitedMap
Métodos
CannyEdge
DetectCannyEdges
Differentiate
HysterisisThresholding
Travers


DetectaRegiones
Clase
Campos
Altura
Ancho
ax1
ax2
Canny
ColorPixel
cont
escribir
Nuevo
region
revision
Visitado
Métodos
DetectaRegiones
GenerarEdificiosenBordes
ManzanasCalles
Region


Renderizado
Clase
Campos
Altura
Ancho
CeldaPared
ColorPared
ColorPiso
device
leer
ParedBuffer
PisoBuffer
PuntosPared
Métodos
CalcPoint
ConstruccionEdificios
ConstruccionParedes
ConstruccionPiso
CuadroPiso
DelimitadorEdifcios
Dibujo
Escenario
InicializacionEdificios
LimitesEdificios
Renderizado


Pared
Clase
Campos
ExistePared


Camara
Clase
Campos
CamaraReferecia
MatrixVista
Mirar_a
posicion
ResincronizacionVista
rotacion
Propiedades
Posicion
Projeccion
Rotacion
Vista
Métodos
ActualizarVista
Camara
Movimiento
MovimientoAdelante
MovimientoAvance
Game1
Clase
Game
Campos
altura
ancho
aux
camara
col
convertir3D
drawSurface
efecto
gameForm
graphics
Juego
moveScale
otro
parentForm
pictureBox
rotateScale
Métodos
Draw
Game1
gameForm_VisibleChanged
graphics_PreparingDeviceSettings
Initialize
LoadContent
pictureBox_SizeChanged
UnloadContent
Update

## CONCLUSIONES

En la realización del proyecto me pude dar cuenta de que se trata el procesamiento digital de imágenes siguiendo cada una de las etapas.

Durante la realización del proyecto se tuvieron algunas dificultades, las cuales algunas se pudieron resolver y otras no. Los problemas que no se pudieron resolver fueron al tratar de quitar las imágenes no pertenecientes al mapa (letras, señales, flechas), debido a que la aplicación no las quita todas, sin embargo el margen de error en esta parte es muy bajo. Si bien hay una solución para los problemas que no se pudieron resolver, no se pudieron lograr por cuestiones de tiempo, pero se pueden atajar como trabajo a futuro.

Debido a que no se quita todo lo que no pertenece al mapa en etapas posteriores se va arrastrando los errores haciendo que la imagen original del mapa se distorsione un poco generando ruido, es decir que en partes que se debería tomar como manzana urbana se toma como calle o viceversa. Sin embargo, esto solo sucede cuando las calles están muy pegadas una de la otra (en la imagen, no en la realidad), en el caso que están separadas la aplicación mejora los resultados.

Además otro problema que no se resolvió en su totalidad es que debido a que los pixeles que se encuentran dentro de una región no son del mismo color, es decir que una manzana urbana que a simple vista parece de un tono de gris al analizar los pixeles son de diferentes tonalidades de gris; por lo que al interpretar la imagen algunos pixeles quedan en una región diferente a la que pertenecen.

Debido a los inconvenientes hay edificios que no debería ir en ese espacio en el mapa.

A pesar del problema antes mencionado se lograró cumplir todos los objetivos planeados en la propuesta original; reduciendo el ruido en la imagen de mapa virtual de *Google maps* utilizando el filtro gaussiano, detectar los bordes en la imagen mediante el algoritmo de *Canny Edge,* eliminar señales que no pertenecen al mapa (esto se logro en el caso de que las calles no estén muy pegadas entre sí) y se generó el modelo aproximado 3D de los edificios según su distribución en el mapa.

## TRABAJOS A FUTURO

- Mejorar el análisis de señales no pertenecientes al mapa.
- Lograr tener pixeles de una sola tonalidad de acuerdo a la región a la que pertenecen.
- El mapa en 3D podría ser utilizado en el caso de existir inundaciones en la ciudad en donde correría en agua atreves de las calles.

# APÉNDICE DE CÓDIGO FUENTE

## Clase Imagen

### *Atributos públicos*

- int Altura
- int Ancho
- Bitmap imagen
- Bitmap ImagenModificada
- ColoresRGB[,] Col_Original
- ColoresRGB[,] Col_Sin_Ruido

### *Atributo privado*

- int[,] ImagenGris

### *Costructor*

```
public Imagen(Bitmap Objeto)
    {
      imagen = Objeto;
      Altura = imagen.Height;
      Ancho = imagen.Width;
      ImagenGris = new int[Ancho, Altura];

      ImagenModificada = new Bitmap(Ancho, Altura);

      Col_Original = new ColoresRGB[Ancho, Altura];
      Col_Sin_Ruido = new ColoresRGB[Ancho, Altura];

      if (File.Exists("Regiones.txt"))
      {
        File.Delete("Regiones.txt");
      }
      ProcesamientoImagen();
    }
```

### *Procesamiento de la imagen*

```
public void ProcesamientoImagen()
    {
      Imagen_Gris();

      FiltroGaussiano FiltroGauss = new FiltroGaussiano(ImagenGris, Ancho, Altura);
      EliminaRuido ImagenSinRuido = new EliminaRuido(FiltroGauss.ImagenFiltroGauss,
Ancho, Altura, Col_Original);
      CannyEdge Canny = new CannyEdge(ImagenSinRuido.ImagenSRuido,
FiltroGauss.tamFiltro, Ancho, Altura);
```

```
        DetectaRegiones Region = new
DetectaRegiones(ImagenSinRuido.Col_Modificado, Canny.EdgeMap);
        Col_Sin_Ruido = Region.Nuevo;
        DisplayImageColor(Region.Nuevo);
    }
```

### *Convertir la Imagen a Gris*

```
public void Imagen_Gris()
    {
        Color ColorDelPixel;
        int colorGris;

        for (int x = 0; x < Ancho; x++)
        {
            for (int y = 0; y < Altura; y++)
            {
                Col_Sin_Ruido[x, y] = new ColoresRGB();
                Col_Original[x, y] = new ColoresRGB();

            }
        }

        for (int x = 0; x < Ancho; x++)
        {
            for (int y = 0; y < Altura; y++)
            {
                ColorDelPixel = imagen.GetPixel(x, y);

                Col_Original[x, y].Rojo = Col_Sin_Ruido[x, y].Rojo = ColorDelPixel.R;
                Col_Original[x, y].Verde = Col_Sin_Ruido[x, y].Verde = ColorDelPixel.G;
                Col_Original[x, y].Azul = Col_Sin_Ruido[x, y].Azul = ColorDelPixel.B;
                colorGris = (ColorDelPixel.R + ColorDelPixel.G + ColorDelPixel.B) / 3;
                ImagenGris[x, y] = colorGris;

            }
        }
    }
```

### *Convierte la matriz imagen  a bitmap a gris*

```
private void DisplayImageGris(int[,] Imagen)
    {
        Color ColNuevo;

        for (int i = 0; i < Ancho; i++)
        {
            for (int j = 0; j < Altura; j++)
            {
```

```
            int aux = (int)(Imagen[i, j]);
            if (aux > 255)
               aux = 255;
            ColNuevo = Color.FromArgb(aux, aux, aux);
            ImagenModificada.SetPixel(i, j, ColNuevo);
         }
      }
   }
```

### *Convierte la matriz imagen  a bitmap a color*

```
   private void DisplayImageColor(ColoresRGB[,] Col)
   {
      Color ColNuevo;
      int Rojo, Verde, Azul;

      for (int i = 0; i < Ancho; i++)
      {
         for (int j = 0; j < Altura; j++)
         {
            Rojo = Col[i, j].Rojo;
            Verde = Col[i, j].Verde;
            Azul = Col[i, j].Azul;

            if (Rojo > 255)
               Rojo = 255;
            if (Verde > 255)
               Verde = 255;
            if (Azul > 255)
               Azul = 255;
            ColNuevo = Color.FromArgb(Rojo, Verde, Azul);
            ImagenModificada.SetPixel(i, j, ColNuevo);
         }
      }
   }
```

## Clase filtro gaussiano

### *Atributo público*

- int[,] ImagenFiltroGauss

### *Atributos privados*

- int[,] GaussMascara
- int tamFiltro
- int GaussDivide
- int Ancho, Altura

### *Constructor*

```
public FiltroGaussiano(int[,] ImagenGris, int An, int Al)
{
   Ancho = An;
   Altura = Al;
   ImagenFiltroGauss = new int[Ancho, Altura];
   GaussianFilter(ImagenGris);
}
```

### *Filtro Gaussiano*

```
private int[,] GaussianFilter(int[,] Data)
{
   float Sum = 0;
   ImagenFiltroGauss = Data;
   for (int i = 2; i <= ((Ancho - 1) - 2); i++)
   {
      for (int j = 2; j <= ((Altura - 1) - 2); j++)
      {
         Sum = 0;
         for (int k = -2; k <= 2; k++)
         {
            for (int l = -2; l <= 2; l++)
            {
               Sum = Sum + ((float)Data[i + k, j + l] * GaussMascara[2 + k, 2 + l]);
            }
         }
         ImagenFiltroGauss[i, j] = (int)(Math.Round(Sum / (float)GaussDivide));
      }
   }
   return ImagenFiltroGauss;
}
```

## Eliminar ruido

### *Atributos públicos*

- int[,] ImagenSRuido
- ColoresRGB[,]  Col_Modificado

### *Atributos privados*

- ColoresRGB[,] Col_Original
- StreamReader leer
- int DB_Ancho, DB_Altura, DB_Mediana

- int[,] Imagen
- int I_Ancho, I_Altura
- float DB_Promedio, DB_Varianza, DB_Desviacion
- float I_Varianza, I_Desviacion
- float I_Promedio = 0
- float I_Sumatoria = 0

### *Constructor*

```
public EliminaRuido(int[,] Imagen, int I_Ancho, int I_Altura, ColoresRGB[,] Col_Original)
    {
      this.Col_Original = Col_Original;
      Col_Modificado = Col_Original;
      this.Imagen = Imagen;
      this.I_Ancho = I_Ancho;
      this.I_Altura = I_Altura;
      P_Rojo = P_Verde = P_Azul = 0.0f;
      leer = new StreamReader("DatosRuido.txt");
      QuitarRuido();
    }
```

### *Elimina el ruido*

```
    void QuitarRuido()
    {
      while (!leer.EndOfStream)
      {
        string[] palabras = leer.ReadLine().Split(' ');
        DB_Ancho = Convert.ToInt16(palabras[1]);
        DB_Altura = Convert.ToInt16(palabras[2]);
        DB_Promedio = (float)Convert.ToDouble(palabras[3]);
        DB_Desviacion = (float)Convert.ToDouble(palabras[4]);
        DB_Varianza = (float)Convert.ToDouble(palabras[5]);
        DB_Mediana = Convert.ToInt16(palabras[6]);

        int[] I_num_ord = new int[DB_Ancho * DB_Altura];

        for (i = 0; i <= I_Ancho - DB_Ancho; i++)
        {
          for (j = 0; j <= I_Altura - DB_Altura; j++)
          {
            I_Promedio = Imagen[i, j];
            I_Sumatoria = 0;
```

```
                int total = 0;
                for (x = i; x < i + DB_Ancho; x++)
                {
                   for (l = j; l < j + DB_Altura; l++)
                   {
                      I_Promedio = (I_Promedio + Imagen[x, l]) / 2;
                      I_num_ord[total] = Imagen[x, l];
                      total++;
                   }
                }

                //Ordena los numeros
                Array.Sort(I_num_ord);

                int I_Mediana = I_num_ord[total / 2];

                for (int p = 0; p < total; p++)
                {
                   I_Sumatoria = (float)(Math.Pow(I_num_ord[p] - I_Promedio, 2) +
I_Sumatoria);
                }

                I_Varianza = I_Sumatoria / (total - 1);
                I_Desviacion = (float)(Math.Sqrt(I_Varianza));
                double distancia = 0;

                distancia = Math.Sqrt(Math.Pow(DB_Promedio - Math.Round(I_Promedio,
                2), 2) + Math.Pow(DB_Mediana - I_Mediana, 2) +
                   Math.Pow(DB_Varianza - Math.Round(I_Varianza, 2), 2) +
                Math.Pow(DB_Desviacion - Math.Round(I_Desviacion, 2), 2));

               //Comprueba si la imagen es parecida a la de la base de datos (para
               posteriormente              //quitarla)
                if (distancia < 10)
                {
                   for (x = i; x < i + DB_Ancho; x++)
                   {

                      for (l = j; l < j + DB_Altura; l++)
                      {
                         I_Promedio = Imagen[x, l];
                         for (int o = x; o < i + DB_Ancho && o <= I_Ancho; o++)
```

```
                {
                    for (int p = l; p < j + DB_Altura && p < l_Altura; p++)
                    {
                        I_Promedio = (I_Promedio + Imagen[o, p]) / 2;
                        P_Rojo = (P_Rojo + Col_Original[o, p].Rojo) / 2;
                        P_Verde = (P_Verde + Col_Original[o, p].Verde) / 2;
                        P_Azul = (P_Azul + Col_Original[o, p].Azul) / 2;

                    }
                }
                Imagen[x, l] = Convert.ToInt16(I_Promedio);
                Col_Modificado[x, l].Rojo = Convert.ToInt16(P_Rojo);
                Col_Modificado[x, l].Verde = Convert.ToInt16(P_Verde);
                Col_Modificado[x, l].Azul = Convert.ToInt16(P_Azul);
            }
        }
    }
  }
 }
 leer.Close();
 ImagenSRuido = Imagen;
}
```

## Canny Edge

### *Atributos públicos*

- int Ancho, Alto
- Bitmap Obj
- float[,] NonMax
- float[,] Gradiente
- int[,] PostHysteresis
- float[,] GNH
- float[,] GNL
- int[,] EdgeMap
- int[,] VisitedMap

### *Atributos privados*

- int FGaussiano_Tamano
- int[,] FiltroImagen
- float Umbral_t1 = 40, Umbral_t2 = 45

- float[,] DerivativeX
- float[,] DerivativeY
- int[,] EdgePoints

### *Constructor*

```
public CannyEdge(int[,] Imagen,int Tam_Filtro, int An, int Al)
{
   Ancho = An;
   Alto = Al;
   //Imagen con Filtro Gaussiano
   FiltroImagen = Imagen;
   FGaussiano_Tamano = Tam_Filtro;

   EdgeMap = new int[Ancho, Alto];
   VisitedMap = new int[Ancho, Alto];
   Gradiente = new float[Ancho, Alto];
   NonMax = new float[Ancho, Alto];
   PostHysteresis = new int[Ancho, Alto];
   DerivativeX = new float[Ancho, Alto];
   DerivativeY = new float[Ancho, Alto];
   DetectCannyEdges();
   return;
}
```

### *Canny edge*

```
private void DetectCannyEdges()
{
   //Mascara de Prewit
   int[,] Dx = {{1,0,-1},
            {1,0,-1},
            {1,0,-1}};

   int[,] Dy = {{1,1,1},
            {0,0,0},
            {-1,-1,-1}};
   DerivativeX = Differentiate(FiltroImagen, Dx);
   DerivativeY = Differentiate(FiltroImagen, Dy);

   int i, j;

   for (i = 0; i <= (Ancho - 1); i++)
```

```
        {
            for (j = 0; j <= (Alto - 1); j++)
            {
                Gradiente[i, j] = (float)Math.Sqrt((DerivativeX[i, j] * DerivativeX[i, j]) +
(DerivativeY[i, j] * DerivativeY[i, j]));

            }

        }

        for (i = 0; i <= (Ancho - 1); i++)
        {
            for (j = 0; j <= (Alto - 1); j++)
            {
                NonMax[i, j] = Gradiente[i, j];
            }
        }

        int Limit = FGaussiano_Tamano / 2;
        int r, c;
        float Tangent;
        for (i = Limit; i <= (Ancho - Limit) - 1; i++)
        {
            for (j = Limit; j <= (Alto - Limit) - 1; j++)
            {

                if (DerivativeX[i, j] == 0)
                    Tangent = 90F;
                else
                    Tangent = (float)(Math.Atan(DerivativeY[i, j] / DerivativeX[i, j]) * 180 /
Math.PI);

                //Horizontal Edge
                if (((-22.5 < Tangent) && (Tangent <= 22.5)) || ((157.5 < Tangent) &&
(Tangent <= -157.5)))
                {
                    if ((Gradiente[i, j] < Gradiente[i, j + 1]) || (Gradiente[i, j] < Gradiente[i, j -
1]))
                        NonMax[i, j] = 0;
                }
```

```
            //Vertical Edge
            if (((-112.5 < Tangent) && (Tangent <= -67.5)) || ((67.5 < Tangent) &&
(Tangent <= 112.5)))
            {
               if ((Gradiente[i, j] < Gradiente[i + 1, j]) || (Gradiente[i, j] < Gradiente[i - 1,
j]))
                  NonMax[i, j] = 0;
            }

            //+45 Degree Edge
            if (((-67.5 < Tangent) && (Tangent <= -22.5)) || ((112.5 < Tangent) &&
(Tangent <= 157.5)))
            {
               if ((Gradiente[i, j] < Gradiente[i + 1, j - 1]) || (Gradiente[i, j] < Gradiente[i -
1, j + 1]))
                  NonMax[i, j] = 0;
            }

            //-45 Degree Edge
            if (((-157.5 < Tangent) && (Tangent <= -112.5)) || ((67.5 < Tangent) &&
(Tangent <= 22.5)))
            {
               if ((Gradiente[i, j] < Gradiente[i + 1, j + 1]) || (Gradiente[i, j] < Gradiente[i -
1, j - 1]))
                  NonMax[i, j] = 0;
            }

         }
      }


      //PostHysteresis = NonMax;
      for (r = Limit; r <= (Ancho - Limit) - 1; r++)
      {
         for (c = Limit; c <= (Alto - Limit) - 1; c++)
         {

            PostHysteresis[r, c] = (int)NonMax[r, c];
         }

      }
```

```
float min, max;
min = 100;
max = 0;
for (r = Limit; r <= (Ancho - Limit) - 1; r++)
   for (c = Limit; c <= (Alto - Limit) - 1; c++)
   {
      if (PostHysteresis[r, c] > max)
      {
         max = PostHysteresis[r, c];
      }

      if ((PostHysteresis[r, c] < min)&&(PostHysteresis[r, c] >0))
      {
         min = PostHysteresis[r, c];
      }
   }

GNH = new float[Ancho, Alto];
GNL = new float[Ancho, Alto]; ;
EdgePoints = new int[Ancho, Alto];

for (r = Limit; r <= (Ancho - Limit) - 1; r++)
{
   for (c = Limit; c <= (Alto - Limit) - 1; c++)
   {
      if (PostHysteresis[r, c] >= Umbral_t2)
      {
         EdgePoints[r, c] = 1;
         GNH[r, c] = 255;
      }
      if ((PostHysteresis[r, c] < Umbral_t2) && (PostHysteresis[r, c] >= Umbral_t1))
      {
         EdgePoints[r, c] = 2;
         GNL[r, c] = 255;
      }
   }
}

HysterisisThresholding(EdgePoints);

for (i = 0; i <= (Ancho - 1); i++)
   for (j = 0; j <= (Alto - 1); j++)
```

```
        {
            EdgeMap[i, j] = EdgeMap[i, j]*255;
        }

    return;
}
```

## Detecta Regiones

### *Atributos privados*

- bool[,] Visitado
- int[,] Canny
- ColoresRGB[,] ColorPixel
- int[,] region
- int cont = 2
- int revision = 0
- StreamWriter escribir = new StreamWriter("Regiones.txt", true)

### *Atributos publicos*

- int Ancho
- int Altura
- ColoresRGB[,] Nuevo

### *Costructor*

```
public DetectaRegiones(ColoresRGB[,] ColorPixel, int[,] Canny)
{
    Ancho  = ColorPixel.GetLength(0);
    Altura = ColorPixel.GetLength(1);
    this.ColorPixel = ColorPixel;
    this.Canny = Canny;
    Nuevo = new ColoresRGB[Ancho, Altura];
    Visitado = new bool[Ancho, Altura];
    region =  new int[Ancho, Altura];

    for (int x = 0; x < Ancho; x++)
    {
        for (int y = 0; y < Altura; y++)
        {
            Nuevo[x, y] = new ColoresRGB();
            Visitado[x, y] = false;
            region[x, y] = 0;
```

```
            }
        }
        ManzanasCalles();
        GenerarEdificiosenBordes();
    }
```

### *ManzanasCalles*

```
    void ManzanasCalles()
    {
        for (int y = 0; y < Altura; y++)
        {
            for (int x = 0; x < Ancho; x++)
            {
                if (Canny[x, y] != 255)
                {
                    if (ColorPixel[x, y].Rojo > 250 && ColorPixel[x, y].Verde > 250 &&
ColorPixel[x, y].Azul > 250)
                        Nuevo[x, y].Rojo = 255;
                    else if (Math.Sqrt(Math.Pow(ColorPixel[x, y].Rojo - 252.70, 2) +
Math.Pow(ColorPixel[x, y].Verde - 252, 2) + Math.Pow(ColorPixel[x, y].Azul - 146.5, 2)) <
15)
                        Nuevo[x, y].Rojo = 255;
                    else if (Math.Sqrt(Math.Pow(ColorPixel[x, y].Rojo - 250.25, 2) +
Math.Pow(ColorPixel[x, y].Verde - 194.98, 2) + Math.Pow(ColorPixel[x, y].Azul - 75.62,
2)) < 15)
                    {
                        Nuevo[x, y].Rojo = 255;
                        Nuevo[x, y].Verde = 255;
                    }
                    else if (Math.Sqrt(Math.Pow(ColorPixel[x, y].Rojo - 203, 2) +
Math.Pow(ColorPixel[x, y].Verde - 223, 2) + Math.Pow(ColorPixel[x, y].Azul - 170, 2)) <
15)
                        Nuevo[x, y].Rojo = 255;
                    else
                        Nuevo[x, y].Verde = 255;
                }
                else
                {
                    Nuevo[x, y].Rojo = 255;
                    Nuevo[x, y].Verde = 255;
                    Nuevo[x, y].Azul = 255;
                }
```

```
            }
        }
    }
```

### *Generar Edificios en bordes*

```
    void GenerarEdificiosenBordes()
    {
        Random z = new Random();
        for (int y = 0; y < Altura; y++)
        {
            for (int x = 0; x < Ancho; x++)
            {
                if (Nuevo[x, y].Rojo == 255 && Nuevo[x, y].Verde == 255 && Nuevo[x, y].Azul
== 255 && !Visitado[x, y])
                {
                    Visitado[x, y] = true;
                    Nuevo[x, y].Rojo = 0;
                    Nuevo[x, y].Azul = 0;
                    //Altura de los edificios
                    int z1 = z.Next(1, 5);
                    escribir.WriteLine(x + " " + z1 + " " + y);
                    Region(x, y, z1);
                }
            }
        }
        escribir.Close();

        for (int y = 0; y < Altura; y++)
        {
            for (int x = 0; x < Ancho; x++)
            {
                if (Nuevo[x, y].Rojo == 255 && Nuevo[x, y].Verde == 0 && Nuevo[x, y].Azul ==
0)
                {
                    Nuevo[x, y].Rojo = 255;
                    Nuevo[x, y].Verde = 255;
                    Nuevo[x, y].Azul = 255;

                }
                else
```

```
                if (Nuevo[x, y].Rojo == 0 && Nuevo[x, y].Verde == 255 && Nuevo[x, y].Azul
== 0)
                {
                    Nuevo[x, y].Rojo = 209;
                    Nuevo[x, y].Verde = 195;
                    Nuevo[x, y].Azul = 195;
                }

            }
        }
    }

    int ax1, ax2;
    void Region(int x, int y, int z)
    {
        switch (revision)
        {
            case 0:
                if (x + 1 < Ancho)
                    if (Nuevo[x + 1, y].Rojo == 255 && Nuevo[x + 1, y].Verde == 255 &&
Nuevo[x + 1, y].Azul == 255 && !Visitado[x+1, y])
                    {
                        revision = 0;
                        ax1=x+1;
                        Visitado[ax1, y] = true;
                        Nuevo[ax1, y].Rojo = 0;
                        Nuevo[ax1, y].Azul = 0;
                        escribir.WriteLine(ax1 + " " + z + " " + y);
                        Region(x + 1, y, z);
                    }
                    else
                    {
                        revision = 1;
                        Region(x, y, z);
                    }
                else
                {
                    revision = 2;
                    Region(x, y, z);
                }
                break;
            case 1:
```

```
            if (x + 1 < Ancho && y + 1 < Altura)
               if (Nuevo[x + 1, y + 1].Rojo == 255 && Nuevo[x + 1, y + 1].Verde == 255 &&
Nuevo[x + 1, y + 1].Azul == 255 && !Visitado[x+1, y+1])
               {
                  ax1 = x + 1;
                  ax2 = y +1;
                  Nuevo[ax1, ax2].Rojo = 0;
                  Nuevo[ax1, ax2].Azul = 0;
                  Visitado[ax1, ax2] = true;
                  revision = 0;
                  escribir.WriteLine(ax1+ " " + (z) + " " + ax2);
                  Region(ax1, ax2, z);
               }
               else
               {
                  revision = 2;
                  Region(x, y, z);
               }
            else
            {
               revision = 2;
               Region(x, y, z);
            }
            break;
         case 2:
            if (y + 1 < Altura)
               if (Nuevo[x, y + 1].Rojo == 255 && Nuevo[x, y + 1].Verde == 255 &&
Nuevo[x, y + 1].Azul == 255 && !Visitado[x, y+1])
               {
                  ax1 = x;
                  ax2 = y + 1;
                  Nuevo[ax1, ax2].Rojo = 0;
                  Nuevo[ax1, ax2].Azul = 0;
                  Visitado[ax1, ax2] = true;
                  revision = 0;
                  escribir.WriteLine(ax1 + " " + (z) + " " + ax2);
                  Region(ax1, ax2, z);
               }
               else
               {
                  revision = 3;
                  Region(x, y, z);
```

```
                }
            else
            {
                revision = 3;
                Region(x, y, z);
            }
            break;
        case 3:
            if (x - 1 > 0 && y + 1 < Altura)
                if (Nuevo[x - 1, y + 1].Rojo == 255 && Nuevo[x - 1, y + 1].Verde == 255 &&
Nuevo[x - 1, y + 1].Azul == 255 && !Visitado[x-1, y+1])
                {
                    ax1 = x - 1;
                    ax2 = y + 1;
                    Nuevo[ax1, ax2].Rojo = 0;
                    Nuevo[ax1, ax2].Azul = 0;
                    Visitado[ax1, ax2] = true;
                    revision = 0;
                    escribir.WriteLine(ax1 + " " + (z) + " " + ax2);
                    Region(ax1, ax2, z);

                }
                else
                {
                    revision = 4;
                    Region(x, y, z);
                }
            else
            {
                revision = 4;
                Region(x, y, z);
            }
            break;
        case 4:
            if (x - 1 > 0)
                if (Nuevo[x - 1, y].Rojo == 255 && Nuevo[x - 1, y].Verde == 255 && Nuevo[x
- 1, y].Azul == 255 && !Visitado[x-1, y])
                {
                    ax1 = x - 1;
                    ax2 = y;
                    Visitado[ax1, ax2] = true;
                    revision = 0;
```

```
                Nuevo[ax1, ax2].Rojo = 0;
                Nuevo[ax1, ax2].Azul = 0;
                escribir.WriteLine(ax1 + " " + (z) + " " + ax2);
                Region(ax1, ax2, z);
              }
              else
              {
                revision = 5;
                Region(x, y, z);
              }
            else
            {
              revision = 5;
              Region(x, y, z);
            }
            break;
          case 5:
            if (x - 1 > 0 && y - 1 > 0)
              if (Nuevo[x - 1, y - 1].Rojo == 255 && Nuevo[x - 1, y - 1].Verde == 255 &&
Nuevo[x - 1, y - 1].Azul == 255 && !Visitado[x-1, y-1])
              {
                ax1 = x - 1;
                ax2 = y - 1;
                Visitado[ax1, ax2] = true;
                Nuevo[ax1, ax2].Rojo = 0;
                Nuevo[ax1, ax2].Azul = 0;
                revision = 0;
                escribir.WriteLine(ax1 + " " + (z) + " " + ax2);
                Region(ax1, ax2, z);
              }
              else
              {
                revision = 6;
                Region(x, y, z);
              }
            else
            {
              revision = 6;
              Region(x, y, z);
            }
            break;
          case 6:
```

```
            if (y - 1 > 0)
              if (Nuevo[x, y - 1].Rojo == 255 && Nuevo[x, y - 1].Verde == 255 &&
Nuevo[x, y - 1].Azul == 255 && !Visitado[x, y-1])
              {
                ax1 = x;
                ax2 = y - 1;
                Visitado[ax1, ax2] = true;
                revision = 0;
                Nuevo[ax1, ax2].Rojo = 0;
                Nuevo[ax1, ax2].Azul = 0;
                escribir.WriteLine(ax1 + " " + (z) + " " + ax2);
                Region(ax1, ax2, z);
              }
              else
              {
                revision = 7;
                Region(x, y, z);
              }
            else
            {
              revision = 7;
              Region(x, y, z);
            }
            break;
          default:
            if (x - 1 > 0)
              if (Nuevo[x - 1, y].Rojo == 255 && Nuevo[x - 1, y].Verde == 255 && Nuevo[x
- 1, y].Azul == 255 && !Visitado[x-1, y])
              {
                ax1 = x - 1;
                ax2 = y;
                Nuevo[ax1, ax2].Rojo = 0;
                Nuevo[ax1, ax2].Azul = 0;
                Visitado[ax1, ax2] = true;
                revision = 0;
                escribir.WriteLine(ax1 + " " + (z) + " " + ax2);
                Region(ax1, ax2, z);
              }
            revision = 0;
            break;
        }
      }
```

# BIBLIOGRAFÍA


[1] Google. (2012, Junio 19). Google maps [En línea] Disponible: https://maps.Google.com.mx/

[2]Wikipedia contributors. (2012, Junio 19). Gaussian filter [En línea] Disponible: http://en.wikipedia.org/wiki/Gaussian_filter

[3] J. Canny, "A Computational Approach to Edge Detection", Proceedings of IEEE transactions on pattern analysis and machine intelligence, Vol. PAMI-8, No. 6, 1986, pp. 1-20 [En línea] Disponible: http://portal.acm.org/

[4] R. C. Gonzalez y R. E. Woods, Digital Image Processing, Editorial Addison-Wesley, 2a edición, 2002.

[5] Wikipedia contribution. (2012, Junio 17). Procedural generation [En línea] Disponible:

http://en.wikipedia.org/wiki/Procedural_generation

[6] Tu primer click del día! :: 180.com.uy. (2012, Julio 13). Apple presenta su propio programa de mapas, desafiando a Google [En línea] Disponible: http://www.180.com.uy/articulo/26769

[7] R. Bravo Cruz y E. Escamirosa Toriz, "Software para el cálculo de intensidad en el ambiente 3D", Universidad Autónoma Metropolitana Unidad Azcapotzalco, Proyecto terminal, Ingeniería Electrónica. 2005.

[8] M. A. Rojas Sandoval, "Algoritmo para acoplamiento automático por similitudes de imágenes digitales", Universidad Autónoma Metropolitana Unidad Azcapotzalco, Proyecto terminal, Ingeniería en Computación. 2010.

[9] L. Marín Díaz, "Sistema de reconocimiento del alfabeto dactológico utilizando procesamiento de imágenes", Universidad Autónoma Metropolitana Unidad Azcapotzalco, Proyecto terminal, Ingeniería en Computación. 2010.

[10] J. Xiao, T. Fang y P. Zhao, "Image-based Street-side City Modeling", Proceeding of ACM Transactions on Graphics, Vol. 28, No. 5, Article 114, pp. 1-12 [En línea] Disponible: http://portal.acm.org

[11] D. Simón Zorita, "Reconocimiento automático mediante patrones biométricos de huella dactilar", Universidad Politécnica de Madrid, Tesis doctoral, Ingeniería en Telecomunicaciones, 2003.

[12] Wikipedia contributors. (2012, Junio 19). Canny edge detector [En línea] Disponible:

http://en.wikipedia.org/wiki/Canny_edge_detector

[13] Kurt Jeages, XNA 4.0 Game Development by Example, Editorial [PACKT] PUBLISHING, 1a edición, 2010.

[14] Sean James, 3D Graphics with XNA Game Studio 4.0, Editorial [PACKT] PUBLISHING, 1a edición, 2010.

[15] Kurt Jeagers, XNA 4 3D Game Development by Example, Editorial [PACKT] PUBLISHING, 1a edición, 2012.

[16] Gonzalo Pajares Martinsanz y Jesús M., Visión por computador : imágenes digitales y aplicaciones, Editorial Alfaomega, Ra-Ma, 2a edición, 2008.