DISEÑO Y APLICACIÓN DE ALGORITMOS PARA EL TRATAMIENTO DE IMÁGENES DIGITALES EN LA EXTRACCIÓN Y CORRECCIÓN DE DATOS DE PRECIPITACIÓN DE LLUVIA REGISTRADOS EN PLUVIOGRAMAS, CON FINES DE ANÁLISIS Y ALMACENAMIENTO.

EDGAR MAURICIO CORTÉS GARCÍA

BUCARAMANGA
UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERIAS FISICO - MECANICAS
ESCUELA DE INGENIERIA DE SISTEMAS
MAYO, 2004

DISEÑO Y APLICACIÓN DE ALGORITMOS PARA EL TRATAMIENTO DE IMÁGENES DIGITALES EN LA EXTRACCIÓN Y CORRECCIÓN DE DATOS DE PRECIPITACIÓN DE LLUVIA REGISTRADOS EN PLUVIOGRAMAS, CON FINES DE ANÁLISIS Y ALMACENAMIENTO.

EDGAR MAURICIO CORTÉS GARCÍA

Trabajo de grado
Para optar por el título de
INGENIERO DE SISTEMAS

BUCARAMANGA
UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERIAS FISICO - MECANICAS
ESCUELA DE INGENIERIA DE SISTEMAS
MAYO, 2004

TITULO.
DISEÑO Y APLICACIÓN DE ALGORITMOS PARA EL TRATAMIENTO DE IMÁGENES DIGITALES EN LA EXTRACCIÓN Y CORRECCIÓN DE DATOS DE PRECIPITACIÓN DE LLUVIA REGISTRADOS EN PLUVIOGRAMAS, CON FINES DE ANÁLISIS Y ALMACENAMIENTO.*

AUTORES.
Edgar Mauricio Cortés García**

PALABRAS CLAVE.
Pluviograma
Señal
Imágenes
Tratamiento digital
Métodos numéricos
Spline
Bezier

CONTENIDO.
El análisis e interpretación de señales es una tarea común en muchas de las áreas científicas y profesionales actuales, un ingeniero civil analiza pluviogramas para obtener la información necesaria que le permita hacer predicciones climatológicas, análisis del potencial hidrológico de una región u otros tipo de estudios.

El inconveniente en este procedimiento se encuentra en que el pluviograma esta en un papel y los valores de precipitación con que se hace cualquier tipo de tratamiento deben ser hallados “contado líneas" en la cuadrícula del pluviograma, un proceso que a pesar de la destreza obtenida con el tiempo, no deja de ser una tarea agotadora en la cual se “Gasta” tiempo valioso. Se propone como un paso inicial hacia la solución de este tipo de inconvenientes el desarrollar un conjunto de algoritmos que puedan extraer la señal del papel para ser modificada y luego sometida externamente a análisis automáticos por computador donde se obtenga la información e indicadores finales que serán interpretados y analizados por el experto.

Las metodologías para el tratamiento de imágenes digitales se perfilan como un pilar en la resolución de problemas como el planteado, algoritmos que van desde la rotación hasta el etiquetado de segmentos conexos, pasando por el filtrado, umbralización y morfología matemática, son aplicados a la imagen con el objetivo final de distinguir la señal del resto de la imagen. Por otra parte, la implementación de algoritmos basados en métodos numéricos para realizar el ajuste de la señal extraída utilizando curvas de tipo Spline o Bezier se constituyen como dos diferentes pero efectivos métodos para corregir la información contenida en la señal.

---

* Proyecto de Grado

** Facultad de Ingenierías Físico-Mecánicas

Ingeniería de Sistemas

Ing. Sistemas, Ph.D. Alfonso Mendoza Castellanos

TITLE.
DESING AND APPLICATION OF ALGORITHMS FOR THE TREATMENT OF DIGITAL IMAGES DURING THE EXTRACTION AND CORRECTION OF RAIN FALLING DATA PROCESS REGISTRERED IN PLUVIOGRAMS, WITH THE PURPOSE OF ANALYSING AND SAVING INFORMATION.*

AUTHOR.
Edgar Mauricio Cortés García.**

KEY WORDS.
Pluviogram
Signal
Images
Digital treatment
Numeric methods
Spline
Bezier

CONTENT.

Nowadays, the signals analysis and interpretation process is a common work in several scientific and professional areas; for example, a civil engineer usually analyses pluviograms in order to obtain the basic information useful to let him make not only some climatologic predictions, but also, an analysis of the hydrologic potential of a specific region, and other kind of procedures.

The major problem in the process of signals analysis and interpretation of pluviograms is related to the lack of a systematic tool for the rain falling data analysis. In other words, due to the complicated presentation of pluviograms, an expert can spend a lot of time trying to find out all the important information that could help him to make an adequate prediction. A possible solution for this sort of problems is to develop a kit of algorithms that could split up the signal from the paper to be modified and analyzed using software designed to obtain the data and the final indicators. In this way, the expert could improve his /her signal analysis and interpretation process of pluviograms.

The methodologies for digital images treatment seem to be an amazing tool to solve some problems like the pluviograms analysis and interpretation. Based on a kit of algorithms that offer different options of images management (such as rotation, labeling, filtering, thresholding, and mathematic morphology) an expert can take the signal isolated from the rest of the image. In addition to that, inside the use of algorithms based on numeric methods, the curves Spline and Bezier seem to be two effective methods not only to make some signal adjustments, but also to correct the signal data.

---

* Proyecto de Grado

** Facultad de Ingenierías Físico-Mecánicas

Ingeniería de Sistemas

Ing. Sistemas, Ph.D. Alfonso Mendoza Castellanos

INTRODUCCIÓN

La tecnología y las ciencias de la computación deben servir al investigador como una herramienta que le libere de labores que por naturaleza una computadora puede realizar mejor que él, como por ejemplo los extensos cálculos matemáticos involucrados en las tantas áreas de investigación existentes, dejando al investigador en posición de invertir su tiempo en lo que por naturaleza puede hacer mejor que las computadoras, analizar e interpretar información que le permita brindar soluciones o explicar fenómenos en miras a avanzar en la consecución de un objetivo, muchas veces gracias a una característica netamente humana ausente en las placas de silicio que componen los microprocesadores y en las líneas de código escritas por un programador, el sentido común.

El análisis e interpretación de señales es una tarea común en muchas de las áreas científicas y profesionales actuales, por ejemplo, un cardiólogo analiza el electrocardiograma para diagnosticas enfermedades cardiacas, así como un ingeniero civil analiza pluviogramas para obtener la información necesaria que le permita hacer predicciones climatológicas, análisis del potencial hidrológico de una región u otro tipo de estudio.

Un registro pluviográfico análogo es el registro de los valores de precipitación acumulada respecto al tiempo registrada en un gráfico con valores máximos generalmente de 10mm de precipitación, con el objeto de llevar a cabo análisis es necesario un proceso de digitalización. El inconveniente se encuentra en que el pluviograma esta en un papel y los valores de precipitación con que se hace cualquier tipo de tratamiento deben ser hallados “contado líneas" en la cuadrícula del pluviograma, un proceso que a pesar de la destreza obtenida con el tiempo, no deja de ser una tarea agotadora en la cual se “Gasta” tiempo valioso.

El presente proyecto surge como un paso inicial hacia la solución de este tipo de inconvenientes al desarrollar un conjunto de algoritmos que puedan extraer la señal del papel para ser modificada y luego sometida externamente a análisis automáticos

por computador donde se obtenga la información e indicadores finales que serán interpretados y analizados por el experto.

Este libro contiene la descripción del proceso que se siguió hasta llegar a la implementación de los algoritmos que se plantearon anteriormente así como la interfaz gráfica utilizada para integrarlos y utilizarlos en la extracción y modificación de señales pluviométricas.

El capítulo 2 abarca una descripción del ambiente de desarrollo, incluyendo los aspectos de software, hardware, metodologías de desarrollo de software candidatas y se explica la forma en que se seleccionó la metodología a utilizar durante el desarrollo de este proyecto

Para un mayor enfoque del lector hacia los temas involucrados en el desarrollo de este proyecto, en el capítulo 3 se elaboró un resumen de los conceptos teóricos necesarios, incluyendo información acerca del tratamiento de imágenes digitales, métodos numéricos y conceptos técnicos referentes al área de sistemas, como son los conceptos acerca de programación orientada a objetos, también se hace una pequeña introducción a la pluviometría.

En el capítulo 4 se trata el tema del diseño y construcción de los algoritmos y la interfaz gráfica que los soporta. También se hace una descripción detallada del proceso de tratamiento al pluviograma desde la captura hasta su análisis y almacenamiento

Como una ayuda a los usuarios y a futuras modificaciones en mira a ampliar o mejorar el conjunto de algoritmos de tratamiento de imágenes digitales y métodos numéricos así como la interfaz gráfica especializada en la utilización de estos algoritmos en le tratamiento de imágenes pluviométricas digitales, se incluye en este libro el manual técnico y el manual de usuario contenidos en los capítulos 5 y 6 respectivamente. El manual técnico contiene la documentación de las clases y librerías reutilizadas e implementadas; en lo referente a clases MFC (Microsoft Foundament Class), este capítulo contiene un resumen de las más importantes

utilizadas, en cuanto a clases y librerías implementadas contiene una extensa documentación siguiendo el estándar propuesto por Microsoft en la documentación MSDN (Microsoft Developer Network). Por otra parte el manual de usuario contiene una descripción de los elementos que componen la interfaz gráfica desarrollada para soportar los algoritmos utilizados en el tratamiento de pluviogramas digitales.

1. ANTECEDENTES Y SITUACION PROBLEMA

1.1. SITUACION PROBLEMA

El IDEAM (Instituto de Hidrología, meteorología y estudios ambientales) es una institución científica y tecnológica con perfil internacional, recursos humanos altamente calificados, tecnologías de punta y todos sus procesos automatizados, que suministra información hidrometeorológica y ambiental en tiempo real sobre los procesos naturales y sus interrelaciones con el sistema social, económico y cultural. El IDEAM tiene como misión suministrar la información y el conocimiento ambiental a la comunidad colombiana para su avance hacia el desarrollo sostenible, su función es la de generar conocimiento, producir y suministrar datos e información ambiental además de realizar estudios, investigaciones, inventarios y actividades de seguimiento y manejo de la información que sirvan para fundamentar la toma de decisiones en materia política, ambiental y para suministrar las bases para el ordenamiento ambiental del territorio, al manejo, el uso y el aprovechamiento de los recursos naturales biofísicos del país.

La subdirección de hidrología del IDEAM está orientada a la adquisición de nuevos conocimientos sobre los procesos naturales que involucran el agua, su relación con otros elementos del medio natural y su efecto en diferentes sectores de la actividad nacional y de estos sobre el recurso. La subdirección de hidrología pone a disposición de la comunidad la información y los conocimientos sobre los procesos naturales que determinan el ciclo del agua, la oferta y la demanda hídrica, así como la interacción con los procesos socioeconómicos, como una herramienta fundamental en la definición de políticas, planificación y toma de decisiones para lograr el desarrollo sostenible y mejorar la calidad de vida de la población colombiana. Con esta información y conocimiento sobre el recurso hídrico, se determina la presencia de fenómenos naturales adversos y las variaciones en el régimen hidrológico, en particular, los relacionados con desbordamientos, inundaciones, sequías y contaminación, con el fin de tomar decisiones y llevar a cabo acciones que permitan reducir el impacto económico y social y orientar a la

comunidad nacional sobre la mejor utilización de los recursos hídricos y sus condiciones[1].

El IDEAM dispone a lo largo de toda Colombia de estaciones de monitoreo que permiten registrar la variación y el comportamiento de las precipitaciones de lluvia durante todo el año sobre todo el territorio nacional.

La medición de la pluviosidad se convierte entonces en una herramienta clave para la estimación del recurso hídrico de una región por cuanto da una razón de la cantidad de lluvia caída en un lugar determinado durante un periodo de tiempo dado. Con el fin de registrar estos datos, las estaciones de monitoreo disponen de pluviógrafos[2] entre sus herramientas de medición, los cuales, como resultado de su funcionamiento trazan pluviogramas[3].

Independientemente del diseño y tipo de pluviógrafo utilizado para la toma de estas mediciones, su funcionamiento básico es el siguiente: Un cilindro con papel pluviográfico permanece girando y en contacto con una aguja entintada que va registrando la variación de las precipitaciones de lluvia que se presentan durante las 24 horas del día. El papel tiene la longitud suficiente para registrar dicha actividad durante este periodo de tiempo. Para llevar a cabo tal registro el pluviógrafo dispone de un contenedor cuya funcionalidad es la de acumular el agua de lluvia que cae durante todo un periodo de registro (normalmente 24 horas). De esta forma y en todo momento, la aguja entintada va registrando en el papel la cantidad de agua acumulada en el contenedor ya sea que llueva o no, en el caso de que no se presente precipitación la gráfica dibujada no presentara ninguna variación con el paso del tiempo. En caso contrario, la gráfica registrara la forma en que varia en el tiempo la precipitación de agua a medida que llueve. Una vez el contenedor se encuentra lleno de agua este es vaciado automáticamente, momento en el cual la

[1] Referencia a la página WEB del IDEAM www.ideam.gov.co

[2] Aparato para medir la cantidad de agua de lluvia que cae en un lugar y tiempo dados.

[3] Gráfica que registra la cantidad de lluvia que cae en un lugar durante un periodo de tiempo determinado.

aguja vuelve a su estado inicial y continua registrando la precipitación de agua a medida que el contenedor vuelve a llenarse. Los pluviogramas son luego objeto de análisis por parte de especialistas en hidrología del IDEAM, quienes se basan en los datos aportados de las actividades registradas en las gráficas de forma individual e histórica para hacer predicciones, tomar decisiones, hacer planificaciones, determinar el estado del recurso hídrico de la región y contribuir con el cumplimiento de la misión del IDEAM y los objetivos planteados por la subdirección de hidrología perteneciente al instituto.

Debido a la naturaleza del mecanismo utilizado para la toma de mediciones por los pluviógrafos, existen algunos inconvenientes con la utilización de este método de registro. Para empezar, debido a las condiciones de humedad que poseen las regiones donde se encuentran algunas de las estaciones de registro durante ciertas épocas del año, las gráficas pueden presentar distorsiones en forma de manchones de tinta sobre la línea registrada, ocasionados precisamente debido a la humedad. Los manchones no son los únicos problemas ocasionados por la humedad, también se presentan gráficas con líneas poco definidas debido al taponamiento de la aguja.

Otro de los inconvenientes más frecuentes con este método de registro es que debido a descalibración del pluviógrafo se pueden presentar desfases en los valores registrados.
Un tercer inconveniente se presenta en algunos papeles de pluviometría que registran más de una gráfica, esto se puede dar porque por alguna razón no se cambia el papel luego de un ciclo de registro o porque se manipula de tal manera que pueda registrar una gráfica más corriendo papel hacia arriba o abajo sobre el tambor rotatorio.

Por otra parte, el solo hecho de registrar gráficas sobre papel hace que se presenten ciertos inconvenientes a corto y largo plazo, como por ejemplo problemas de almacenamiento debido a la gran cantidad de información almacenada en archivos, lo cual puede derivar en dificultades al hacer consultas individuales o análisis históricos de la información, por otra parte, la manipulación manual de los pluviogramas y el simple paso del tiempo generan degradación del papel haciendo

con el tiempo más difícil la consulta de información y más delicada su manipulación.

Todos los inconvenientes mencionados anteriormente derivan en dificultades al momento de querer realizar estudios y análisis a la información pluviométrica capturada, ya que dichos análisis están basados en la información de la gráfica que se hacen “contando cuadritos" sobre el papel, lo cual puede ser bastante tedioso y difícil dependiendo de la cantidad de información a analizar y del estado y en que se encuentre el pluviograma y de la calidad de la información contenida en él.

Por otra parte el GPH (Grupo de Predicción y Modelamiento Hidroclimatológico), es un grupo de investigación perteneciente a la escuela de Ingeniería Civil que de manera similar a la subdirección de hidrología del IDEAM, utiliza los pluviogramas como fuente de información para la realización de estudios y análisis con fines tanto educativos como investigativos. El GPH no dispone de estaciones de registro pluviométrico propias, luego sus integrantes se ven obligados a requerir y utilizar los pluviogramas obtenidos de las estaciones de registro del IDEAM, sin embargo, el IDEAM es muy reservado con esta información, debido a que por su naturaleza, la incorrecta manipulación del papel puede llevar a su daño o deterioro, para evitar esto, no se permite el préstamo de pluviogramas, solo se permiten un momento para sacarle copias. Como se trabaja entonces sobre copias, estas acarrean además de los mismos problemas de los pluviogramas originales la de que la información fotocopiada puede ser a veces de baja calidad y su deterioro en cualquier caso es mucho más rápido.

1.2. PROPUESTA

La tecnología y las ciencias de la computación deben servir al usuario como una herramienta que mejore agilizando y facilitando la forma en que se lleva acabo cualquier tipo de proceso, pensado en esto, la propuesta en este proyecto se basa en la implementación de un conjunto de algoritmos enfocados al tratamiento de imágenes digitales y la aplicación de métodos numéricos que aplicados en conjunto a la imagen digital de un pluviograma permita de él al extracción y corrección de la

señal pluviométrica para luego ser analizada y almacenada en medios magnéticos utilizando mucho menos espacio que el que ocuparía en el formato de una imagen digital tradicional. De esta manera independizamos el análisis de este tipo de información de la existencia física de pluviogramas una vez que estos sean digitalizados por los algoritmos mencionados.

Debido a que en un futuro pueden surgir mejores algoritmos y técnicas de análisis de imágenes digitales o métodos numéricos, así como pueden ser aplicadas para este propósito diferentes áreas de la tecnología informática que no fueron consideradas en este proyecto como por ejemplo la inteligencia artificial, este proyecto opta por la metodología de programación orientada a objetos (POO), la cual se sustenta, entre otros aspectos, en la facilidad que brinda para encapsular información y la disposición de una estructura de software jerárquica que permite el crecimiento del software basado en la herencia de clases y la inclusión de nuevos métodos como miembro de las mismas que permitan una nueva manipulación de los datos miembro que describen y caracterizan estas clases. De esta manera se brinda una estructura sólida a futuros proyectos que tengan como objetivo mejorar los alcances del presente o alcanzar metas similares en proyectos de diferente índole.

Debido a la gran cantidad de información que se debe manipular cuando se trabaja con imágenes digitales, al particular tamaño de las mismas al tratarse específicamente de pluviogramas y a los complicados algoritmos que deben ser aplicados sobre estas, se hace necesario la utilización de un lenguaje de programación que además de soportar perfectamente la POO también provea al programador de recursos que puedan mejorar el rendimiento de los algoritmos implementados en cuanto a velocidad de ejecución se refiere y disponga además de herramientas de producción que permitan optimizar los ejecutables finales con el mismo objetivo. Teniendo en cuenta estas necesidades se optó por el lenguaje de programación Visual C++ 6.0 de Microsoft por cumplir a cabalidad con los requerimientos planteados anteriormente.

1.3. ALCANCES DEL SISTEMA PROPUESTO

Una vez establecida más claramente la idea definiendo las características básicas de los algoritmos que se deseaban implementar, se da paso a establecer las funciones que cubrirían y las cuales se listan a continuación:

- Leer la imagen digital de un pluviograma y almacenar su información pictórica en una estructura de datos que permita su manipulación.
- Leer o guardar un fichero con un formato propio de la aplicación que represente un proyecto de análisis. A partir de este puede ser graficado nuevamente el pluviograma objeto del estudio y ser sometido a los análisis que la aplicación hace sobre la información pluviométrica.
- Convertir el formato de representación de la imagen desde cualquiera y hacia cualquiera de los formatos RGB (Red, Green and Blue), HSI (Hue, Saturation and Intensity) o HSV (Hue, Saturation and Value).
- Escoger inicialmente y de manera automática uno de los canales R, G o B de la imagen como sugerencia del canal que debe ser sometido al análisis por los algoritmos de tratamiento digital de imagen.
- Visualizar de manera independiente cualquiera de los canales independientemente del formato de representación en que se encuentre la imagen.
- Calcular un histograma del canal seleccionado como canal activo.
- Calcular a partir del histograma de un canal un nivel de umbralización de manera automática.
- Aplicar el procedimiento de umbralización sobre un canal con el fin iniciar la separación de la señal pluviométrica del resto de la información pictórica dando como resultado una imagen binaria.
- Aplicar procedimientos morfológicos de apertura, cierre, dilatación y erosión sobre una imagen binaria con el fin de reparar segmentos de líneas perdidos durante el procedimiento de umbralización y eliminar segmentos de líneas que no fueron correctamente umbralizados durante el correspondiente proceso y que se presentan en la imagen binaria como información no deseada.

- Aplicar procedimientos morfológicos de esqueletización sobre la imagen binaria con el fin de obtener un conjunto de líneas finas de mínimo espesor que representen la forma estructural de los trazos contenidos dentro de la imagen independiente del ancho que estos tengan.
- Ejecutar sobre una imagen binaria un procedimiento de etiquetado de elementos conexos que permita distinguir los trazos individuales dentro de la imagen binaria con el fin extraerlos a estructuras de datos convenientes para su manipulación y aplicar sobre ellos criterios que permitan evaluar y decidir cuales de ellos serán descartados del conjunto y cuales de ellos serán objeto de posteriores tratamientos numéricos y análisis.
- Realizar un ajuste a curvas Bezier o Spline sobre el trazo resultante mediante métodos numéricos.
- Manipular la curva ajustada con el fin de corregir la información substraída del pluviograma original.
- Aplicar sobre la información pluviométrica final los análisis requeridos por las entidades interesadas en el proyecto

Comprendiendo que la ejecución automática y sistemática de los algoritmos mencionados anteriormente puede dar origen a información defectuosa que corrompería los resultados finales del análisis, nuestra propuesta incluye una sencilla interfaz gráfica que permita manipular los algoritmos de manera interactiva a medida que se van presentando los resultados de la ejecución de cada uno de ellos con el fin de que el usuario pueda corregir o adaptar el algoritmo de manera que se mejoren los resultados arrojados por el mismo.

1.4. OBJETIVOS DEL PROYECTO

El objetivo general de este proyecto desarrollar un prototipo software que permita extraer los datos de precipitación de lluvia aportados por pluviogramas mediante el tratamiento de su imagen digital, con el fin de corregir, analizar y guardar la información obtenida.

Entrando más en detalle de este objetivo general, se plantearon los siguientes objetivos específicos:

- Diseñar e implementar un conjunto de librerías utilizando Visual C++ 6.0 y la metodología de desarrollo orientada a objetos que permitan:
- La descripción de la información pictórica contenida en imágenes digitales con fines de manipulación y procesamiento.
- Aplicar un tratamiento a la imagen digital de pluviogramas con el fin de aislar y extraer la información pictórica perteneciente a la señal pluviométrica.
- Diseñar e implementar un módulo de “Extracción de Datos” que cree un vector con la información pluviométrica mediante la conversión de la información pictórica de la señal aislada en unidades de precipitación de lluvia (mm/Unidad de tiempo), con el fin de proporcional una estructura de datos mas adecuada para la corrección de valores, análisis y almacenamiento digital.
- Diseñar e implementar un módulo de “Corrección de Datos” que permita hacer cambios y ajustes a los datos de precipitación de lluvia de la señal pluviométrica extraída, con el fin de corregir posibles errores en las magnitudes de los datos, errores posiblemente ocasionados por fallas o dificultades técnicas en la estación pluviométrica que registró el pluviograma objeto del procesamiento o alteración de información ocasionada en el proceso de procesamiento de la imagen o extracción de datos.
- Implementar un sistema de almacenamiento de información apoyado en Bases de datos, archivos planos u otro método de almacenamiento de datos que permita guardar el vector de datos que representa la señal pluviométrica extraída y corregida junto con la información de origen que la caracteriza con el fin de disponer de ella para posteriores representaciones gráficas y análisis.
- Diseñar e implementar un módulo de análisis preliminar de la señal pluviométrica que calcule mediante el procesamiento de la información almacenada a partir de cada pluviograma:
    - Curva de maza de precipitación.

    - Histograma de precipitación.
    - Histograma de Intensidad.
- Establecer un protocolo para la captura de la imagen digital de pluviogramas que garantice unas condiciones mínimas necesarias para su procesamiento.

En mira a solucionar los inconvenientes planteados anteriormente se propone al GPH una solución informática que permita la extracción y la corrección de datos contenidos en los pluviogramas, de esta manera podemos proporcionar a la información una estructura de datos que facilite su almacenamiento y admita su futura representación y análisis tanto individual como histórico.

1.5. DESCRIPCION DE OBJETIVOS

Los objetivos específicos planteados anteriormente están enfocados a brindar este tipo de solución a través de la investigación y adaptación de procedimientos en el área de procesamiento de imágenes digitales y métodos numéricos.

1.5.1. Objetivo específico 1.

La creación de rutinas mediante la metodología de programación orientada a objetos esta fundamentada en la creación de módulos reutilizables que aprovechen las ventajas de esta metodología tanto en el proyecto actual como en proyectos futuros. Por otra parte, la utilización de Visual C++ como lenguaje de programación se fundamenta en que es un lenguaje de programación que soporta la programación orientada a objetos y que dota al software programado en él de gran velocidad en comparación con lenguajes de programación como Visual Basic o Java

Un primer paso a dar hacia el procesamiento de imágenes digitales es proporcionar a la información pictórica contenida en un archivo de imagen una estructura conveniente para su manipulación. Matrices, listas enlazadas de datos o vectores simples son solo algunos de los formatos que podrían darse a este tipo de información, además la estructura elegida podría manipular datos de tipo entero corto, largo, número decimales sencillos o dobles entre otros. Uno de los criterios

más importantes a tener en cuenta en la selección del tipo de estructura de datos y tipo de datos contenidos en ella es la velocidad de ejecución que podrían tener las rutinas de procesamiento sobre la información contenida en ellas, ya que la cantidad de información que se tendrá almacenada en ellas podría llegar a ser muy grande y las rutinas de procesamiento y manipulación demasiado pesadas cuando se aplican a la imagen completa.

Luego de proporcionar una estructura de datos a la información tenemos que crear los procedimientos para el tratamiento de la imagen digital que permitan aislar la señal pluviométrica contenida dentro de la información pictórica. Gran parte del trabajo se halla en la creación de estas librerías ya aunque existe toda una teoría sobre la manipulación de imágenes digitales y procedimientos de mejoramiento, segmentación y descripción, estos se comportan de manera diferente cuando se aplican a un tipo de imagen en particular, es decir, no se obtienen los mismos resultados en la imagen de un paisaje o un rostro humano que los que se obtienen en una imagen tan particular como la que es objeto de nuestro estudio. También es posible que antes de realizar el aislamiento de la señal se deban aplicar algoritmos que mejoren la calidad de la imagen dependiendo de su estado actual. La calidad de la imagen es difícil de estimar de manera cuantitativa, el ser humano hace esta evaluación de manera subjetiva y luego le da un valor numérico que puede cambiar de una persona a otra, procedimiento difícil de reproducir en una máquina que no trabajo con interpretaciones subjetivas.

1.5.2. Objetivo específico 2.

Uno de los fines de este proyecto es proporcionar a la señal que se extraiga del pluviograma una estructura de datos que facilite el análisis de la información contenida en él. De esta forma, luego de tener la información pictórica de la señal totalmente aislada se debe proceder a convertir esta información a unidades precipitación por unidad de tiempo, de la misma manera que se encuentra en el pluviograma original y organizarla en vector de datos.
El módulo de extracción de datos será el encargado de esta labor, mediante la intervención del usuario este módulo se encargará de realizar el calculo de las

proporciones de la imagen en cuanto a la escala del tiempo y la escala que mide la precipitación de lluvia a tener en cuanta a la hora de realizar la extracción de datos.

1.5.3. Objetivo específico 3.

La información extraída por el módulo de extracción de datos es la base para empezar a trabajar sobre la señal, pero es importante primero corregir algunos errores que puedan presentar los datos extraídos, ya sean errores ocasionados durante la manipulación de la imagen, la conversión de datos o errores contenidos en el pluviograma fuente como por ejemplo errores de calibración u ocasionados por múltiples gráficas registradas en el pluviograma entre otros. Esta tarea estará cubierta por le módulo de corrección de datos mediante la utilización de métodos numéricos. Los métodos numéricos a emplear son motivo de investigación, debido a que es difícil automatizar totalmente el proceso se deben elegir y adaptar métodos numéricos que soporten la intervención de usuario con la intención de aprovechar la capacidad de evaluación subjetiva que este posee.

1.5.4. Objetivo específico 4.

Luego de tener la información disponible y corregida es importante proporcionar un sistema de almacenamiento electrónico que permita disponer de la información para su representación y análisis. Tener la información guardada de esta forma contribuye a evitar el deterioro de la misma, a hacer mucho más fácil disponer de copias para otras dependencias del grupo de investigación o de la universidad y a aprovechar el espacio físico ya que no se necesitará mas de los pluviogramas originales ni de los extensos archivos donde se guardan.

1.5.5. Objetivo específico 5.

Con el fin de mostrar la utilidad del procedimiento aplicado a los pluviogramas procesados y del nuevo formato en el que se dispone la información se pretende

implementar un módulo de análisis preliminar de información, en base a esta se calcularan:

Curva de maza de precipitación.
Histograma de precipitación.
Histograma de Intensidad.
El calculo de estas gráficas actualmente se hacen “contando cuadritos” sobre el pluviograma, con este módulo se pretende mostrar la utilidad al momento de disponer de la información para que una computadora haga los cálculos que actualmente se hacen a mano.

1.5.6. Objetivo específico 6.

Finalmente, La creación de un protocolo para la captura de la imagen digital del pluviograma que será objeto del procesamiento se hace necesaria debido a que las condiciones en que dicha imagen puede ser tomada pueden llegar a ser muy variadas en cuanto a resolución de imagen, resolución de color, formato de imagen y condiciones de iluminación entre otras características. La gran variedad en que se pueden encontrar estas propiedades haría que los algoritmos aplicados en cada nivel del procesamiento tuvieran que adaptarse a las condiciones en que se encuentre la imagen haciendo más difícil el trabajo de ya que no se puede asegurar que toda imagen en cualquier condición pueda ser procesada de manera satisfactoria.
El establecimiento de este protocolo pretende disminuir el conjunto de condiciones en que se puedan encontrar las imágenes digitales con el fin de que aumenten las posibilidades de que los algoritmos implementados y aplicados sobre las imágenes trabajen adecuadamente y arrojen resultados satisfactorios.

2. AMBIENTE DE DESARROLLO

2.1. REQUERIMIENTOS DE HARDWARE

2.1.1. Desarrollo

Para el desarrollo de los algoritmos y la interfaz de usuario propuesta se utilizó un computador con las siguientes características:

- Procesador AMD AThlon(tm) XP 1800+ a 1.54 GHz.
- Memoria RAM de 256 MB.
- Disco Duro de 20 GB.
- Unidad de CD de 52X.
- Unidad de disco extraíble de $3^{1/2}$.
- Conexión a Internet por Módem.

2.1.2. Mínimos de ejecución

Para la utilización de los algoritmos en una interfaz que permita su manipulación se requiere.

- Procesador Intel Celeron 700 MHz.
- Memoria RAM de 64 MB.
- Disco Duro de 2 Megas de espacio en el disco para instalar la aplicación, 3 megas de espacio en el disco por cada imagen en formato BMP que se desee guardar y 500 Kb por cada señal extraída que se desee guardar.
- Unidad de CD de 52X.
- Unidad de disco extraíble de $3^{1/2}$.

2.2. REQUERIMIENTOS DE SOFWARE

2.2.1. Desarrollo

Para el desarrollo de los algoritmos y la interfaz de usuario propuesta se utilizó el siguiente software:
Sistema operativo Microsoft Windows XP Profesional Edition.
Microsoft Visual C++ 6.0 Enterprise Edition.

2.3. METODOLOGIA DE DESARROLLO SOFTWARE

"Un proceso define quién está haciendo qué, cuándo, y cómo alcanzar un determinado objetivo. (...) Un proceso efectivo captura y presenta las mejores prácticas que el estado actual de la tecnología lo permite. En consecuencia reduce el riesgo y hace el proyecto mas predecible."[4]

2.3.1. Proceso unificado

Entre las metodologías más utilizadas para el desarrollo software se encuentra el proceso unificado, el cual es un proceso que proporciona normas para el desarrollo eficiente de software de calidad dentro de los plazos y presupuestos planeados. Entendiendo proceso de desarrollo de software como el conjunto de actividades necesarias para transformar los requisitos de un usuario en un sistema software.

El Proceso unificado esta dirigido por casos de uso, centrado en la arquitectura, y es iterativo e incremental.

DIRIGIDO POR CASOS DE USO: El desarrollo del software se centra en la importancia del desarrollo para el usuario y no en términos de funciones que debe cumplir el sistema. Los casos de uso dirigen el proceso durante todos los flujos de trabajo de las distintas fases.

Un caso de uso es una descripción de un conjunto de secuencias de acciones que un sistema lleva a cabo, un fragmento de su funcionalidad y que proporciona a un

[4] Jacobson, Ivar. Booch, Grady. Rumbaugh, James. El Proceso Unificado de Desarrollo de Software. Primera edición. Addison Wesley. España, 2000.

resultado interés para un actor determinado, donde un actor puede ser un usuario, un sistema o un rol.

CENTRADO EN LA ARQUITECTURA: Al describir la arquitectura se obtiene una mayor comprensión del sistema, se organiza el desarrollo y se fomenta la reutilización. Esta arquitectura abarca la organización del sistema software, los elementos estructurales que compondrán el sistema y sus interfaces, así como su comportamiento y colaboraciones entre elementos.

***ES ITERATIVO E INCREMENTAL*:** Un proceso iterativo permite una comprensión creciente de los requerimientos, a la vez que se va haciendo crecer el sistema abordando las tareas más riesgosas primero. El trabajo de desarrollo se divide de manera planeada en partes más pequeñas llamadas iteraciones lo cual genera progresivamente un incremento en el proyecto total.

En cada iteración, se identifican y especifican los casos de uso relevantes, se crea un diseño utilizando la arquitectura seleccionada como guía, se implementa el diseño mediante componentes, y se verifican que los componentes satisfacen los casos de uso. Si una iteración cumple sus objetivos el desarrollo continúa con la siguiente iteración, en caso contrario, se revisan las decisiones previas y se prueba un nuevo enfoque.

Un desarrollo iterativo, guiado por los casos de uso y centrado en la arquitectura, construye un software mediante pequeños incrementos, y añade cada incremento a la acumulación previa de incrementos de tal forma que siempre se tenga una construcción ejecutable. La arquitectura proporciona la estructura sobre la cual guiar las iteraciones mientras que los casos de uso definen los objetivos y dirigen el trabajo de cada iteración.

De esta manera el proceso reduce el riesgo de grandes retrasos en la entrega de un producto, se fijan metas más inmediatas por lo cual se puede controlar mejor el avance del proyecto.

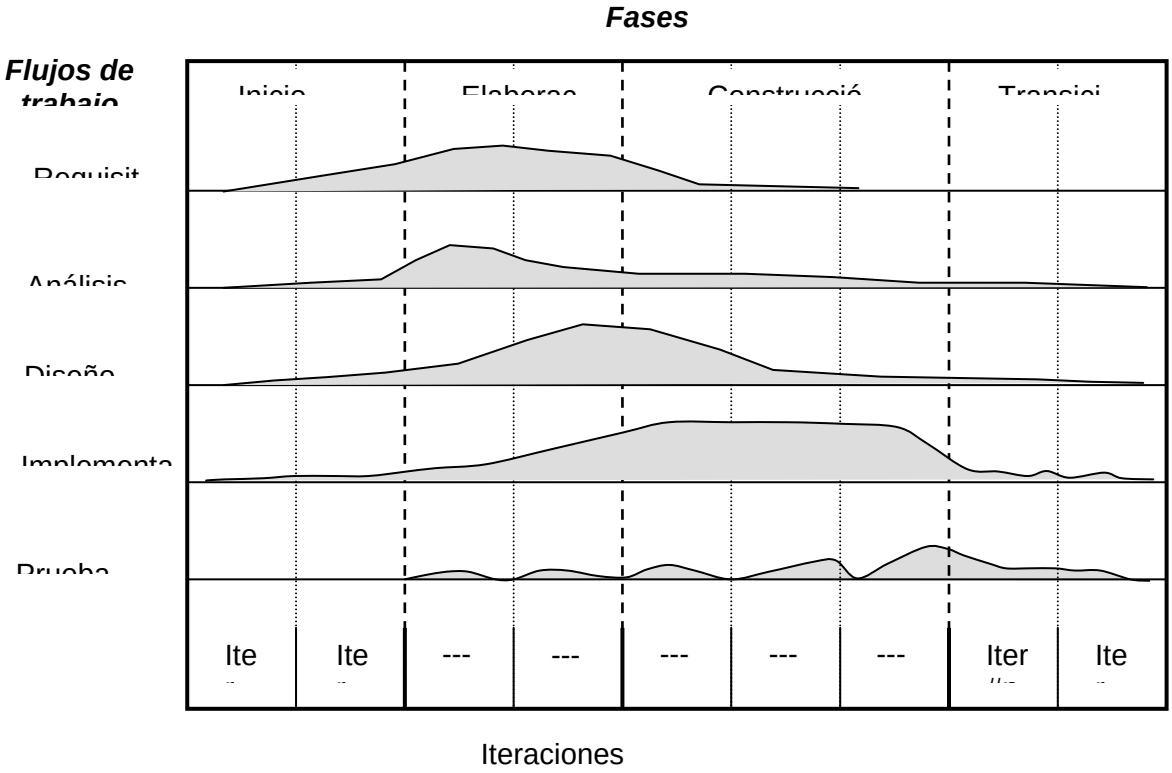


Figura 1. *Flujos de trabajo y fases del proceso unificado*

Fuente: Jacobson, Ivar. Booch, Grady. Rumbaugh, James. El Proceso Unificado de Desarrollo de Software.

El Proceso Unificado divide el proceso de desarrollo en ciclos, donde se obtiene una nueva versión del producto al final de cada ciclo. Cada ciclo se divide en cuatro Fases: Inicio, Elaboración, Construcción, y Transición. Cada una de estas fases concluye con un hito[5] bien definido donde deben tomarse decisiones respecto al

[5] Un hito es un evento que se sucede en el tiempo y que controla la iniciación o finalización de una grupo de tareas en un proyecto. El hito no consume tiempo ni

proyecto como la reestructuración del cronograma de trabajo. Cada una de estas fases se divide a su vez en iteraciones.

Cada iteración sigue la estructura de un pequeño ciclo de vida en cascada, pasando a través de los cinco flujos de trabajo fundamentales: requisitos, análisis, diseño, implementación y prueba.[6] La iteración también incluye la planificación que precede a los flujos de trabajo y la evaluación que va detrás de ellos.

2.3.2. Cascada

Hay dos tipos de cascada: cascada pura y cascada modificada.

2.3.2.1. Cascada pura

Un proyecto progresa a través de una secuencia ordenada de pasos llamados etapas, los cuales van desde el concepto inicial de software hasta la prueba del sistema, sus etapas no se solapan y una vez iniciada una de ellas es muy difícil devolverse a la anterior. Al final de cada etapa se realiza una evaluación para determinar si se puede pasar a la siguiente o se continúa en la actual hasta su cumplimiento. Este modelo no proporciona resultados tangibles en forma de software hasta que finaliza su ciclo.

---

recursos pero el hecho de que suceda permite que otras tareas puedan llevarse a cabo.

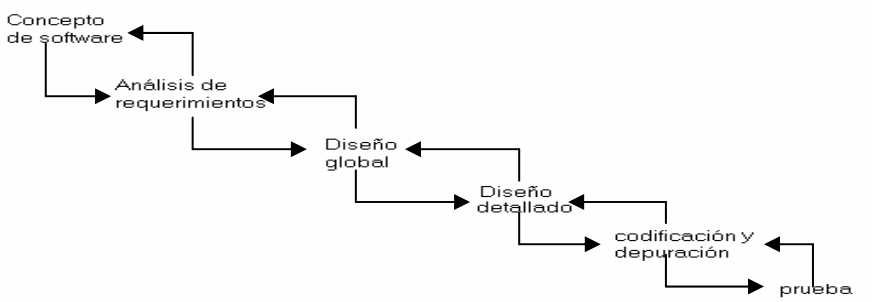
[6] Tomado del libro “Proceso Unificado de Desarrollo de Software” de Jacobson, Booch y Rumbaugh. Estas fases difieren del proceso unificado de Rational explicado en su sitio www.rational.com debido que en su clasificación se hacen una división entre modelo de negocio y requisitos, aquí tomado como un solo flujo llamado requisitos. Además en su método se unifica la fase de análisis y diseño. Para el resto del proyecto la metodología que se seguirá es la presentada en el libro citado en esta nota.

2.3.2.2. Cascada modificada

Es una modificación del modelo de cascada pura con fases solapadas donde aparecen subproyectos que resultan de dividir el sistema en subsistemas lógicamente independientes. Otra modificación es la llamada Cascada con Reducción de Riesgos, la cual elimina el problema de tener que comprender la totalidad de requerimientos antes de continuar con el diseño de la arquitectura.

Figura 2.2 Modelo de cascada pura.

Fuente: Jacobson, Ivar. Booch, Grady. Rumbaugh, James. El Proceso Unificado de Desarrollo de Software.



2.3.3. D.R.A. (Desarrollo rápido de aplicaciones)

Se utiliza un enfoque de construcción basada en componentes reutilizables y utilizando técnicas de cuarta generación en lugar de software con lenguajes de programación de tercera generación. Una de sus ventajas es el desarrollo de proyectos en periodos cortos de tiempo (de 60 a 90 días).

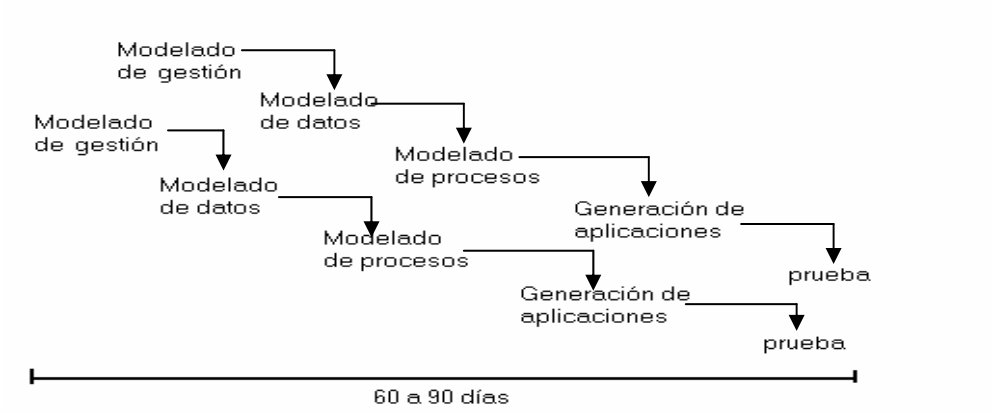


Figura 2. *Modelo D.R.A.*

Fuente: Jacobson, Ivar. Booch, Grady. Rumbaugh, James. El Proceso Unificado de Desarrollo de Software.

2.3.4. Prototipado

Hay dos clases de prototipado: el prototipado simple y el prototipado evolutivo.

2.3.4.1. Prototipado simple

Se construyen prototipos o acercamientos al sistema que sirven para identificar los requisitos del cliente. El paradigma de construcción de prototipos comienza con la recolección de los requisitos conocidos, se realiza un diseño rápido centrado en los aspectos visibles para el cliente y se construye el primer prototipo, éste será evaluado por el cliente y será usado para refinar los requisitos del software a desarrollar. Se construye un nuevo prototipo y se continúa esta secuencia hasta que se satisfacen las necesidades del cliente y el desarrollador comprende lo que necesita hacer. Este modelo es útil cuando no se definen los requisitos fácilmente.

2.3.4.2. Prototipado evolutivo

Toma sus bases del prototipado simple pero posee mayores controles sobre la calidad y desarrolla primero las áreas de mayor riesgo del sistema de tal forma que

el prototipo pueda ser tomado como producto final una vez se llegue a su fin. El prototipo evolutivo es un enfoque donde se desarrollan primero las partes seleccionadas del sistema y luego el resto a partir de esas partes. Está constituido por:

Concepto inicial
Diseño e implementación del prototipo inicial
Refinar el prototipo hasta que sea aceptable
Completar y entregar el prototipo

2.3.5. Entrega por etapas o modelo incremental

En este modelo no se entrega el producto total al final del proyecto sino que se muestra al cliente en etapas refinadas sucesivamente proporcionando una funcionalidad útil antes de entregar el 100% del proyecto.

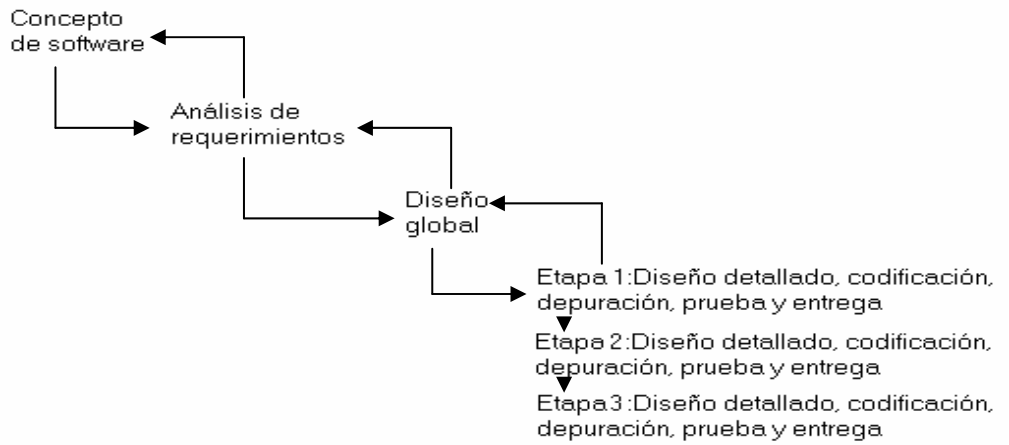


Figura 3. *Modelo de entrega por etapas o modelo incremental*

Fuente: Jacobson, Ivar. Booch, Grady. Rumbaugh, James. El Proceso Unificado de Desarrollo de Software.

2.3.6. Modelo en espiral

Es un modelo de ciclo de vida orientado a riesgos que divide un proyecto software en mini proyectos, cada mini proyecto se centra en uno o más riesgos. El método parte de una escala pequeña en medio del espiral, se localizan los riesgos, se genera un plan para manejarlos y a continuación se establece una aproximación a la siguiente iteración, después de controlar todos los riesgos más importantes el modelo en espiral finaliza con un modelo de ciclo de vida en cascada, con prototipado u otro modelo.

2.4. PROCESO DE ELECCION DE METODOLOGIA

Debido a la gran variedad de metodologías disponibles, se tuvieron en cuenta los siguientes aspectos para la selección de la metodología a utilizar en este proyecto:

Grado de identificación de los requerimientos.
Comprensión de la arquitectura a utilizar.
Grado de fiabilidad del sistema.
Grado de desarrollo en la generación del sistema.
Nivel de manejo de riesgos.
Estado de la planificación del proyecto.
Tiempo necesario en la gestión.
Existencias de modificaciones durante el transcurso del proyecto.
Presentación de progresos a clientes y directivos interesados en el proyecto.
Nivel de sofisticación para directivos y desarrolladores.

Para evaluar dichos aspectos se utilizó la siguiente matriz de decisión[7]:

Capacidad del modelo	Cascada Pura	Cascada Modificada	DRA	Prototipado evolutivo	Espiral	Incremental	Proceso Unificado
A: alto	10	3	10	0	0	10	0

[7] Calificación de 0 a 10 puntos, siendo 10 el puntaje máximo.

B: alto	10	3	10	8	0	10	8
C: alto	10	10	5	5	10	10	10
D: alto	10	10	6	10	10	10	10
E: medio	2	6	10	6	10	7	10
F: definido	5	5	10	0	5	6	10
G: medio	2	10	5	6	6	7	4
H: medio	2	6	5	10	6	6	10
I: alto	2	7	10	8	10	9	10
J: medio	6	4	5	2	2	6	10
TOTALES	59	64	76	55	59	81	82

Figura 4. *Matriz de decisión de la metodología de desarrollo*

Teniendo en cuenta los resultados obtenidos a partir de la anterior matriz de decisión, se ha optado por utilizar el proceso unificado con algunas modificaciones al mismo para adaptarse al tipo de arquitectura escogida para el sistema.

3. MARCO TEORICO

3.1. PROGRAMACION ORIENTADA A OBJETOS

3.1.1. Introducción

La programación orientada a objetos es una evolución de la programación procedural basada en funciones. La POO nos permite agrupar secciones de código con funcionalidades comunes.

Una de las principales desventajas de la programación procedural basada en funciones es su construcción, cuando una aplicación bajo este tipo de programación crece, la modificación del código se hace muy trabajosa y difícil debido a que el cambio de una sola línea en una función, puede acarrear la modificación de muchas otras líneas de código pertenecientes a otras funciones que estén relacionadas.

Con la programación orientada a objetos se pretende agrupar el código encapsulándolo y haciéndolo independiente, de manera que una modificación debida al crecimiento de la aplicación solo afecte a unas pocas líneas.

Actualmente una de las áreas más candentes en la industria y en el ámbito académico es la orientación a objetos. La orientación a objetos promete mejoras de amplio alcance en la forma de diseño, desarrollo y mantenimiento del software ofreciendo una solución a largo plazo a los problemas y preocupaciones que han existido desde el comienzo en el desarrollo de software: la falta de portabilidad del código y reusabilidad, código que es difícil de modificar, ciclos de desarrollo largos y técnicas de codificación no intuitivas.

Un lenguaje orientado a objetos ataca estos problemas. Tiene tres características básicas:

Debe estar basado en objetos.
Basado en clases.

Ser capaz de tener herencia de clases.

Muchos lenguajes cumplen uno o dos de estos puntos; muchos menos cumplen los tres. La barrera más difícil de sortear es usualmente la herencia.

El concepto de programación orientada a objetos (OOP) no es nuevo, lenguajes clásicos como SmallTalk se basan en ella. Dado que la OOP. Se basa en la idea natural de la existencia de un mundo lleno de objetos y que la resolución del problema se realiza en términos de objetos, un lenguaje se dice que está basado en objetos si soporta objetos como una característica fundamental del mismo.

El elemento fundamental de la OOP es, como su nombre lo indica, el **objeto**. Podemos definir un objeto como **un conjunto complejo de datos y programas que poseen estructura y forman parte de una organización**. Estos objetos contienen una serie de procedimientos e información destinadas a resolver un grupo de tareas con un denominador común. Un procedimiento que este situado en un objeto no podrá ser usado por otro procedimiento perteneciente a otro objeto, si no es bajo una serie de reglas. Los datos que mantenga el objeto, permanecerán aislados del exterior y sólo se podrá acceder a ellos siguiendo ciertas normas.

Esta definición especifica varias propiedades importantes de los objetos. En primer lugar, un objeto no es un dato simple, sino que contiene en su interior cierto número de componentes bien estructurados. En segundo lugar, cada objeto no es un ente aislado, sino que forma parte de una organización jerárquica o de otro tipo. El objetivo de POO es catalogar y diferenciar el código, basándose en estructuras jerárquicas dependientes, al estilo de un árbol genealógico.

Los objetos se crean a partir de una serie de especificaciones o normas que definen como va a ser el objeto, esto es lo que en POO se conoce como una clase.

3.1.2. EVOLUCION DE LA POO

A fin de comprender lo que es la POO, es necesario comprender sus raíces. Así pues, comenzaremos por examinar la historia del proceso de programación analizada cómo evolución POO y deduciendo, en consecuencia, por qué es tan importante este concepto.

3.1.2.1. Programación lineal.

Los lenguajes de programación lineal (BASIC,COBOL Y FORTRAN) no tenían facilidad para reutilizar el código existente de programas. De hecho se duplicaban segmentos de software cada vez más en muchos programas. Los programas se ejecutaban en secuencias lógicas, haciendo la lógica difícil de comprender. El control de programas era difícil y se producían continuos saltos a lo largo del referido programa. Aún más, los lenguajes lineales no tenían capacidad de controlar la visibilidad de los elementos llamados datos.

3.1.2.2. Programación Modular

El soporte más elemental de la programación Modular llegó con la aparición de la subrutina. Una subrutina ha creado una secuencia de instrucciones a las que se les da un nombre independiente; una vez que se ha definido, la subrutina se puede ejecutar simplemente incluyendo el nombre del programa siempre que se requiera. Las subrutinas proporcionan una división natural de las tareas; diferentes programas utilizan. Aunque las subrutinas proporcionan el mecanismo básico de la programación Modular, se necesita mucha disciplina para crear software bien estructurado. Sin esta disciplina, es fácil escribir programas compilados y tortuosos difíciles de modificar y comprender, así como imposible de mantener. Esta ha sido la panorámica durante muchos años en el desarrollo del software.

3.1.2.3. Programación Estructurada.

Un concepto importante en el campo de la programación Estructurada es la Abstracción, ya que la Abstracción se puede definir como la capacidad de examinar algo sin preocuparse de los detalles internos. En un programa estructurado, es

suficiente conocer que un procedimiento sea fiable, para que se pueda utilizar sin tener que conocer cómo funciona su interior. Esto se conoce como una Abstracción funcional y es el núcleo de la programación estructurada. Hoy casi todos los lenguajes de programación tienen construcciones que facilitan la programación estructurada.

3.1.3. Conceptos sobre clases

3.1.3.1. Definición de clase

De la definición de objeto, se sabe que la estructura y comportamiento de objetos similares están definidos en su clase común. Mientras que un objeto es una entidad concreta que existe en el tiempo y el espacio[8], una clase representa sólo una abstracción. Tal abstracción representa un grupo o conjunto de atributos y operaciones comunes, generada a partir de un proceso de descomposición de alguna complejidad y que lleva a establecer una clasificación basada en aspectos de calidad, grado de competencia o condición

Las propiedades definen los datos o información del objeto, permitiendo modificar o consultar su estado, mientras que los métodos son las rutinas que definen el comportamiento del objeto. Es necesario tener muy clara cual es la diferencia entre un objeto y una clase, a este respecto podemos decir que una clase constituye la representación abstracta de algo mientras que un objeto constituye la representación concreta de lo que la clase define. Imaginemos los planos de una casa diseñados por un arquitecto, en ellos encontramos el esquema de la casa, las medidas, los materiales etc. Una vez construida la casa podremos comprobar que cumple todo lo que los planos determinaban, de esta manera podemos comparar los planos de la casa con las clases en POO, y la casa en si con un objeto creado a partir de una clase. Se debe destacar también que con los mismos planos se

[8] Una clase es un conjunto de objetos que comparten una estructura común y un comportamiento común. Un objeto es una instancia de una clase [Booch].

pueden crear muchas casas iguales, lo mismo ocurre en POO, a partir de una clase se pueden crear muchos objetos iguales.

Un objeto no es una clase, pero curiosamente una clase puede ser un objeto. Los objetos que no comparten estructura y comportamiento similares no pueden agruparse en una clase porque por definición, no están relacionados entre sí a no ser por su naturaleza general como objetos. En muchos casos, un proceso de descomposición no ayuda a establecer de manera correcta las agrupaciones requeridas para definir un grupo de clases, y mucho menos definir una abstracción en una sola clase. En tales casos es más conveniente capturar esas abstracciones como una agrupación de clases cuyas instancias colaboran para proporcionar el comportamiento y estructura deseados.

3.1.3.2. Interfaz e implementación

Mientras que un objeto individual es una entidad concreta que desempeña algún papel en el sistema global, la clase captura la estructura y comportamiento comunes a todos los objetos relacionados. Por lo que una clase define un contrato o interfaz que vincula a la abstracción que representa con todos sus clientes por medio de las operaciones definidas, así un lenguaje con comprobación estricta de tipos puede detectar violaciones de este contrato o interfaz durante la compilación. Este contrato o interfaz permite establecer la visión interna y externa de una clase.

La interfaz de una clase proporciona una visión externa y por lo tanto enfatiza la abstracción a la vez que oculta su estructura y los secretos de su comportamiento. Esta interfaz se compone principalmente de las declaraciones de todas las operaciones aplicables a instancias de esta clase, pero también puede incluir la declaración de otras clases, constantes, variables y excepciones según se necesita para completar la abstracción. La implementación de una clase es su visión interna que engloba los secretos de su comportamiento.

La creación de un objeto a partir de una clase se conoce como instanciación de un objeto.

3.1.4. CARACTERISTICAS DE LA POO

Hay un cierto desacuerdo sobre exactamente que características de un método de programación o lenguaje le califican como "orientado a objetos", pero hay un consenso general en que las características siguientes son las más importantes

3.1.4.1. Abstracción

Una abstracción de datos de cualquier lenguaje de programación, es la capacidad de crear tipos de datos definidos por el usuario.

La abstracción se define como la "***extracción de propiedades esenciales de un concepto***". Con la abstracción de datos, las estructuras de datos e ítems se pueden utilizar sin preocuparse sobre los detalles exactos de la implementación.

La abstracción de datos permite no preocuparse de los detalles no esenciales. La abstracción de datos existe en casi todos los lenguajes de programación. Las estructuras de datos y tipos de datos son un ejemplo de abstracción. Los procedimientos y las funciones son otro ejemplo. Sin embargo, solo recientemente han emergido lenguajes que soportan sus propios tipos abstractos de datos TAD (Tipos de datos abstractos), como Pascal, Ada, Modula-2 y naturalmente C++.

En los Tipos Abstractos de Datos de C++ es posible incluir, además de los datos, funciones que manipulan a esos datos. Estos se pueden implementar en C++ con los tipos de estructura (struct) y clase (class).

En C++ un tipo de dato definido por el usuario se declara con una estructura:

```
struct fecha {
    int dia;
    int mes;
    int anyo;
```

};

Esta estructura tiene tres miembros día, mes y anyo, a veces esta definición se sitúa en un archivo de cabecera y se incluye al principio de todos los programas que utiliza la estructura. Se desea declarar una variable que tenga el formato de la estructura, y se hace de la siguiente manera:

fecha hoy; //declara una variable estructurada

Cada objeto en el sistema sirve como modelo de un "agente" abstracto que puede realizar trabajo, informar y cambiar su estado, y "comunicarse" con otros objetos en el sistema sin revelar cómo se implementan estas características. Los procesos, las funciones o los métodos pueden también ser abstraídos y cuando los están, una variedad de técnicas son requeridas para ampliar una abstracción.

3.1.4.2. Encapsulación

También llamada "ocultación de la información", esto asegura que los objetos no pueden cambiar el estado interno de otros objetos de maneras inesperadas; solamente los propios métodos internos del objeto pueden acceder a su estado. Cada tipo de objeto expone una interfaz a otros objetos que especifica cómo otros objetos pueden interactuar con él. Algunos lenguajes relajan esto, permitiendo un acceso directo a los datos internos del objeto de una manera controlada y limitando el grado de abstracción.

Uno de los objetivos de la encapsulación es ocultar los detalles de la implementación de un objeto, con lo cual se consigue una correcta funcionalidad para la abstracción que se está representando.

3.1.4.3. Polimorfismo

Las referencias y las colecciones de objetos pueden contener objetos de diferentes tipos, y la invocación de un comportamiento en una referencia producirá el

comportamiento correcto para el tipo real del referente. Cuando esto ocurre en "tiempo de ejecución", esta última característica se llama asignación tardía o asignación dinámica. Algunos lenguajes proporcionan medios más estáticos (en "tiempo de compilación") de polimorfismo, tales como las plantillas y la sobrecarga de operadores de C++.

La característica de polimorfismo de la POO permite que las instancias de diferentes clases reaccionen de una forma particular al llamado de una función. Por ejemplo, en una jerarquía de formas gráficas (punto, línea, cuadro, rectángulo, etc.), cada forma tiene una función >Draw responsable de contestar adecuadamente a peticiones del trazo de esa forma.

El polimorfismo permite que las instancias de diferentes clases respondan a la misma función en forma adecuada para cada clase.

El origen del termino polimorfismo es simple: ya que describe la capacidad de código de C++ de comportarse de diferentes maneras dependiendo de situaciones que se presenten al momento de la ejecución. Este comportamiento a menudo escapa del control directo del programador.

El polimorfismo no es tanto una característica de objetos como lo es de funciones miembro de una clase. Se implanta a través de la arquitectura de clases; sin embargo, solo las funciones miembro de la clase puede ser polimorficas, y no la clase completa

Considere una serie de clases que se emplean para implantar una base de datos que lleve el registro de obras maestras de arte. Las clases están organizadas en una jerarquía, con la intensión de permitir al programador tener acceso a cualquier objeto de la base de datos sin que sepa de que tipo de objeto se trata

Cualquier carácteristica común en la base de datos pertenece al nodo más alto posible del árbol. Todos y cada uno de los nodos de la base de datos tiene un autor

y una fecha. Otra información puede variar entre clases, como el método que se emplea para desplegar información acerca de un objeto.

Esta función de presentación es un candidato para el polimorfismo y se implanta con el mecanismo de las funciones virtuales.

3.1.4.4. Herencia

Organiza y facilita el polimorfismo y la encapsulación permitiendo a los objetos ser definidos y creados como tipos especializados de objetos preexistentes. Estos pueden compartir (y extender) su comportamiento sin tener que volver a implementar su comportamiento. Esto suele hacerse habitualmente agrupando los objetos en clases y las clases en árboles o enrejados que reflejan un comportamiento común.

La herencia es propiedad que permite a los objetos construirse a partir de otros objetos. El concepto de herencia está presente en nuestras vidas diarias donde las clases se dividen en subclases.

La herencia permite entonces describir entidades (clases) por diferencia entre ellas; por eso se dice que programar con orientación a objetos es programar para diferenciar. La utilidad de la herencia es múltiple. En primer lugar hay que analizar, diseñar, codificar y poner a punto, que en cuanto se tiene un concepto definido, jamás lo repetimos. Como muchos, si el concepto nuevo es muy parecido a uno anterior, se define el nuevo como la diferencia respecto al anterior. Si surge un error en la utilización de la nueva entidad, seguro que el error está en lo que se ha añadido por que lo que se ha heredado ya se había producido.

Hay diferentes tipos de herencia los más importantes son simples y múltiples. Con esto se quiere decir que, al contrario de la herencia biológica donde siempre se hereda de dos padres, en la programación orientada a objetos se puede heredar de uno, dos o más padres.

La definición de cliente minorista a partir de cliente es un ejemplo de herencia simple y se podría definir un vehículo anfibio como una herencia (subclase) de otras dos clases, coche y barco.

Es posible que todas las descripciones anteriores sean interesantes, pero sólo los detalles que se ofrecen a través de un programa real demuestran que es la herencia y como funciona. Además aquí se muestra un breve ejemplo de dos clases, donde la segunda de ellas hereda propiedades de la primera.

```
class box{
public:
int width,heigth;
void SetWidth(int w) {width = w;}
void SetHeigth(int h) {heigth = h; }
};
class ColoredBox: public Box{
public:
int color;
void SetColor(int c) {coor =c;}
```

La clase Box recibe el nombre de clase de base de la clase ColoredBox que a su vez recibe el nombre de clase derivada. A si mismo, las clases base denominadas a veces clases protegidas o primarias. La clase ColoredBox se declara solo con una función, pero también hereda dos funciones y dos variables de su clase base. Así, se puede crear el código siguiente:

```
ColoredBox cb;
Void main( )
{
//utilice funciones miembro en ColoredBox
cb.SetColor(5) // no hereda
cb.SetWidth(3); // heredadas
cb.SetHeigth(50); // heredadas
```

```
}
```

Observe como las funciones heredadas se utilizan exactamente como no heredadas. La clase ColoredBox ni siquiera tuvo que mencionar el hecho de que las funciones Box::SetWidth( ) Box::SetHeigth() fueron heredadas. Esta uniformidad de expresión es una característica formidable de C++.

3.1.5. Problemas derivados de la utilización de OOP en la actualidad

Un sistema orientado a objetos, por lo visto, puede parecer un paraíso virtual. El problema sin embargo surge en la implementación de tal sistema. Muchas compañías oyen acerca de los beneficios de un sistema orientado a objetos e invierten gran cantidad de recursos luego comienzan a darse cuenta que han impuesto una nueva cultura que es ajena a los programadores actuales. Específicamente los siguientes temas suelen aparecer repetidamente:

Curvas de aprendizaje largas: Un sistema orientado a objetos ve al mundo en una forma única. Involucra la conceptualización de todos los elementos de un programa, desde subsistemas a los datos, en la forma de objetos. Toda la comunicación entre los objetos debe realizarse en la forma de mensajes. Esta no es la forma en que están escritos los programas orientados a objetos actualmente; al hacer la transición a un sistema orientado a objetos la mayoría de los programadores deben capacitarse nuevamente antes de poder usarlo.

Dependencia del lenguaje: A pesar de la portabilidad conceptual de los objetos en un sistema orientado a objetos, en la práctica existen muchas dependencias. Muchos lenguajes orientados a objetos están compitiendo actualmente para dominar el mercado. Cambiar el lenguaje de implementación de un sistema orientado a objetos no es una tarea sencilla; por ejemplo C++ soporta el concepto de herencia múltiple mientras que SmallTalk no lo soporta; en consecuencia la elección de un lenguaje tiene ramificaciones de diseño muy importantes.

Determinación de las clases: Una clase es un molde que se utiliza para crear nuevos objetos. En consecuencia es importante crear el conjunto de clases adecuado para un proyecto. Desgraciadamente la definición de las clases es más un arte que una ciencia. Si bien hay muchas jerarquías de clase predefinidas usualmente se deben crear clases específicas para la aplicación que se este desarrollando. Luego, en 6 meses ó 1 año se da cuenta que las clases que se establecieron no son posibles; en ese caso será necesario reestructurar la jerarquía de clases devastando totalmente la planificación original.

Desempeño: En un sistema donde todo es un objeto y toda interacción es a través de mensajes, el tráfico de mensajes afecta el desempeño. A medida que la tecnología avanza y la velocidad de procesamiento, potencia y tamaño de la memoria aumentan, la situación mejorará; pero en la situación actual, un diseño de una aplicación orientada a objetos que no tiene en cuenta el desempeño no será viable comercialmente.
Idealmente, habría una forma de atacar estos problemas eficientemente al mismo tiempo que se obtienen los beneficios del desarrollo de una estrategia orientada a objetos. Debería existir una metodología fácil de aprender e independiente del lenguaje, y fácil de reestructurar que no drene el desempeño del sistema.

3.2. LENGUAJE DE MODELADO

Para facilitar el entendimiento del modelado se utilizó el lenguaje de modelado **UML** (Unified Modeling Language). Este es un lenguaje para especificar, visualizar, construir y documentar las diferentes etapas del desarrollo de software, así como para modelado de procesos de negocio u otros sistemas no-software. UML reúne una colección de las mejores prácticas en la ingeniería que han sido utilizadas con éxito para modelar sistemas grandes y complejos, ya que cubre tanto objetos conceptuales como los procesos de negocio y funciones del sistema, como también objetos concretos como clases en un lenguaje de programación, esquemas de base de datos y componentes reutilizables de software.

UML ha sido creado por los expertos en la metodología orientada a objetos tales como Grady Booch, Ivar Jacobson, y James Rumbaugh en Rational Software, utilizando información de otros importantes expertos en metodología, vendedores de software, y usuarios finales. Su objetivo era unificar los diversos sistemas que había y crear un lenguaje de modelado con las mejores características de cada uno.

El UML fue adoptado por el OMG (Object Management Group) como estándar en noviembre de 1997 y comenzó rápidamente a ser utilizado en el diseño, especificación, construcción, visualización y documentación de software.

3.2.1. Concepto de lo estático y lo dinámico

En UML se manejan dos conceptos para describir construcciones que representan aspectos del sistema, los cuales se modelan por medio de vistas: la estática y la dinámica.

En el nivel superior las vistas se pueden dividir en tres áreas: Clasificación estructural, comportamiento dinámico y gestión del modelo.

Figura 5. *Herramientas de las vistas de UML*

VISTAS	*HERRAMIENTAS*
Clasificación Estructural	Diagramas de clases, casos de uso, componentes, nodos, y elementos
Comportamiento dinámico	Diagramas de maquina de estados, de actividad y de interacción.
Gestión del modelo	Diagramas de paquetes

La **clasificación estructural** describe los elementos del sistema y sus relaciones con otros elementos. Los clasificadores incluyen clases, casos de uso, componentes, nodos, y elementos proporcionan la base sobre la cual se construye el comportamiento dinámico.

La clasificación de las vistas incluye la vista estática, la vista de casos de uso y la vista de implementación.

Algunos de los elementos que se pueden clasificar como estáticos son los siguientes:

***Paquete**:* Es el mecanismo de que dispone UML para organizar sus elementos en grupos, se representa un grupo de elementos del modelo. Un sistema es un único paquete que contiene el resto del sistema, por lo tanto, un paquete debe poder anidarse, permitiéndose que un paquete contenga otro paquete.

Clases: Una clase representa un conjunto de objetos que tienen una estructura, un comportamiento y unas relaciones con propiedades parecidas. Describe un conjunto de objetos que comparte los mismos atributos, operaciones, métodos, relaciones y significado. En UML una clase es una implementación de un tipo.

Las clases pueden tener varios parámetros formales, son las clases denominadas plantillas. Sus atributos y operaciones vendrán definidas según sus parámetros formales. Las plantillas pueden tener especificados los valores reales para los parámetros formales, entonces reciben el nombre de clase parametrizada instanciada. Se puede usar en cualquier lugar en el que se podría aparecer su plantilla.

Relacionando con las clases nos encontramos con el término utilidad, que se corresponde con una agrupación de variables y procedimientos globales en forma de declaración de clase, también puede definirse como un estereotipo (o nueva clase generada a partir de otra ya existente) de un tipo que agrupa variables globales y procedimientos en una declaración de clase. Los atributos y operaciones que se agrupan en una utilidad se convierten en variables y

operaciones globales. Una utilidad no es fundamental para el modelado, pero puede ser conveniente durante la programación.

Tipos: Es un descriptor de objetos que tiene un estado abstracto y especificaciones de operaciones pero no su implementación. Un tipo establece una especificación de comportamiento para las clases.

Interfaz: Representa el uso de un tipo para describir el comportamiento visible externamente de cualquier elemento del modelo.

Relación entre clases: Las clases se relacionan entre sí de distintas formas, que marcan los tipos de relaciones existentes:

Asociación: Es una relación que describe un conjunto de vínculos entre clases. Pueden ser binarias o n-arias, según se implican a dos clases o más. Las relaciones de asociación vienen identificadas por los roles, que son los nombres que indican el comportamiento que tienen los tipos o las clases, en el caso del rol de asociación (existen otros tipos de roles según la relación a la que identifiquen). Indican la información más importante de las asociaciones. Es posible indicar el número de instancias de una clase que participan en una relación mediante la llamada multiplicidad. Cuando la multiplicidad de un rol es mayor que 1, el conjunto de elementos que se relacionan puede estar ordenado. Las relaciones de asociación permiten especificar qué roles van a estar asociados con otro objeto mediante un calificador. El calificador es un atributo o conjunto de atributos de una asociación que determina los valores que indican cuales son los valores que se asociarán. Una asociación se dirige desde una clase a otra (o un rol a otro), el concepto de navegabilidad se refiere al sentido en el que se recorre la asociación.

Composición: Es un tipo de agregación donde la relación de posesión es tan fuerte como para marcar otro tipo de relación. Las clases en UML tienen un tiempo de vida determinado, en las relaciones de composición, el tiempo de vida de la clase que es parte del todo (o agregado) viene determinado por el tiempo de vida de la clase que

representa el todo, por tanto es equivalente a un atributo, aunque no lo es porque es una clase y puede funcionar como tal en otros casos.

Generalización: Cuando se establece una relación de este tipo entre dos clases, una es una Superclase y la otra es una Subclase. La subclase comparte la estructura y el comportamiento de la superclase. Puede haber más de una clase que se comporte como subclase.

Dependencia: Una relación de dependencia se establece entre clases (o roles) cuando un cambio en el elemento independiente del modelo puede requerir un cambio en el elemento dependiente.

Las vistas físicas modelan los conceptos de la aplicación desde un punto de vista lógico. Las vistas físicas modelan la estructura de la implementación de la aplicación por si misma, su organización de componentes y su despliegue en nodos de ejecución

Figura 6. *Clasificación estructural de las vistas de UML*

Clasificacion structural

Vista estática	Se exhibe en los diagramas de clase. Las clases se pueden describir con varios niveles de precisión. Al empezar el diseño, el modelo captura los aspectos más lógicos del problema.
Vista de casos de uso	Modela la funcionalidad del sistema según lo perciben los usuarios externos, llamados actores. Un caso de uso es una unidad coherente de funcionalidad expresada como transacción entre actores y sistema
Vista de implementación	Hay dos vistas físicas: la vista de implementación y la vista de despliegue.

Clasificacion structural

	La vista de implementación modela los componentes de un sistema a partir de los cuales se construye la aplicación La vista de implementación se representa en diagramas de componentes. La vista de despliegue representa la disposición de las interfaces de las instancias de componentes de ejecución en instancias de nodos. Un nodo es un recurso de ejecución, tal como una computadora, un dispositivo o memoria.

El **comportamiento dinámico** describe el comportamiento del sistema en el tiempo. Las vistas de un comportamiento dinámico incluyen vista de maquina de estados, vista de actividad y vista de interacción.

Figura 7. *Vistas de comportamiento dinámico de UML*

Comportamiento dinámico	
Vista de interacción	Describe secuencias de intercambio de mensajes entre los roles que **Diagrama de secuencia:** muestra un conjunto de mensajes, dispuestos en una secuencia temporal. Cada rol en la secuencia se muestra como una línea de vida, es decir una línea vertical que representa el rol durante cierto plazo de tiempo

Comportamiento dinámico	
	Diagrama de colaboración: Modela los objetos y los enlaces significativos dentro de una interacción. Un uso del diagrama de colaboración es mostrar la implementación de una operación. La colaboración muestra los parámetros y las variables locales de la operación
Maquina de estados	modela las posibles historias de vida de un objeto de una clase. Una maquina de estados contiene los estados conectados por transiciones. Cada Las maquinas de estados se pueden utilizar para escribir interfaces de usuario, controladores de dispositivo y otros subsistemas reactivos
Vista de actividades	Un grafo de actividad es una variante de una maquina de estados, que muestra las actividades de computación implicadas en la ejecución de un calculo. Representa un paso en el flujo de trabajo o en la ejecución de una operación.

Tanto los diagramas de secuencia como los diagramas de colaboración muestran interacciones, pero acentúan aspectos diferentes ya que un diagrama de secuencia muestra secuencias en el tiempo como dimensión geométrica, pero las relaciones entre los roles son implícitas y en un diagrama de colaboración se muestran las relaciones entre roles geométricamente y relaciona los mensajes con las relaciones, pero las secuencias temporales están menos claras, porque vienen dadas por los números de secuencia.

La **gestión del modelo** describe la organización de los propios modelos en unidades jerárquicas. El paquete es la unidad genérica de organización para los modelos. La vista de gestión del modelo cruza las otras vistas y las organiza para el trabajo de desarrollo y el control de configuración.

3.2.2. Etapas y actividades en el desarrollo de software basado en UML

Análisis de Requerimientos
Diseño del sistema
Diseño detallado
Implementación y pruebas

3.2.3. Elementos notacionales de UML

Los elementos notacionales que presenta el UML pretenden ser un lenguaje común para el modelamiento de cualquier sistema y se agrupan en tipos de diagrama:

Diagrama de Clases
Diagrama de Objetos
Diagrama de Casos de Uso
Diagrama de Secuencia
Diagrama de Colaboración
Diagrama de Estados
Diagrama de Actividades
Diagrama de Componentes
Diagrama de Ejecución

3.3. PROCESAMIENTO DE IMÁGENES DIGITALES

3.3.1. Formación de la Imagen

Antes de iniciar ningún estudio sobre técnicas de procesamiento y/o análisis de imágenes digitales debemos preguntarnos cuál es la base de la formación de dichas

imágenes. De forma genérica podemos asociar el concepto de imagen al de un " mapa espacial" o espacio-temporal que sobre una determinada información nos produce un tipo de sensor. Ejemplos de este tipo de "mapas" son nuestras percepciones de una determinada situación o escena a partir de nuestros sistemas sensoriales (vista, oído, tacto, gusto, y olfato). Ahora nos restringiremos al caso de la percepción visual.

Para centrar aun más nuestro estudio nos preguntaremos sobre el significado real de lo que significa ver o adquirir información a través de un sistema sensorial de tipo visual. Si nos centramos en el modelo visual humano asociaremos el concepto de ver con el de percibir una señal luminosa con una intensidad mínima y en un rango de frecuencia espectral dado. Sin embargo hoy en día son bien conocidas las posibilidades de obtener imágenes a partir de sensores que trabajen en condiciones muy distintas de iluminación a las que es sensible el ojo humano, p.e. el infrarrojo, rayos X, etc. Así pues, la posibilidad de formar imágenes debemos asociarla al tipo de sensor usado y a las posibilidades de que dicho sensor sea capaz de captar y decodificar la información que le llega. Este punto de vista nos sitúa en una situación más general de lo que sería el simple estudio de las imágenes en color captadas por nuestros ojos.

Otra pregunta importante es "¿cuanta información podemos obtener sobre una determinado escena cuando usamos un determinado sistema sensorial?”.

Para contestar adecuadamente debemos hacer uso de las propiedades físicas de la interrelación de la materia con las ondas electromagnéticas como vehículo de transmisión. Un hecho conocido de la física básica es que el espectro electromagnético de una sustancia es lo que podríamos llamar la huella digital de la misma. Así pues, si observamos una escena usando un sensor que es sensible a unas determinadas bandas de frecuencia, la información que obtendremos será la gráfica de valores de la respuesta espectral de dicha escena en las frecuencias del espectro a las que es sensible el sensor. Por ejemplo, en el caso de humanos, los ojos distinguen solamente tres bandas del espectro electromagnético asociadas a los colores rojo, azul y verde.

Ya que los rangos de sensibilidad de los sensores son muy pequeños en comparación con la información suministrada por el total del espectro electromagnético, el problema de identificar una determinada materia o sustancia a partir del conjunto de valores espectrales en los que es sensible un sensor se presenta como un problema en general difícil. Para paliar este problema habrá que hacer uso o de información complementaria de tipo estructural extraída de la propia imagen o de información experimental previa o de la experiencia. En el caso de identificación de objetos será la geometría de la forma de los mismos la información complementaria más relevante.

3.3.2. La imagen digital

3.3.2.1. Función de imagen

Una función de imagen es una función en dos dimensiones que asigna un valor correspondiente a un nivel de intensidad a cada par de coordenadas x, y:

$$f(x,y) = i(x,y)\,r(x,y)$$

Ecuación 1. *Función de imagen*

donde:

i(x,y) : Función de iluminación .

r(x,y) : Función de reflectancia.

Los niveles de intensidad que puede asumir la función de imagen monocromática f(x,y) , en cada par de coordenadas , se conocen como niveles de gris l .

Para que la función de imagen pueda ser manipulada adecuadamente por medio de una unidad de cómputo, se requiere que dicha función sea discretizada tanto en amplitud (l) como en distribución espacial (x,y) .

$$f(i,j) = \sum_{m=0}^{N-1}\sum_{n=0}^{N-1} f(m,n)\,\delta(i-m, j-n)$$

Ecuación 2. *Función de imagen discreta.*

3.3.2.2. Tipos de sensores e imágenes

Muchos y muy diferentes son los sensores de tipo opto-electrónico que existen hoy en día para la adquisición de imágenes en los distintos campos de aplicación científica y técnica. Cada modalidad de sensor responde en su diseño a la necesidad de adquirir una determinada información acerca de los objetos que se observan que en general serán función del estudio o uso que queramos hacer de ellos. La siguiente tabla resume la gran mayoría de tipos de imágenes usados en las aplicaciones así como los sensores usados para captarlas.

3.3.2.3. Efectos del muestreo y la cuantización

Las dos principales causas que producen perdida de información cuando capturamos una imagen son la naturaleza discreta de los pixeles de la imagen y el rango limitado de valores de intensidad luminosa que somos capaces de medir en cada pixel.

El muestreo de una imagen tiene el efecto de reducir la resolución espacial de la misma. En una imagen digitalizada, el número de renglones y columnas correspondiente a la rejilla de muestreo, ocasionará una mayor o menor definición de la imagen, tal como se puede apreciar en los siguientes ejemplos. De igual forma, el número de bits asignado a los niveles de cuantización ocasionará un efecto como el mostrado.

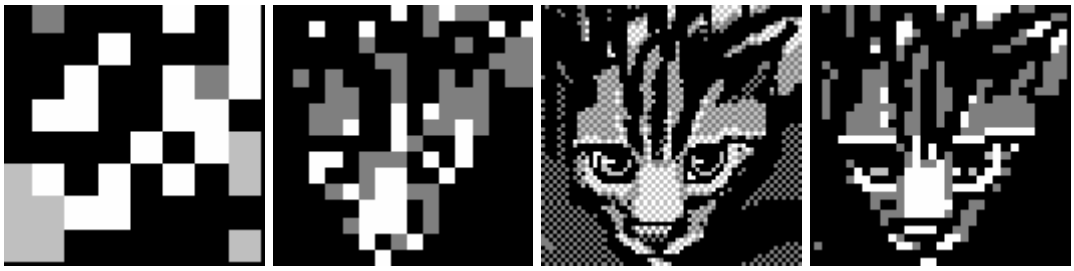


Figura 8. *Imágenes en resoluciones 8x, 16x16, 32x32 y 64x64.*

El efecto de cuantización viene dado por la imposibilidad de tener un rango infinito de valores de medida para la intensidad de brillo de los pixeles. Como ya hemos comentado antes, la tecnología actual permite en casos extras llegar hasta 10 bits de información, aunque lo general es tener 8 bit, o equivalentemente 256 niveles de gris para codificar este valor lumínico.

Figura 9. *Imágenes de 4 niveles de gris (2 bits) y 2 Niveles de gris (4 bits)*

No existen criterios que nos permitan decidir el número óptimo de pixeles y de bits con los que hacer un muestreo una determinada imagen. Distintos estudios experimentales, usando la opinión de personas, han llegado a la conclusión de que existen determinados valores de umbral por encima de los cuales no se aprecia una significativa ganancia pero por debajo de ellos si se aprecia una perdida efectiva en la imagen. En cualquier caso y dado el estado de desarrollo actual de los métodos automáticos de análisis e interpretación de imágenes la conclusión de obtener una imagen con la mayor resolución y calidad posible parece por ahora la única viable.

3.3.3. Procesamiento en el dominio del espacio

Se entiende por procesamiento en el dominio del espacio, la realización de operaciones directamente sobre el valor de los pixeles que forman la imagen de entrada. En esta categoría quedan comprendidos los siguientes tipos de procesamiento: Operaciones lógico-aritméticas, transformaciones punto a punto, transformaciones geométricas y convolución bi-dimensional.

3.3.3.1. Algunas Relaciones Básicas entre Pixeles

En esta sección se considerarán alunas relaciones básicas pero importantes entre los pixeles de una imagen digital. Como se mencionó una imagen se indica por f(x,y). Cuando se haga referencia a un pixel en particular, se emplearán letras minúsculas, como p y q. Un subconjunto de pixeles de f(x,y) se indicará mediante S.

3.3.3.1.1. Vecinos de un Pixel

Un pixel p de coordenadas (x,y) tiene cuatro vecinos horizontales y verticales cuyas coordenadas vienen dadas por:
(x + 1, y), (x – 1, y), (x,y + 1), (x,y – 1)

Este conjunto de pixel, denominado los 4 – vecinos de p, se representa por N_4 (p). Cada pixel está a una unidad de distancia de (x,y) , y algunos de los vecinos de p caen fuera de la imagen digital si (x,y) está en el borde de la imagen.

Los cuatro vecinos en diagonal del p tienen las coordenadas:
(x + 1, y + 1), (x + 1, y – 1), (x – 1, y + 1), (x – 1, y – 1)

y se representan por N_8(p). Estos puntos, junto a los 4 – vecinos, se denominan los 8 – vecinos de p, y se representan por N_8(p). Al igual que antes, algunos puntos de N_D(p) y N_8(p) caen fuera de la imagen si (x,y está en el borde de la misma.

3.3.3.1.2. Conectividad

La conectividad entre pixel es un concepto importante empleado para establecer los límites de los objetos y los componentes de áreas en una imagen. Para determinar si dos pixel están conectados, debe determinarse si son adyacentes en algún

sentido (como ser 4 – vecinos) y si sus niveles de gris cumplen un criterio especificado de similitud (como ser iguales). Por ejemplo, en una imagen binaria con valores 0 y 1, dos pixel pueden ser 4 – vecinos pero no estarán conectados a menos que tengan el mismo valor.

Sea V el conjunto de valores de nivel de gris empleados para definir la conectividad; por ejemplo, en una imagen binaria, se tendrá V = [1] para la conectividad entre pixel con valor 1. En una imagen con escala de grises, para la conectividad entre pixel con un rango de valores de intensidad de, por ejemplo, 32 a 64, se tiene V = [32,33,...,63,64]. Se considera tres tipos de conectividad:

4 – conectividad. Dos pixel p y q con valores dentro de V están 4 – conectados si q pertenece a $N_4(p)$.
8 – conectividad. Dos pixel p y q con valores dentro de V están 8 – conectados si q pertenece a $N_8(p)$.
m – conectividad (conectividad mixta). Dos pixel p y q con valores dentro de V están m – conectados si
q pertenece a $N_4(p)$, o bien
q pertenece a $N_D(p)$ y además el conjunto $N_4(p) \cap N_4(p)$ es vacío (Este es el conjunto de pixel que son 4 – vecinos de p y de q cuyos valores están en V).

Un pixel p es adyacente de un pixel q si están conectados. Se puede definir 4-, 8- o m-adyacencia, dependiendo del tipo de conectividad especificada. Dos subconjuntos de la imagen S_1 y S_2 son adyacentes si algún pixel de S_1 adyacente a algún pixel de S_2.

Un camino desde el pixel p de coordenadas (x,y) al pixel q de coordenadas (s,t) es una sucesión de diversos pixel de coordenadas:

$(X_0, Y_o), (X_1, Y_1),..., (X_n, Y_n)$

donde $(X_0, Y_o) = (X, Y)$ y (X_n, Y_n), = (s,t), (X_i, Y_i) es adyacente a (X_{i-1}, Y_{i-1}), $1 \leq i \leq n$, y n es la longitud del camino. Así podemos definir 4-, 8- y m-caminos, dependiendo del tipo de adyacencia especificado.

Si p y q son pixeles de un subconjunto S especificado de la imagen, se dirá que p está conectado con q dentro de S si existe un camino desde p hasta q que consista totalmente de pixeles de S. Para cualquier pixel p dentro de S, el conjunto de pixeles de S conectados a p se denomina componente conexa de S. Por tanto, cualquier par de pixeles de una misma componente conexa están conectados entre sí, y componentes conexas distintas son disjuntas.

La capacidad de asignar etiquetas diferentes a las distintas componentes conexas disjuntas de una imagen es de importancia fundamental en el análisis automatizado de la imagen. En la sección siguiente se desarrollará un sencillo procedimiento secuencial de etiquetado de componentes conexas que actúa sobre dos filas de una imagen binaria cada vez.

3.3.3.1.3. Etiquetado de Componentes Conexas

Imagínese el barrido de una imagen pixel, de izquierda a derecha y de arriba abajo, y supóngase que, por el momento, estamos interesados en componentes 4 – conexas. Sea p el pixel en cada paso del proceso del barrido y sean r y t los vecinos superior e izquierdo de p, respectivamente. La naturaleza de la secuencia de barrido asegura que cuando se llega a p los puntos r y t ya han sido encontrados (y etiquetados si fuesen uno s1).

Con los conceptos anteriormente establecidos, considérese el procedimiento siguiente: si el valor de p es 0, simplemente se continúa hasta la siguiente posición de barrido. Si el valor de p es 1, se examinan r y t. Si ambos son 0, se asignan una nueva etiqueta a p (por lo que hasta ahora se sabe, basados en la información presente, es la primera vez que se ha encontrado esta componente conexa). Si solo uno de los dos vecinos es igual a 1, se asigna su etiqueta a p. Si ambos son 1

y tienen la misma etiqueta, se asigna esta etiqueta a p. Si ambos son 1 y tienen etiquetas diferentes, se asignan una se las etiquetas a p y una nota de que ambas etiquetas son equivalentes (es decir, los puntos r y t están conectados a través de p). Al final del barrido todos los puntos con valor igual a 1 han sido etiquetados, aunque algunas de estas etiquetas puedan ser equivalentes. Todo lo que necesitamos hacer ahora es clasificar todos los pares de etiquetas equivalentes en clases de equivalencia, asignar una etiqueta diferente a cada clase, y luego dar una segunda pasada a través de la imagen reemplazando cada etiqueta por la etiqueta asignada a su clase de equivalencia.

Para etiquetar componentes 8 – conectados debemos proceder de la misma forma, pero ahora los vecinos diagonales superiores, indicados por q y s, también han de ser examinados. La naturaleza de la secuencia de barrido asegura que estos vecinos hayan sido procesados en el momentos que se llegue al punto p. I p es 0 será necesario moverse a la siguiente posición de barrido. Si p es 1 y los restantes cuatro vecinos son todos 0, ha de asignarse una nueva etiqueta a p. Si únicamente uno de los vecinos es 1, entonces ha de asignarse su etiqueta a p. Si dos o más de los vecinos son 1, entonces ha de asignarse una de sus etiquetas a p y ha de hacerse una anotación oportuna de las equivalencias. Tras completar el barrido de la imagen, se han de convertir los pares de etiquetas equivalentes en clases de equivalencia, asignando una única etiqueta para cada clase. Finalmente se realiza un segundo barrido de la imagen, cambiando cada etiqueta por la etiqueta asignada a su clase de equivalencia.

3.3.3.1.4. Medidas de Distancia

Para los pixeles p, q y z de coordenadas (x, y), (s, t) y (u, v) respectivamente, D es una función distancia o una métrica si:

$D(p, q) \geq 0$ ($D(p, q) = 0$ si y solo si $p = q$),
$D(p, q) = D(q, p)$, y
$D(p, z) \leq D(p, q) + D(q, z)$.

La distancia euclídea entre p y q está definida por

$$D_e(p,q) = [(x - s)^2 + (y - t)^2]^{1/2}$$

Ecuación 3. *Distancia entre dos puntos de una imagen*

Para esta medida de distancia, los pixeles que están a una distancia menos o igual que algún valor r de (x,y) son los puntos contenidos en un círculo de radio r con centro en (x, y).

La distancia D_4 (denominada también distancia <<city – block>>) entre dos puntos p y q se define como:

$$D_4(p,q) = |x - s| + |y - t|$$

Ecuación 4. *Distancia <<city – block>> entre pixeles*

En este caso los pixeles que están a una distancia (x,y), D_4 menor o igual que un determinado valor r forma un rombo centrado en (x,y). Por ejemplo, los pixels con una distancia $D_4 \leq 2$ desde (x,y) (el punto central) forman los siguientes contornos de distancia constante:

```
      2
    2 1 2
  2 1 0 1 2
    2 1 2
      2
```

Figura 10. *Distancia <<city – block>> entre pixeles*

Los pixeles con distancia $D_4 = 1$ son los 4 – vecinos de (x,y), la distancia D_8 (también llamada distancia de tablero de ajedrez) entre p y q se define como:

$$D_8(p,q) = \max(|x - s|, |y - t|)$$

Ecuación 5. *Distancia de tablero de ajedrez entre pixeles*

En este caso los pixels a una distancia D_8 de (x,y) menor o igual que un cierto valor r forma un cuadrado centrado en (x,y). Por ejemplo, los pixeles a una distancia $D_8 \leq 2$ de (x,y) (el punto central) forman los siguientes contornos de distancia constante:

2 2 2 2 2
2 1 1 1 2
2 1 0 1 2
2 1 1 1 2
2 2 2 2 2

Figura 11. *Distancia de tablero de ajedrez entre pixeles*

Los pixeles con una distancia $D_8 = 1$ son los 8 – vecinos de (x,y). Las distancia D_4 entre dos puntos p y q es igual a la longitud del 4 – camino más corto entre ambos puntos. Lo mismo se aplica para la distancia D_8. De hecho, se puede considerar tanto la distancia D_4 como la distancia D_8 entre p y q sin importar si existe un camino que conecte ambos puntos puestos que las definiciones de distancia implican únicamente alas coordenadas de ambos puntos. Sin embargo, para la m – conectividad, el valor de la distancia (la longitud del camino) entre dos pixeles depende de los valores de los pixeles que constituyen el camino y los de sus vecinos. Por ejemplo, considérese la siguiente distribución de pixeles y supóngase que p, p_2 y p_4 tienen un valor que p_1 y p_3 pueden tener un valor 0 ó 1:

p_3 p_4
p_1 p_2
p

si solo se permite la conectividad entre pixeles de valor 1, y p_1 y p_3 son igual a 0 entonces la m – distancia entre p y p_4 es 2. Si p_1 o p_3 son 1, entonces la distancia es 3. Si p_1 y p_3 son ambos 1, la distancia es 4

3.3.3.2. Operaciones Lógico-Aritméticas

Una vez que las imágenes han sido digitalizadas y guardadas en forma de matriz en el correspondiente sector de memoria (buffer) de la computadora, resulta inmediata

la posibilidad de realizar operaciones aritméticas y lógicas entre ellas. Estas operaciones son realizadas punto a punto entre pixeles correspondientes. Si bien el resultado de la suma de dos imágenes puede ser almacenado en un sector de memoria adicional formado por los valores reales obtenidos de la operación, es evidente que al momento de presentarlos en pantalla en forma de niveles de gris, podría excederse el rango de niveles de gris disponible. es por esto que en la mayoría de programas de procesamiento digital de imágenes disponibles, se presentan las imágenes normalizadas con el valor máximo de los elementos de la imagen presentado en blanco en pantalla, y el valor mínimo presentado en negro.

Las operaciones lógico aritméticas entre imágenes se definen a partir de las mismas operaciones entre pixeles correspondientes

Cuando el resultado de la operación excede el rango de niveles de gris o rango dinámico de la imagen es práctica común escalar estos valores para que ocupen completamente dicho rango. Esto es realizado únicamente con el objeto de observar resultados en pantalla, aun cuando se puedan conservar en memoria los valores reales. En la siguiente figura se muestra el resultado obtenido al realizar algunas operaciones lógico-aritméticas entre dos imágenes.

SUMA	$f(i,j) = g(i,j) + h(i,j)$
RESTA	$f(i,j) = g(i,j) - h(i,j)$
MULTIPLICACION	$f(i,j) = g(i,j) \times h(i,j)$
DIVISION	$f(i,j) = g(i,j) \div h(i,j)$
AND	$f(i,j) = g(i,j) AND\ h(i,j)$
OR	$f(i,j) = g(i,j) OR\ h(i,j)$
COMPLEMENTO	$f(i,j) = NOT\ g(i,j)$

Ecuación 6. *Operaciones Aritmético Lógicas*

Figura 12. *En su respectivo orden: (a)Imagen "flor" (b) Imagen "cuadro" (c) Suma de "flor" y "cuadro" (d) Resultado del operador OR entre "flor" y "cuadro" (e) AND entre "flor" y "cuadro"*

3.3.3.3. Transformaciones punto a punto.

Este tipo de procesamiento consiste fundamentalmente en la reasignación de niveles de gris punto a punto, a través de una función de transformación de la forma s = T (r). Esta función se encuentra representada en la figura. En esta función, los niveles de gris de la imagen de entrada están representados por la variable r, mientras que la variable s representa los niveles de gris de la imagen
de salida.

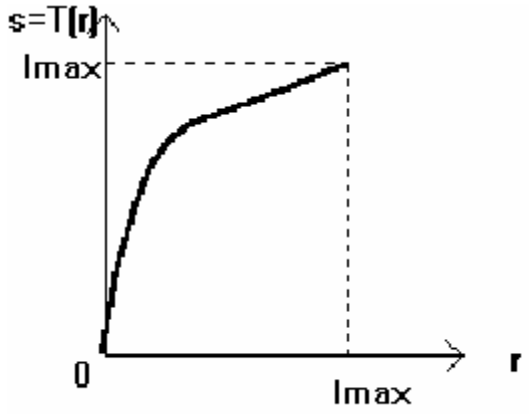


Figura 13. *Función de transformación de niveles de gris punto a punto.*

Ejemplo: Inversión de contraste.
La función de transformación correspondiente a una inversión de contraste, está dada por una recta con pendiente negativa como la que se muestra en la figura. Es evidente que los niveles de gris bajos correspondientes a zonas obscuras de la

imagen de entrada se verán transformados a los niveles de gris mas altos o claros de la imagen de salida y viceversa.

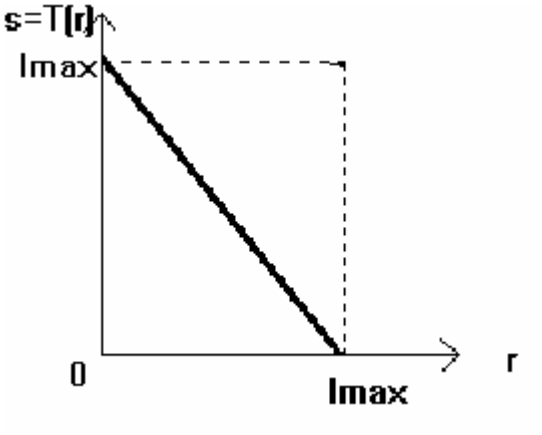


Figura 14. *Inversión de contraste*

3.3.4. Dominio de la frecuencia espacial

3.3.4.1. Convolución

Este tipo de procesamiento consiste fundamentalmente en realizar una transformación que involucre a un determinado número de puntos en la vecindad del pixel central, en vez de hacer la transformación pixel a pixel como fué el caso en la sección anterior. Esto se realiza a través de una operación de convolución entre dos funciones bi-dimensionales, la primera formada por la imagen por procesar y la segunda constituida por la mascarilla de convolución, cuyos coeficientes definen el tipo de procesamiento por realizarse. La convolución entre dos funciones bi-dimensionales tiene sentido cuando se considera que la transformación involucrada reúne las características de linealidad, homogeneidad e invarianza al corrimiento. En la figura se representa esquemáticamente este tipo de procesamiento.

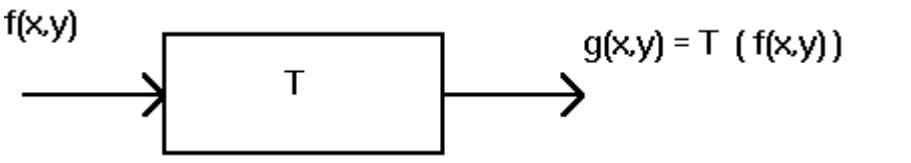


Figura 15. *Representación esquemática de una transformación en el dominio del espacio, aplicada a la imagen de entrada f(x,y)*

Un sistema lineal, homogéneo e invariante al corrimiento puede ser caracterizado a través de su respuesta al impulso. La función respuesta al impulso se define como:

$$h(x,y) = T\left[\delta(x,y)\right]$$

Ecuación 7. *Función respuesta al impulso*

En forma general, una imagen digitalizada se puede representar como una suma de funciones impulso localizadas en el plano xy, con magnitud proporcional al nivel de gris en las coordenadas correspondientes, tal como se expresa en la ecuación **5.**

$$f(x,y) = \sum_{m=0}^{M} \sum_{n=0}^{N} f(m,n)\delta(x-m, y-n)$$

Ecuación 8. *Suma de funciones impulso.*

La imagen a la salida es obtenida al aplicar la función de transformación T tal como se indica:

$$g(x,y) = T\left[f(x,y)\right]$$

$$g(x,y) = T\left[\sum_{m=0}^{M} \sum_{n=0}^{N} f(m,n)\delta(x-m, y-n)\right]$$

Ecuación 9. *Función de transformación T (a).*

Si T satisface la propiedad de linealidad, la transformación de una suma de funciones corresponde con la suma de las transformaciones individuales, entonces, la ecuación 7 se puede expresar como:

$$g(x,y) = \sum_{m=0}^{M} \sum_{n=0}^{N} T\left[f(m,n)\delta(x-m, y-n)\right]$$

Ecuación 10. *Función de transformación T (b).*

Si el sistema satisface la propiedad de homogeneidad, la transformación se puede expresar como:

$$g(x,y) = \sum_{m=0}^{M} \sum_{n=0}^{N} f(m,n) T\left[\delta(x-m, y-n)\right]$$

Ecuación 11. *Función de transformación T (c).*

De acuerdo con la definición de la respuesta al impulso del sistema, la imagen de salida se obtiene a través de la operación:

$$g(x,y) = \sum_{m=0}^{M} \sum_{n=0}^{N} f(m,n)\ h(x-m, y-n)$$

$$g(x,y) = f(x,y) * h(x,y)$$

Ecuación 12. *Función de transformación (d).*

Estas dos últimas ecuaciones corresponden precisamente con la operación de convolución en dos dimensiones entre la función de imagen de entrada f(x,y) y la respuesta al impulso del sistema, representada como se muestra:

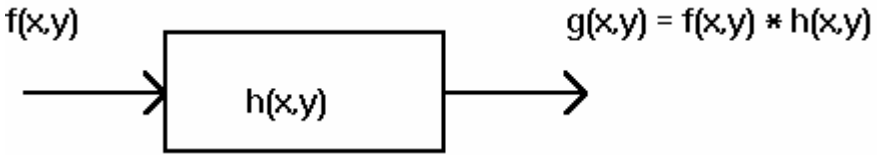


Figura 16. *la operación de convolución en dos dimensiones*

El tipo de procesamiento estará definido entonces, por los valores de los coeficientes en la matriz correspondiente a la respuesta al impulso del sistema, la cual será referida en adelante como la ***mascarilla de convolución****.* En la figura **12** se representan esquemáticamente las operaciones involucradas en la convolución

en dos dimensiones de una función de imagen con una mascarilla de convolución de dimensión 3X3; tal como se observa, el pixel central en la imagen de salida se forma al multiplicar el valor de los pixeles dentro de la vecindad formada por el cuadro de 3X3 por el correspondiente coeficiente de la mascarilla de convolución. Esta mascarilla es entonces deslizada a todo lo largo y ancho de la imagen de entrada para obtener el total de pixeles en la imagen de salida.

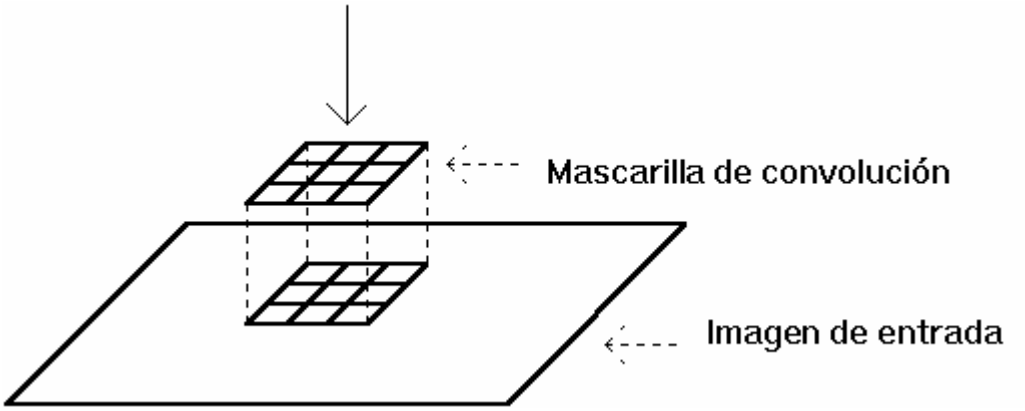


Figura 17. *Convolución en dos dimensiones.*

3.3.5. Filtrado espacial

El principal objetivo de las técnicas de mejoramiento de imagen es procesar una imagen con el fin de hacerla más adecuada para una determinada aplicación o procesamiento posterior. Depende por tanto del problema específico a resolver el que se emplee una u otra técnica. Los métodos de mejora de imagen se pueden dividir en dos campos diferentes: métodos en el dominio frecuencial y métodos en el dominio espacial. Los primeros se basan en modificar la transformada de Fourier de la imagen, mientras que los segundos se basan en manipulaciones directas sobre los pixeles de la imagen.

Una característica común a todos los tipos de datos ráster es la llamada "frecuencia espacial", que define la magnitud de cambios de los datos por unidad de distancia en una determinada zona de la imagen. Areas de la imagen con pequeños cambios o con transiciones graduales en los valores de los datos se denominan áreas de bajas frecuencias (como por ejemplo la superficie de una masa de agua en reposo).

Areas de grandes cambios o rápidas transiciones se conocen como áreas de altas frecuencias (por ejemplo suelo urbano con densas redes de carreteras). Así, los filtros espaciales se pueden dividir en tres categorías:

3.3.5.1. Filtros pasa bajos

Enfatizan las bajas frecuencias, suavizando las imágenes y suprimiendo ruidos. Se trata de asemejar el ND de cada píxel al ND de los pixeles vecinos, reduciendo la variabilidad espacial de la imagen. Ello produce un emborronamiento de los bordes, perdiéndose en nitidez visual de la imagen, pero ganando en homogeneidad.

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

Figura 18. *Ejemplo de un kernel de filtro pasa-bajo.*

Vemos que lo que se realiza es una media aritmética de los nueve pixeles que componen la ventana de filtrado, con lo que se reducen los espurios y la variabilidad de la imagen.

Otro tipo de filtro pasa-bajo es el que aplica la mediana en vez de la media. Es el llamado **filtro de mediana**, y presenta la ventaja de que como medida estadística, la mediana es menos sensible a valores extremadamente desviados y se modifican menos los valores originales, ya que la mediana es en principio, uno de los valores concretos de la ventana de filtrado.

En un filtro de mediana (median filter), el valor de cada pixel es substituido por el valor a la mitad de los niveles de gris dentro de una vecindad, previamente ordenados (sort).

Este método es particularmente efectivo para remover ruido impulsivo, y cuando los cambios fuertes en nivel de gris característicos de las orillas de un objeto dentro de una imagen se desean preservar. El tamaño y forma de la región en la cual se realiza este análisis estadístico es definido previamente por el usuario.

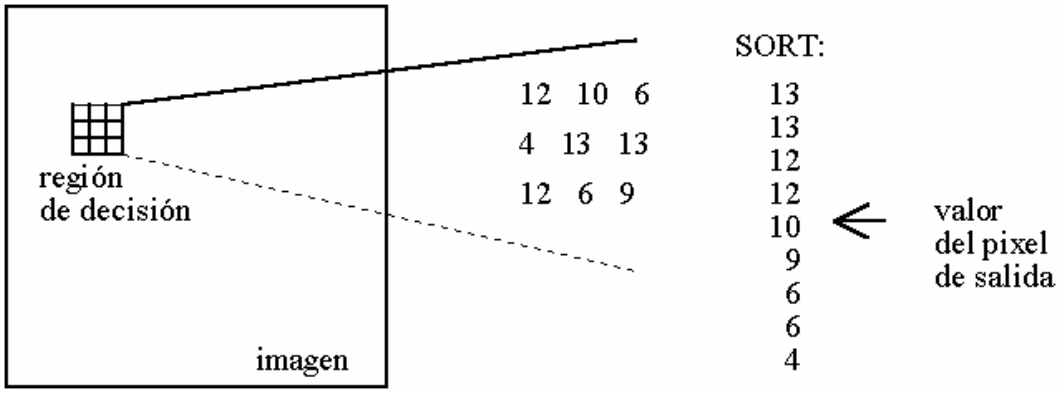


Figura 19. *Filtro de mediana.*

La siguiente figura muestra el resultado de aplicar un filtrado de mediana con una región de decisión formada por un cuadro de 3X3 pixeles, a una imagen con ruido:

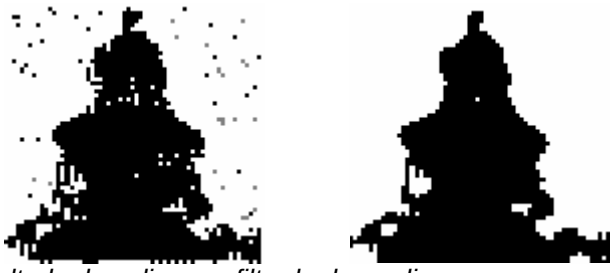


Figura 20. *Resultado de aplicar un filtrado de mediana.*

3.3.5.2. Filtros pasa altos

Enfatizan las altas frecuencias, para mejorar o afilar las características lineales como carreteras, fallas, o límites en general. Realizan por tanto el efecto contrario a los filtros pasa-bajos, eliminando estos las bajas frecuencias.

$$\begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix}$$

Figura 21. *Ejemplo de un kernel de filtro pasa-alto.*

Otra forma de obtener una imagen así filtrada es sustraer a la imagen original, la misma imagen filtrada pasa-bajos. Es lógico ya que si a la imagen le restamos los componentes de baja frecuencia, nos quedaremos con las de alta frecuencia.

3.3.5.3. Filtros detectores de bordes

Realizan otro tipo de operaciones con los datos, pero siempre con el resultado de enfatizar los bordes que rodean a un objeto en una imagen, para hacerlo más fácil de analizar. Estos filtros típicamente crean una imagen con fondo gris y líneas blancas y negras rodeando los bordes de los objetos y características de la imagen.

3.3.6. Morfología Matemática.

En este tipo de filtros, las operaciones entre pixeles están basadas en los conceptos de morfología matemática.

Morfología matemática (tratado de formas) es una herramienta para el procesamiento y análisis de imágenes digitales, que ha sido desarrollada y explorada en las dos últimas décadas.

Su diferencia fundamental con los filtros lineales radica en el hecho de que el filtrado no se realiza a través de discriminación del contenido armónico espectral de la imagen, sino directamente en el dominio del espacio y en función de su forma.

El lenguaje de la morfología matemática binaria es el de la teoría de conjuntos. Los conjuntos en morfología matemática representan las formas presentes en imágenes

binarias o de niveles de gris. El conjunto de todos los pixeles blancos en una imagen en blanco y negro (binaria) constituye una descripción completa de la imagen.

Los puntos en un conjunto sobre los que se aplica la transformación son el conjunto de puntos seleccionado y el complementario el no seleccionado. En las imágenes binarias los puntos seleccionados son los que no pertenecen al fondo.

Las operaciones primarias morfológicas son la erosión y la dilatación. A partir de ellas podemos componer las operaciones de apertura y clausura. Son estas dos operaciones las que tienen mucha relación con la representación de formas, la descomposición y la extracción de primitivas.

3.3.6.1. Operaciones básicas sobre conjuntos

Sean A y B son conjuntos en un n-espacio E^n con elementos $a = (a_1,..., a_n)$ y $b = (b_1,..., b_n)$ respectivamente siendo ambos n-tuplas.

La traslación de A por x En que se nota Ax se define como:
Ax = {c| c = a + x, para algún a A}.

La reflexión de B notada se define como
B^= {x| x = - b, para algún b B}.

Por último la diferencia de dos conjuntos A y B, notada A - B, se define mediante
A - B = {x| x A, x B}

3.3.7. Fundamentos del color

Aunque los procesos seguidos por el cerebro humano para percibir el color no están totalmente comprendidos, la naturaleza física del mismo puede ser expresada con bases formales soportadas por resultados teóricos y experimentales.

En 1666, Sir Isaac Newton descubrió que un haz de luz solar atravesando un prisma de cristal, se transformaba en un haz emergente de luz no blanca, sino consistente en un espectro continuo de colores que iban desde el violeta en un extremo, hasta el rojo en el extremo opuesto.

Básicamente, el color con el que percibimos un objeto viene determinado por la naturaleza de la luz reflejada por dicho objeto. Un cuerpo que refleja luz de forma uniforme en todo el rango del espectro visible (que es una estrecha banda del espectro electromagnético), aparece blanco al observador, mientras que si refleja más en algún rango estrecho del espectro visible, aparecerá al observador con ese color concreto. Por ejemplo, los objetos verdes reflejan luz con longitudes de onda principalmente en el rango de 500 a 570 μm (nanómetros), mientras que absorben la energía electromagnética en el resto de longitudes de onda del visible.

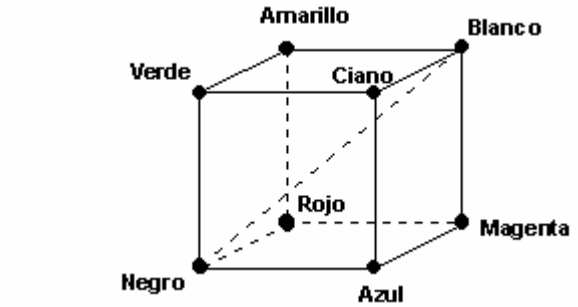
El rango de colores que podemos apreciar, es el resultado de mezclas de luz de diferentes longitudes de onda. Se ha comprobado que ciertas longitudes de onda del rojo (R), verde (V) y azul (A), cuando se combinan en diferentes proporciones (intensidades) producen mayor rango de colores que cualquier otra combinación de tres colores. Por ello, a esos tres colores se les denomina colores primarios. Aunque se ha mal interpretado que con esos tres colores se puede conseguir cualquier otro color, eso no es cierto. Ciertos colores no pueden ser obtenidos por mera combinación de RVA (RGB por sus siglas en ingles). Con propósitos de unificación, se han establecido las siguientes longitudes de onda concretas para los colores primarios: azul: 435.8 μm, verde: 546.1 μm y rojo: 700 μm.

Los colores primarios pueden combinarse para producir colores secundarios: magenta (azul + rojo), cian (verde + azul) y amarillo (rojo + verde).

Combinando los tres colores primarios, o un secundario con su correspondiente primario, en las proporciones adecuadas, se obtiene luz blanca.

También es importante distinguir entre colores primarios de luz y colores primarios de pigmentos o colorantes. En estos últimos, un color primario se define como aquel que resta o absorbe un color primario de luz y refleja o transmite los otros dos. Así, los colores primarios de pigmentos son el magenta, cian y amarillo, mientras que los secundarios son rojo, azul y verde. Una combinación adecuada de los tres colores primarios de pigmentos, o de un primario con su secundario, dará lugar al negro.

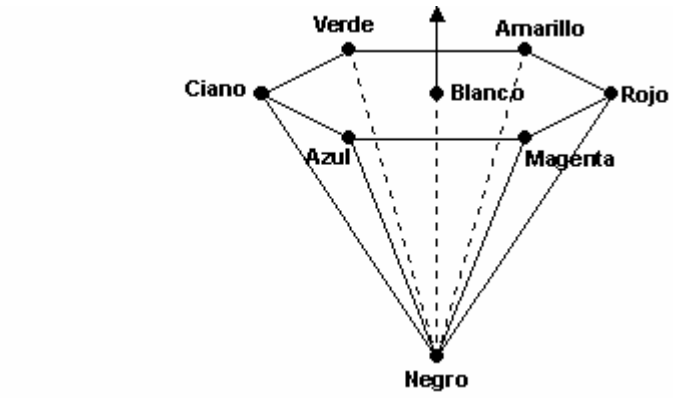
Una forma de representar un sistema de colores de este tipo, es mediante un cubo de colores donde los ejes representan los colores primarios, y combinaciones de distintos valores de esos colores primarios dan el resto de colores (aunque ya he comentado antes que esto no es del todo cierto). La siguiente figura muestra un



cubo de colores, representando los lugares donde se sitúan los colores más puros.

Figura 22. *Representación en perspectiva del modelo 3D para el sistema de colores RVA, RGB por sus siglas en ingles.*

Pero este no es el único sistema en el que se puede representar el color. Existen otros como el de tono, saturación e intensidad (TSI o HSI por sus siglas en ingles), que se define de la siguiente forma: tono es el color, como 'rojo' o 'naranja'. Está directamente asociado con una longitud de onda concreta, y se mide en grados, de 0º a 360º. Saturación es una medida del grado de mezcla de ese tono con otros tonos de alrededor, y se mide en tanto por uno. Intensidad es precisamente eso, el grado de intensidad de ese color, y se mide de cero hasta infinito. Este sistema de



color también se puede 'visualizar' de un manera gráfica, como muestra la siguiente figura.

Figura 23. *Representación en perspectiva del modelo 3D para el sistema de colores TSI (HSI por sus siglas en ingles).*

3.4. TRAZADORES CUBICOS Y CURVAS PARAMETRICAS

3.4.1. Interpolación de Trazadores Cúbicos

La aproximación de una función arbitraria es posible mediante un polinomio en un intervalo cerrado. Sin embargo, la naturaleza oscilatoria de los polinomios de alto grado y la propiedad de que una fluctuación en una parte pequeña del intervalo puede ocasionar importantes fluctuaciones el todo él, limita su utilización.

Un procedimiento alterno consiste en dividir el intervalo en una serie de subintervalos, y en cada subintervalo construir un polinomio generalmente diferente de **aproximación polinómica fragmentaria.**

La aproximación polinómica fragmentaria más simple es la interpolación lineal fragmentaria que consiste en unir una serie de puntos de datos mediante una serie de segmentos de rectas.

La aproximación por funciones lineales ofrece una desventaja: no se tiene la seguridad de que haya diferenciabilidad en los extremos de los subintervalos, lo cual dentro de un contexto geométrico significa que la función interpolante no es “suave” en dichos puntos. A menudo las condiciones físicas indican claramente que se requiere esa condición y que la función aproximante debe ser continuamente diferenciable.

Otro procedimiento consiste en emplear un polinomio fragmentario de tipo Hermite. POr ejemplo, si los valores de la función f y de f’ se conocen en los puntos $x_0<x_1<\ldots<x_n$ podemos emplear un polinomio de Hermite de grado tres en cada uno de los

subintervalos $[x_0, x_1]$, $[x_1, x_2]$,… .$[x_{n-1}, x_n]$. Si queremos determinar el polinomio cúbico de Hermite apropiado en determinado intervalo, basta calcular $H_3(x)$ para ese intervalo. Puesto que los polinomios interpolantes de Lagrange necesarios pata calcular H_3 son de primer grado, podemos hacer el cálculo sin gran dificultad. Sin embargo, para utilizar polinomios fragmentarios de Hermite en la interpolación general, necesitamos conocer la derivada de la función que va ha ser aproximada, lo cual muchas veces no es posible.

El tipo más simple de función de polinomio fragmentario diferenciable en un intervalo entero es la función obtenida de ajustar un polinomio cuadrático entre cada par consecutivo de nodos. Esto se hace construyendo una cuadrática en [X0, X1] que concuerde con la función en X0 y en X1 otra cuadrática en [X1, X2] que concuerde con la función X1 y en X2 y así sucesivamente. U polinomio cuadrático general tiene tres constantes arbitrarias (el término constante, el coeficiente de X y el coeficiente de X^2) y únicamente se requieren dos condiciones para ajustar los datos en los extremos de cada intervalo, por ello, existe una flexibilidad que permite seleccionar la cuadrática de modo que la interpolante tenga una derivada continua en [X0, Xn]. El problema de este procedimiento se presenta cuando hay que especificar las condiciones referentes a la derivada de la interpolante en los extremos X0 y Xn. No hay constantes suficientes para cerciorarse de que se satisfagan las condiciones.

La aproximación polinómica fragmentaria más común utiliza polinomios entre cada par consecutivo de nodos y recibe el nombre de **interpolante de trazadores cúbicos**. Un polinomio cúbico general contiene cuatro constantes; así pues, el procedimiento del trazador cúbico ofrece suficiente flexibilidad para garantizar que el interpolante no solo sea continuamente diferenciable en el intervalo, sino que además tenga una segunda derivada continua en el intervalo. Sin embargo, en la construcción del trazador cúbico no se supone que las derivadas del interpolante concuerden con las de la función, ni siquiera en los nodos.

Dada una función f definida en [a, b] y un conjunto de nodos a= X0<X1… <Xn=b, un interpolante de trazador cúbico S para f es una función que cumple con las condiciones siguientes:

S(x) es un polinomio cúbico denotado Sj(x), en el subintervalo [Xj, Xj+1] para cada j=0, 1,… ,n-1.
S(Xj)=f(Xj) para cada j=0, 1,… ,n-1.
Sj+1(Xj+1)=Sj(Xj+1) para cada j=0, 1,… ,n-2.
S'j+1(Xj+1)=S'j(Xj+1) para cada j=0, 1,… ,n-2.
S''j+1(Xj+1)=S''j(Xj+1) para cada j=0, 1,… ,n-2.
Se satisface uno de los siguientes conjuntos de condiciones de frontera
S''(X0)=S''(Xn)=0 (frontera libre o natural)
S''(X0)=f'(X0) y S''(Xn)=f'(Xn) (**frontera sujeta**)

Aunque los trazadores cúbicos se definen con otras condiciones de frontera, las condiciones anteriores son suficientes en este caso. Cuando se presentan las condiciones de frontera libre el trazador recibe el nombre de trazador natural, y y su gráfica se aproxima a la que adoptaría una varilla larga y flexible si se hiciera pasar por los puntos de datos nodos.

En términos generales, en las condiciones de frontera sujeta se logran aproximaciones más exactas, ya que abarcan más información acerca de la función. Pero para que se cumpla este tipo de condición de frontera, se requiere tener los valores de las derivadas en los extremos o bien una aproximación precisa de ellos.

3.4.2. Curvas Paramétricas de Bezier

La transición suave de un punto a otro, es sin duda un importante problema en el campo de la ingeniería, así podemos encontrar que dependiendo del tipo de problema, existen soluciones diferentes; por ejemplo para un conjunto de puntos $P[x, y]$ puede ser de relevante importancia el que se haga pasar una curva por todos ellos utilizando en este caso un método de interpolación, uno de los más

conocidos es la interpolación polinomial de Lagrange. Otro problema es encontrar la curva que describa el comportamiento general de los datos, sin que forzosamente se pase por todos los puntos, tal es el caso del ajuste de curvas y un método muy conocido es el de Mínimos Cuadrados. Pero cuando el problema es trazar una curva suave desde un punto inicial hasta un punto final, y que ésta sea afectada en su trayectoria por un conjunto de puntos que describen un polígono de apoyo, entonces el problema puede resolverse con las Curvas de Bézier.

Pierre Bézier (1910-1999), francés, ingeniero de profesión, en 1960 resolvió el problema numérico en el trazado de curvas y superficies interpolantes, que parten y llegan a un punto dado, y su trayectoria es afectada por un conjunto de puntos de apoyo. Su utilización desde entonces en la fabricación de vehículos y en la aeronáutica, áreas en la que él incursionó exitosamente, son sin duda algunas de las mas valiosas aplicaciones que se hacen de esta herramienta.

Bézier se ocupó de determinar un método que permitiera unir dos puntos con una curva, que ésta fuera de grado conocido, se controlara y adecuara con el simple hecho de ajustar los puntos de apoyo de la misma y que además su solución permitiera un fácil trazado.

Sus aplicaciones más importantes se encuentran en el área de la aerodinámica, las curvas y superficies de Bézier permiten construir modelos de estudio de una manera sencilla, ésta es la razón principal por la que su aplicación revolucionó la forma de diseñar partes de vehículos, sin embargo su aplicación se extiende a diferentes áreas de la ingeniería y la arquitectura, así se encuentran aplicaciones en el diseño de piezas mecánicas o de elementos arquitectónicos, y en general en donde es importante controlar los puntos de una curva o superficie.

Las curvas de Bézier son en la actualidad herramientas que se encuentran en la mayoría de los paquetes de graficación y dibujo de las computadoras, dependiendo del grado de especialización de este software se puede contar con curvas de mayor o menor grado, pero en todos los casos la transición de un punto a otro con una

curva suave apoyada en un polígono de control se resuelve en la gran mayoría de los casos con las curvas interpolantes de Bézier.

Entonces al definir las curvas de Bézier, decimos que éstas se apoyan en un conjunto de puntos ordenados llamados puntos de control; los cuales afectan la trayectoria de la curva. La curva pasa obligatoriamente solo por el primer y el último punto y no por los demás. Al polígono que se forma al unir secuencialmente los puntos de control se le llama polígono de apoyo o de control. La curva de Bézier que se obtiene como resultado, está en la cubierta convexa del polígono de apoyo.

3.4.2.1. Construcción geométrica de la curva de Bézier

Para iniciar nuestra introducción a la construcción de las curvas de Bézier, propondremos un caso de estudio en donde la curva que se desea obtener es una cúbica. Para lo anterior es necesario considerar cuatro puntos P_0, P_1, P_2, P_3 que forman el polígono de apoyo. La curva iniciará en el punto P_0 y finalizará en el punto P_3; su trayectoria para ir del punto de inicio al punto final depende de la posición de los puntos P_1 y P_2. Como es de notar, en todos los casos el grado n de la curva es menor en uno que el número de puntos del polígono de apoyo.

La notación de los puntos del polígono de apoyo, que depende directamente del grado de la curva, será:

$$P_i = P_{i,n}$$

Ecuación 13. *Curvas paramétricas (1)*

donde i es la identificación del punto en cuestión y n es el nivel de apoyo del polígono y al inicio coincide con el número de puntos de apoyo menos uno. Así para nuestro caso de estudio $i = 0,1,2,3$ con $n = 3$, y los puntos del polígono de apoyo quedan identificados de la siguiente manera $P_0 = P_{0,3}$; $P_1 = P_{1,3}$; $P_2 = P_{2,3}$; $P_3 = P_{3,3}$; como puede observarse en la figura 2.1.

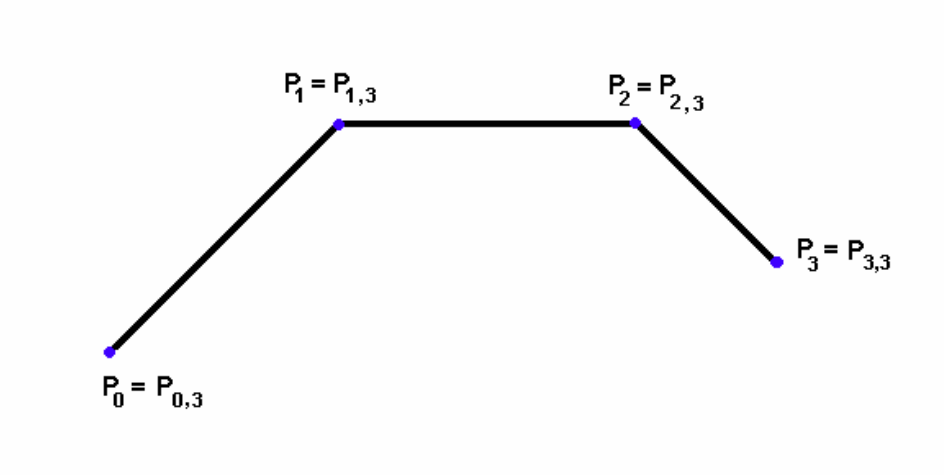


Figura 24. *Nomenclatura de los vértices del polígono de apoyo.*

Sea t un parámetro valuado a lo largo de la curva, cuyos valores posibles son: $0 \le t \le 1$, es decir, los puntos sobre la curva se definirán de acuerdo con el parámetro t, en la curva $C(t)$, y se cumple que en los puntos extremos de la curva $C(0) = P_{0,3}$ y $C(1) = P_{3,3}$.

Para construir un punto $C(t_0)$ de la curva de Bézier se deberá de seguir el siguiente procedimiento: Se toma un punto $P_{0,2}$ que se encuentra entre $P_{0,3}$ y $P_{1,3}$, de tal forma que este se localice a la $t_0 - ésima$ parte de la distancia de $\overline{P_{0,3}P_{1,3}}$ medida desde $P_{0,3}$; siguiendo la misma regla se construyen los puntos $P_{1,2}$ y $P_{2,2}$ para los segmentos $\overline{P_{1,3}P_{2,3}}$ y $\overline{P_{2,3}P_{3,3}}$. Los puntos anteriores se unen para dar cabida al siguiente nivel del polígono de apoyo, el cual queda definido en sus vértices como: $P_{i,2}$ para $i = 0,1,2$.

Con el procedimiento anterior se definen sobre el polígono de nivel 2 los puntos $P_{0,1}$ y $P_{1,1}$, localizados también a la $t_0 - ésima$ parte de los segmentos respectivos. El procedimiento se repite hasta el nivel mas bajo posible, en donde el polígono de

apoyo se ha convertido en un punto $P_{0,0}$ y este es el punto de la curva para el t_0 dado.

$$C(t_0) = P_{0,0}$$

Ecuación 14. *Curvas paramétricas (2)*

En la figura 20 se presenta la localización geométrica de un punto de la curva para cuando $t_0 = 1/3$.

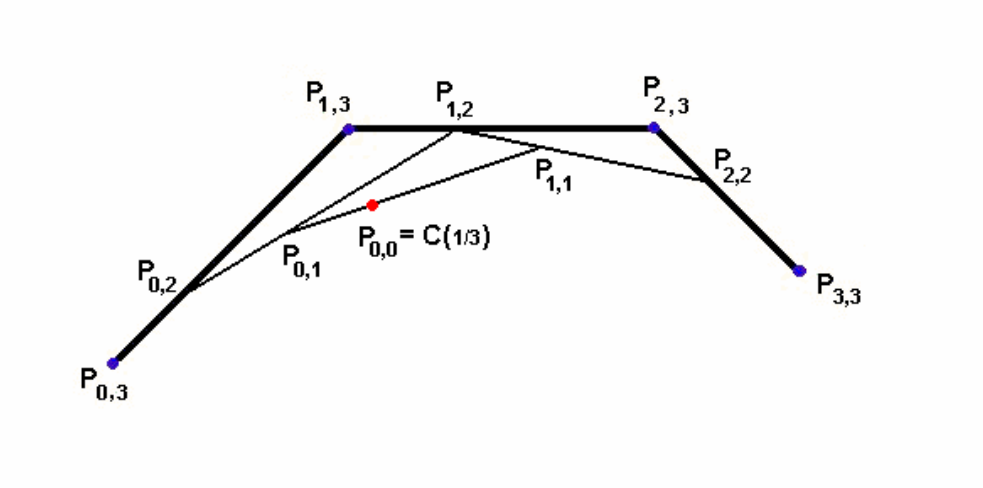


Figura 25. *Construcción del punto C(1/3)*

El procedimiento anterior permite encontrar cualquier punto de la curva $C(t)$ para el valor paramétrico $0 \le t \le 1$, sin importar el número de puntos que formen el polígono de apoyo. Cabe hacer notar que esta construcción cumple la condición de extremos:

$$C(0) = P_{0,0} \text{ y } C(1) = P_{3,3}$$

Ecuación 15. *Curvas paramétricas (3)*

3.4.2.2. Construcción analítica de la curva de Bézier

Para determinar las ecuaciones que nos permitan construir el polinomio paramétrico que describe la curva de Bézier, seguiremos los pasos de la construcción geométrica.

Consideremos para ejemplificar el método aplicado a un polígono de apoyo formado por el siguiente conjunto de puntos $P_{0,3}(5,2)$, $P_{1,3}(12,11)$, $P_{2,3}(23,9)$ y $P_{3,3}(25,2)$. Se desea construir la curva de Bézier de tercer grado en la cubierta convexa del polígono que forman los puntos anteriores, es decir tenemos que $n=3$.
La construcción de los puntos $P_{0,2}$, $P_{1,2}$, $P_{2,2}$ está dada por:

$$P_{i,2} = (1-t)P_{i,3} + tP_{i+1,3}$$

Ecuación 16. *Curvas paramétricas (4)*

para $i = 0,1,2$

Para nuestro ejemplo $t = 1/3$, tenemos:

$$i=0\ ; P_{0,2} = (1-\tfrac{1}{3})P_{0,3} + \tfrac{1}{3}P_{1,3} = (3.33335,2) + (4,3.66667) = (7.33335,5.66667)$$

$$i=1\ ;\ P_{1,2} = (15.66667,10.33333)$$

$$i=2\ ;\ P_{2,2} = (23.66667,6.66667)$$

La construcción de los puntos $P_{0,1}$, $P_{1,1}$ está dada por:

$$P_{i,1} = (1-t)P_{i,2} + tP_{i+1,2}$$

Ecuación 17. *Curvas paramétricas (5)*

para $i = 0,1$

Para nuestro ejemplo tenemos en este nivel del polígono:

$$i=0\ ;\ P_{0,1} = (10.1110,7.22221)$$

$$i=1\ ;\ P_{1,1} = (18.33335,9.11102)$$

La construcción del punto $P_{0,0}$, está dada por:

$$P_{i,0} = (1-t)P_{i,1} + tP_{i+1,1}$$

Ecuación 18. *Curvas paramétricas (6)*

para $i = 0$

Para nuestro ejemplo tenemos como punto de la curva de Bézier:

$i = 0$; $P_{0,0} = (12.85177, 7.8518)$

Para determinar la expresión general que nos permita localizar el punto $P_{0,0}$ se realizará la sustitución de (4) en (5) y el resultado en (6), entonces se tiene que:

Desarrollando (4) para $i = 0,1,2$:

$$P_{0,2} = (1-t)P_{0,3} + tP_{1,3}$$
$$P_{1,2} = (1-t)P_{1,3} + tP_{2,3}$$
$$P_{2,2} = (1-t)P_{2,3} + tP_{3,3}$$

Desarrollando (5) para $i = 01,$

$$P_{0,1} = (1-t)P_{0,2} + tP_{1,2}$$
$$P_{1,1} = (1-t)P_{1,2} + tP_{2,2}$$

Desarrollando (6) para $i = 0$

$$P_{0,0} = (1-t)P_{0,1} + tP_{1,1}$$

Sustituyendo el desarrollo de (5) en el desarrollo de (6)

$$P_{0,0} = (1-t)((1-t)P_{0,2} + tP_{1,2}) + t((1-t)P_{1,2} + tP_{2,2})$$

Sustituyendo en la ecuación anterior el desarrollo de (4)

$$P_{0,0} = (1-t)[(1-t)((1-t)P_{0,3} + tP_{1,3}) + t((1-t)P_{1,3} + tP_{2,3})]$$
$$+ t[(1-t)((1-t)P_{1,3} + tP_{2,3}) + t((1-t)P_{2,3} + tP_{3,3})]$$

Desarrollando los productos

$$P_{0,0} = (1-t)^3 P_{0,3} + (1-t)^2 tP_{1,3} + (1-t)^2 P_{1,3} + (1-t)t^2 P_{2,3}$$
$$+ (1-t)^2 tP_{1,3} + (1-t)t^2 P_{2,3} + (1-t)t^2 P_{2,3} + t^3 P_{3,3}$$

Agrupando

$$P_{0,0} = (1-t)^3 P_{0,3} + 3(1-t)^2 tP_{1,3} + 3(1-t)t^2 P_{2,3} + t^3 P_{3,3}$$

Ecuación 19. *Curvas paramétricas (7)*

Sustituyendo las expresiones (1) y (2) en el resultado anterior:

$$C(t) = (1-t)^3 P_0 + 3(1-t)^2 tP_1 + 3(1-t)t^2 P_2 + t^3 P_3$$

Ecuación 20. *Curvas paramétricas (8)*

El polinomio de la ecuación (8) tiene como coeficientes los mismos del teorema del binomio $(a+b)^n$ para $n=3$, y puede escribirse de la siguiente forma:

$$C(t) = \binom{3}{0}(1-t)^3 P_0 + \binom{3}{1}(1-t)^2 tP_1 + \binom{3}{2}(1-t)t^2 P_2 + \binom{3}{3}t^3 P_3$$

que se resume en la siguiente forma cerrada para el caso de curva de tercer grado:

$$C(t) = \sum_{i=0}^{3} \binom{3}{i} (1-t)^{3-i} t^{i} P_i$$

Ecuación 21. *Curvas paramétricas (9)*

La expresión anterior se puede representar en forma matricial, de hecho para la determinación de los puntos en la ecuación paramétrica (9) cuando $n = 3$, usaremos la siguiente formulación y la generalizaremos.

$$C(t) = \begin{bmatrix} P_0 & P_1 & P_2 & P_3 \end{bmatrix} \begin{bmatrix} (1-t)^3 \\ 3(1-t)^2 t \\ 3(1-t)t^2 \\ t^3 \end{bmatrix}$$

$$C(t) = \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix}^T \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix}$$

$$C(t) = \begin{bmatrix} P_0 & P_1 & P_2 & P_3 \end{bmatrix} B_3 \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix}$$

Ecuación 22. *Curvas paramétricas (10)*

Donde B_3 es la matriz cúbica de Bézier.

3.4.2.3. Curvas de Bézier de grado arbitrario.

La curva de Bézier está definida por $(n+1)$ puntos $P_0,...,P_n$; del desarrollo anterior en que se explicó la construcción geométrica y matemática de las curvas interpolantes de Bézier, se encuentra la forma cerrada que define la ecuación paramétrica de Bézier para el caso general, de la siguiente forma:

$$C(t) = \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-i} t^i P_i$$

Ecuación 23. *Curvas paramétricas (11)*

De la formulación matricial se puede ver de la expresión (10), que el problema para el trazado de estas curvas es la determinación de la matriz de Bézier B_n, para el valor apropiado de n.

La matriz de Bézier de grado n, se construye al desarrollar la sumatoria $\sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-i} t^i$ de la expresión (11), y escribirla en forma matricial como el producto de una matriz de coeficientes que multiplica por la izquierda al vector de los términos de t, cuyas potencias se han ordenado en forma descendente de n a $cero$.

El contar con la matriz de Bézier de grado n, permite realizar la determinación de cualquier punto de la curva al sustituir en el vector anterior el correspondiente valor del parámetro t.

Para construir la curva bastará sustituir los valores de t_0, tales que $0 \le t_0 \le 1$. Para fines prácticos se recomienda dividir el intervalo de validez del parámetro t, en un apropiado número de segmentos s, determinar un paso de evaluación como $p = 1/s$, y evaluar la curva para $t_0 = 0 + ip$ para $i = 0,1,2,\cdots s$; y cada punto así obtenido se une mediante un segmento de recta al punto anterior.

El efecto visual de curvatura suave, será mucho mejor en cada caso si el valor de s es grande.

3.4.2.4. Condiciones de frontera.

Analizaremos ahora las condiciones de frontera de las curvas de Bézier, iniciaremos obteniendo la derivada de (11) con respecto a t.

$$C'(t) = \sum_{i=0}^{n} \binom{n}{i} (i(1-t)^{n-i} t^{i-1} - (n-i)(1-t)^{n-i-1} t^{i}) P_i$$

Ecuación 24. *Curvas paramétricas (12)*

Evaluando las ecuaciones (11) y (12) en los extremos ($t = 0$, $t = 1$), se obtienen las siguientes condiciones de frontera:

Desarrollando la ecuación (11), se tiene:

$$C(t) = \binom{n}{0}(1-t)^{n} P_0 + \binom{n}{1}(1-t)^{n-1} t P_1 + \cdots + \binom{n}{n-1}(1-t)^{1} t^{n-1} P_{n-1} + \binom{n}{n} t^{n} P_n$$

De la inspección de los términos podemos darnos cuenta que al sustituir $t = 0$ se obtiene el punto inicial P_0 del primer término y todos los demás términos se hacen cero. En el caso de $t = 1$ se obtiene solo el punto final P_n, ya que solo el último término prevalece y los demás se hacen cero.

$$C(0) = P_0$$

$$C(1) = P_n$$

De igual forma desarrollaremos la ecuación (12), para analizar las condiciones de frontera.

$$C'(t) = \binom{n}{0}[-n]P_0 + \binom{n}{1}[(1-t)^{n-1} - (n-1)(1-t)^{n-2}t]P_1 +$$

$$\binom{n}{2}[2(1-t)^{n-2}t - (n-2)(1-t)^{n-3}t^2]P_2 + \cdots + \binom{n}{n-1}[(n-1)(1-t)t^{n-2} - t^{n-1}]P_{n-1} + \binom{n}{n}[nt^{n-1}]P_n$$

Al sustituir las condiciones de frontera, encontramos que el valor de la pendiente de la curva de Bézier, en los extremos es:

$$C'(0) = n(P_1 - P_0)$$

$$C'(1) = n(P_n - P_{n-1})$$

Los resultados anteriores nos muestran que la curva pasa por los dos puntos extremos del polígono de apoyo P_0 y P_n; además de que en estos puntos la curva es tangente al primer y al último segmento del polígono de apoyo.

En la formulación matricial de la curva $C(t)$ se tiene, como se mostró en la expresión (10), la transpuesta del vector de coordenadas del polígono de apoyo P^T, la matriz $n-ésima$ de Bézier B_n y el vector de los términos de t, cuyas potencias van desde n a $cero$, que se designará como T. Entonces la curva paramétrica se escribe de la siguiente manera:

$$C(t) = P^T B_n T$$

Curvas paramétricas (13)

Así la curva de Bézier $C(t)$ está en la cubierta convexa de sus puntos de control. Esta es una propiedad importante, y la principal razón por la que estas curvas son muy empleadas; pues esta es una condición que otros métodos no pueden asegurar, generando en ocasiones puntos que no pertenecen y que incluso llegan a estar muy lejos de la cubierta convexa.

3.4.2.5. Propiedades.

A continuación se presentan las propiedades de las curvas de Bézier.

La curva en general no pasa por ninguno de los puntos de control, excepto por el primero y el último.
La curva siempre está en la cubierta convexa de los puntos de control.
Si solo existe un punto de control P_0, entonces $n = 0$ y $C(t) = P_0$ para toda t.
Si solo existen dos puntos de control P_0 y P_1, entonces $n = 1$ y la curva de Bézier es la línea recta que une estos dos puntos.
El término $\sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-i} t^i$, es llamado la función de curvatura.
La función de curvatura es siempre un polinomio de grado menor en uno que el número de puntos de control.
Se pueden generar curvas cerradas haciendo el último punto de control igual al primero de ellos.
Cuando el número de puntos de control se incrementa se presentan necesariamente polinomios de alto grado.
Excepto por el caso redundante de dos puntos de control, en general no es posible generar curvas de Bézier paralelas.

3.4.2.6. Unión de dos curvas.

Un problema importante en el uso de curvas de Bézier es el manejo de polígonos de un alto número de puntos, ya que generan necesariamente polinomios de alto

grado. El manejo de estos polinomios es complicado y se recomienda en estos casos dividir el problema y unir dos o mas curvas.

Para que la unión de estas curvas sean de forma suave se recomienda utilizar puntos intermedios de paso obligado, los cuales pueden ser los puntos iniciales y finales de las curvas a unir.

Para garantizar la transición suave de una curva a otra es necesario que la pendiente de ambas curvas en el punto de unión sea la misma, es decir la pendiente del punto final para la curva de atrás, es igual a la pendiente del punto de inicio para la curva de adelante.

Lo anterior puede garantizarse si en la construcción tenemos cuidado de que la pendiente del último tramo del polígono de apoyo de la curva de atrás es igual a la pendiente del primer tramo del polígono de apoyo de la curva de adelante; y el punto final de la curva de atrás sea el mismo que el primer punto de la curva de adelante (mismas coordenadas). Como se muestra en la figura 26.

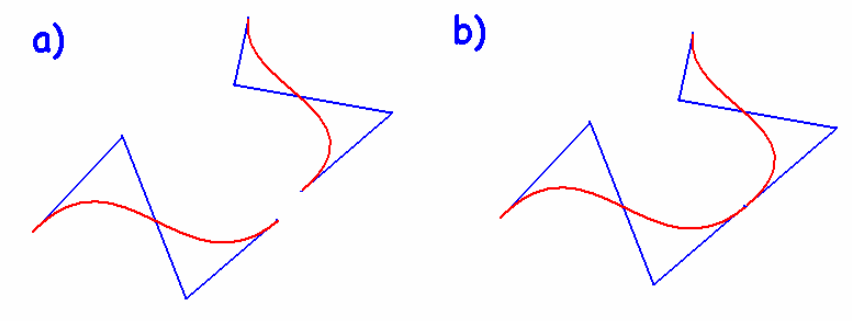


Figura 26. *a) Dos curvas con sus polígonos de apoyo. b) Unión de las curvas*

3.5. CONCEPTOS BÁSICOS ACERCA DE LA PLUVIOMETRIA

3.5.1. AGUA ATMOSFÉRICA

De los diversos procesos meteorológicos que ocurren continuamente en la atmósfera, los más importantes para la hidrología son los de precipitación y evaporación, en los cuales la atmósfera interactúa con el agua superficial. La mayor parte del agua que se precipita sobre la superficie terrestre proviene de la humedad que se evapora en los océanos y que es transportada por la circulación atmosférica a lo largo de grandes distancias. Las dos fuerzas básicas para la circulación atmosférica resultan de la rotación de la Tierra y de la transferencia de energía calórica entre el ecuador y los polos[9].

3.5.2. PRECIPITACIÓN

La precipitación incluye la lluvia, la nieve y otros procesos mediante los cuales el agua cae a la superficie terrestre, tales como granizo y nevisca. La formación de precipitación requiere la elevación de una masa de agua en la atmósfera de tal manera que se enfríe y parte de su humedad se condense. Los tres mecanismos principales para la elevación de masas de aire son la *Elevación Frontal*, donde el aire caliente es elevado sobre aire frío por un pasaje frontal; la *elevación orográfica,* mediante la cual una masa de aire se eleva para pasar por encima de una cadena montañosa; y la *elevación convectiva,* donde el aire se arrastra hacia arriba por una acción convectiva, como ocurre en el centro de una celda de una tormenta eléctrica. Las celdas convectivas se originan por el calor superficial, el cual una inestabilidad vertical de aire húmedo, y se sostienen pro el calor latente de vaporización liberado a medida que el vapor del agua sube y se condesa.

A medida que el aire sube y se enfría, el agua se condensa de un estado de vapor a un estado líquido. Si la temperatura se encuentra por debajo del punto de congelamiento se forman los cristales de hielo. La condensación requiere de una semilla llamada el *núcleo de condensación* alrededor del cual las moléculas del agua se pueden unir o nuclear. Algunas partículas de polvo que flotan en el aire pueden actuar como núcleo de condensación; las partículas que contienen iones son efectivas como núcleos debido a que los iones atraen pro electrostática las

[9] Maidment, David R. **Hidrología aplicada**. Mc Graw Hill

moléculas de agua enlazadas polarmente. Los iones en la atmósfera incluyen partículas de sal que se forman a partir de la evaporación de espuma marina, y compuestos de sulfuro y nitrógeno resultantes de procesos de combustión. Los diámetros de estas partículas varían desde 10^{-3} hasta 10 um y se conoce como *aerosoles.* Como comparación el tamaño de un átomo es aproximadamente de 10^{-4} um, lo cual significa que los aerosoles más pequeños pueden componerse solo de cuentos de átomos.

Las pequeñas gotas de agua crecen mediante la condensación e impactan con las más cercanas a medida que se mueven por la turbulencia del aire hasta que son lo suficientemente grandes para que la fuerza de gravedad sobrepase la fuerza de fricción y empiezan a caer, incrementando su tamaño cuando golpean otras gotas en su descenso. Sin embargo, a medida que la gota cae, el agua se evapora de su superficie y su tamaño disminuye, de tal manera que pueden reducirse nuevamente al tamaño de un aerosol y desplazarse hacia arriba en la nube debido a la turbulencia.

Una corriente ascendente de solamente 0.5 cm/s es suficiente para arrastrar una pequeña gota de 10um. Algunos cristales de hielo del mismo peso, debido a su mayor forma y tamaño, pueden ser arrastrados pro velocidades aún más pequeñas. El ciclo de condensación, caída, evaporación y elevación se repite en promedio unas diez veces antes de que la gota alcance un tamaño crítico de alrededor de 01. Mm, que es suficientemente grande pata que caiga a través de la base de la nube.

Las gotas permanecen esféricas hasta un diámetro de la rededor de 1mm,pero empiezan a aplanarse en el fondo cuando aumentan su tamaño, y dejan de ser estables en su caída al atravesar el aire dividiéndose en pequeñas gotas de llueva. Las gotas de lluvia normales que caen a través de la base de una nube tienen de 0.1 a 3 mm de diámetro.

Algunas observaciones indican que en las nubes pueden existir gotas de agua a temperatura por debajo del punto de congelación, hasta unos –35°C. A esta temperatura, las gotas superenfriadas se congelan aun sin la presencia de núcleos

de congelamiento. La presión de vapor de saturación del valor de agua es menor en hielo que es agua liquida; luego si las partículas de hielo se mezclan con gotas de agua, estas partículas crecerán por efecto de la evaporación de las gotas y las condensación de los cristales de hielo. Los cristales de hielo normalmente forman racimos mediante colisión y fusión y caen como copos de nieve. Sin embargo, algunos cristales de hielo pueden crecer tanto, que caen directamente a la tierra como granizo o nevisca.

Las pequeñas gotas de agua en las nubes se forman por nucleación de vapor sobre los aerosoles, para luego pasar por varios ciclos de condensación – evaporación a medida que circulan en la nube, hasta que alcanzan un tamaño suficientemente grande para caer a través de la base de la nube.

Un registro de pluviografo está compuesto por un conjunto de profundidades de lluvia que se registra para incrementos de tiempo sucesivos. Un hietograma de lluvia es una gráfica de profundidad de lluvia o intensidad en función del tiempo. Sumando los incrementos de lluvia a través del tiempo, un hietograma de lluvia acumulada, o curva de masa de lluvia, se produce tal como se muestra en la siguiente tabla .

Tiempo	**Lluvia**	**Lluvia**	**Totales corrientes**		
(min)	**(pulg)**	**acumulada**	**30 min**	**1h**	**2h**
0		0.00			
5	0.02	0.02			
10	0.34	0.36			
15	0.10	0.46			
20	0.04	0.50			
25	0.19	0.69			
30	0.48	1.17	1.17		
35	0.50	1.67	1.65		

40	0.50	2.17	1.81		
45	0.51	2.68	2.22		
50	0.16	2.84	2.34		
55	0.31	3.15	2.46		
60	0.66	3.81	2.64	3.81	
65	0.36	4.17	2.50	4.15	
70	0.39	4.56	2.39	4.20	
75	0.6	4.92	2.24	4.46	
80	0.54	5.46	2.62	4.96	
85	0.76	6.22	**3.07**	5.53	
90	0.51	6.73	2.92	**5.56**	
95	0.44	7.17	3.00	5.50	
100	0.25	7.42	2.86	5.25	
105	0.25	7.67	2.75	4.99	
110	0.22	7.89	2.43	5.05	
115	0.15	8.04	1.82	4.89	
120	0.09	8.13	1.40	4.32	8.13
125	0.09	8.22	1.05	4.05	**8.20**
130	0.12	8.34	0.92	3.78	7.98
135	0.03	8.37	0.70	3.45	7.91
140	0.01	8.38	0.49	2.92	7.88
145	0.02	8.40	0.36	2.18	7.71
150	0.01	8.41	0.28	1.68	7.24
Profundidad máx.	0.76		3.07	5.56	8.20
Intensidad máx (pulg/h)	9.12.		6.14	5.55	4.10

Figura 27. *Cálculo de profundidad e intensidad de lluvia en un punto*

La máxima profundidad de lluvia o intensidad (profundidad/tiempo) que se registra en un intervalo de tiempo de referencia, para una tormenta, se establece calculando las profundidades de lluvia totales corrientes pata ese intervalo de tiempo empezando en algunos puntos de la tormenta, para luego seleccionar el valor

máximo de esta serie. Por ejemplo, para un intervalo de media hora, la tabla muestra los totales corrientes con las 1.17 pulgadas que se registraron en los primeros 30 minutos, las 1.65 pulgadas desde los 5 hasta los 35 minutos, las 1.81 pulgadas desde los 10 hasta los 40 minutos y así sucesivamente. La máxima profundidad que se registró en 30 minutos fue 3.07 pulgadas desde los 55 hasta los 85 minutos que corresponde a una intensidad promedio de 3.07 pulg/0.5 h = 6.14 pulg/h en ese intervalo.

4. DISEÑO Y CONTRUCCIÓN

El proceso unificado es una metodología de desarrollo que proporciona normas para el desarrollo eficiente de software de calidad dentro de los plazos y presupuestos

planeados. Entendiendo proceso de desarrollo de software como el conjunto de actividades necesarias para transformar los requisitos de un usuario en un sistema software.

El Proceso unificado esta dirigido por casos de uso, centrado en la arquitectura, y es iterativo e incremental.

DIRIGIDO POR CASOS DE USO: El desarrollo del software se centra en la importancia del desarrollo para el usuario y no en términos de funciones que debe cumplir el sistema. Los casos de uso dirigen el proceso durante todos los flujos de trabajo de las distintas fases.

Un caso de uso es una descripción de un conjunto de secuencias de acciones que un sistema lleva a cabo, un fragmento de su funcionalidad y que proporciona a un resultado interés para un actor determinado, donde un actor puede ser un usuario, un sistema o un rol.

CENTRADO EN LA ARQUITECTURA: Al describir la arquitectura se obtiene una mayor comprensión del sistema, se organiza el desarrollo y se fomenta la reutilización. Esta arquitectura abarca la organización del sistema software, los elementos estructurales que compondrán el sistema y sus interfaces, así como su comportamiento y colaboraciones entre elementos.

***ES ITERATIVO E INCREMENTAL*:** Un proceso iterativo permite una comprensión creciente de los requerimientos, a la vez que se va haciendo crecer el sistema abordando las tareas más riesgosas primero. El trabajo de desarrollo se divide de manera planeada en partes más pequeñas llamadas iteraciones lo cual genera progresivamente un incremento en el proyecto total.

En cada iteración, se identifican y especifican los casos de uso relevantes, se crea un diseño utilizando la arquitectura seleccionada como guía, se implementa el diseño mediante componentes, y se verifican que los componentes satisfacen los casos de uso. Si una iteración cumple sus objetivos el desarrollo continúa con la

siguiente iteración, en caso contrario, se revisan las decisiones previas y se prueba un nuevo enfoque.

Un desarrollo iterativo, guiado por los casos de uso y centrado en la arquitectura, construye un software mediante pequeños incrementos, y añade cada incremento a la acumulación previa de incrementos de tal forma que siempre se tenga una construcción ejecutable. La arquitectura proporciona la estructura sobre la cual guiar las iteraciones mientras que los casos de uso definen los objetivos y dirigen el trabajo de cada iteración.

De esta manera el proceso reduce el riesgo de grandes retrasos en la entrega de un producto, se fijan metas más inmediatas por lo cual se puede controlar mejor el avance del proyecto.

El Proceso Unificado divide el proces*o* de desarrollo en ciclos, donde se obtiene una nueva versión del producto al final de cada ciclo. Cada ciclo se divide en cuatro Fases: Inicio, Elaboración, Construcción, y Transición. Cada una de estas fases concluye con un hito[10] bien definido donde deben tomarse decisiones respecto al proyecto como la reestructuración del cronograma de trabajo. Cada una de estas fases se divide a su vez en iteraciones.

Cada iteración sigue la estructura de un pequeño ciclo de vida en cascada, pasando a través de los cinco flujos de trabajo fundamentales: requisitos, análisis, diseño, implementación y prueba.[11] La iteración también incluye la planificación que precede a los flujos de trabajo y la evaluación que va detrás de ellos.

---

[10] Un hito es un evento que se sucede en el tiempo y que controla la iniciación o finalización de una grupo de tareas en un proyecto. El hito no consume tiempo ni recursos pero el hecho de que suceda permite que otras tareas puedan llevarse a cabo.

[11] Tomado del libro “Proceso Unificado de Desarrollo de Software” de Jacobson, Booch y Rumbaugh. Estas fases difieren del proceso unificado de Rational explicado en su sitio www.rational.com debido

4.1. CASOS DE USO Y REQUISITOS

En la fase de inicio del desarrollo de todo proyecto que utilice la metodología del proceso unificado se hace un análisis que justifique la ejecución del proyecto y se definen los alcances que este tendrá, estableciendo metas y límites, la arquitectura y posibles dificultades a sortear durante la ejecución del proyecto.

4.1.1. Lista de Características

La lista de características surge a menudo de la experiencia que los clientes/usuarios tienen con sistemas similares, en este caso las entidades interesadas no habían interactuado nunca con una solución informática que brindara un elemento de referencia que pudiera guiar hacia la obtención de esta lista, sin embargo fue posible hacer una proyección acerca de las necesidades básicas que se deberían cubrir y en base a estas se fue desarrollando el listado. Estas características se traducen en requisitos, a cada uno de ellos se les debe asignar un nombre, descripción, estado (propuesto o aprobado), recursos, prioridad y nivel de riesgo. Para lograr definir los requisitos de una forma mas organizada y facilitar su comprensión estos se agrupan por módulos que permiten:

A. Lectura y organización de datos.
B. Pre-procesamiento de la imagen digital.
C. Extracción de la señal.
D. Edición de la señal extraída.
E. Almacenamiento de la información.
F. Análisis de la señal.

---

que en su clasificación se hacen una división entre modelo de negocio y requisitos, aquí tomado como un solo flujo llamado requisitos. Además en su método se unifica la fase de análisis y diseño. Para el resto del proyecto la metodología que se seguirá es la presentada en el libro citado en esta nota.

En la siguiente tabla se detalla la lista de requisitos iniciales del sistema y se observan las características evaluadas en la captura de requisitos.

La prioridad que puede ser *critica, importante o secundaria*, los requisitos de prioridad *critica* son aquellos mas importantes para los usuarios del sistema, *importante* son los que sirven de apoyo a los requisitos críticos y tienen un pequeño impacto en la definición de la arquitectura, los requisitos de prioridad *secundaria* no son claves en la definición de la arquitectura o para la definición de los riesgos críticos.

Los niveles de riesgos que pueden ser *críticos, significativos u ordinarios*, *crítico* cuando es fundamental en la definición de la arquitectura, *significativo* son aquellos que tienen importancia relativa al cumplimiento de las necesidades de los clientes y *ordinario* son los relativos al rendimiento del sistema.

	Nombre	**Descripción**	**Prioridad**	**Nivel de riesgo**
A1	Leer imágenes desde archivos en formato de mapas de bits (*.BMP)	Los algoritmos implementados deben ser inicialmente alimentados con información, la estructura más básica en que esta puede ser encontrada en en la forma de mapas de bits	Critica	Critico
A2	Leer imágenes desde archivos en diferentes formatos de imagen (*.jpg, *.gif, etc.)	Como alternativa a la carga de información a partir de mapas de bits se propone la carga desde archivos de imágenes con otros formatos	Secundaria	Ordinario

	Nombre	Descripción	Prioridad	Nivel de riesgo
A3	Leer información desde archivos propios de la interfaz a implementar	Una vez procesados los pluviogramas estos se guardan en un archivo propio de la aplicación, el cual la interfaz debe estar en capacidad de cargar nuevamente	Critica	Crítico
A4	Almacenar los canales de la imagen es estructuras que permitan tratarlos de manera independiente o en conjunto	El usuario podrá elegir el canal más adecuado para el procesamiento y podrá tratarlos como entidades individuales o en conjunto como una imagen compuesta	Crítica	Crítico
B1	Corregir la alineación de la imagen cargada.	Debido a que la imagen puede estar inclinada se debe presentar una utilidad que permita su alineación con respecto a la horizontal.	Critica	Significativo
B2	Elegir un área de la imagen para trabajar sobre ella.	El usuario podrá seleccionar un área de la imagen sobre la cual trabajar desechando toda la información que no es relativa a la señal y que influiría en el procesamiento y los resultados finales	Crítica	Crítico
B3	Aplicar filtros espaciales a los canales de la imagen.	El usuario podrá aplicar a los canales de la imagen filtros espaciales definidos por él mismo que permitan realzar o atenuar características particulares de la imagen.	Importante	Crítico
B4	Umbralizar el canal de trabajo activo	El usuario podrá umbralizar el canal activo con el fin de empezar la diferenciación entre la señal y el resto de la imagen	Crítico	Significativo
B5	Proveer una sugerencia acerca del valor de umbralización de manera automática.	El usuario dispondrá de un valor sugerido de umbralización calculado por un algoritmo especializado.	Secundaria	Relativo

	Nombre	Descripción	Prioridad	Nivel de riesgo
B6	Aplicar procedimientos que permitan restaurar segmentos dañados de la imagen luego de la umbralización.	El usuario podrá corregir algunos defectos dejados por el procedimiento de umbralización	Importante	Significativo
C1	Distinguir entre segmentos de líneas que compongan la señal y extraerlos a estructuras convenientes para su manipulación.	Debe proveerse de un procedimiento que de manera automática clasifique y extraiga de manera separada los segmentos de líneas que conforman la señal.	Crítica	Crítico
C2	Filtrar de manera automática los segmentos que puedan no pertenecer a la señal	El usuario podrá establecer un valor de umbral que servirá como criterio para escoger cuales segmentos mostrar como pertenecientes a la señal.	Secundaria	Relativo
C3	Descartar los segmentos que el usuario considere no pertenecen a la señal	El usuario podrá elegir a voluntad cuales segmentos pertenecen o no a la señal.	Importante	Relativo
D1	Editar la señal extraída modificando sus valores en el tiempo.	Se debe estar en capacidad de modificar, agregar o quitar valores en el tiempo de la señal	Crítica	Crítica
D2	La edición de la señal extraída debe relazarse en tiempo de ejecución y de manera interactiva	La edición de la señal se realizara directamente sobre ella modificando su forma mediante la edición con el Mouse de nodos que la definan.	Critica	Crítica
E1	Almacenar la señal extraida	La señal extraída y ajustada será almacenada en un formato propio de los algoritmos implementados	Crítica	Crítica

	Nombre	Descripción	Prioridad	Nivel de riesgo
E2	Los archivos propios de almacenamiento deben ser de poco tamaño.	La señal guardada en el formato propio de los algoritmos debe ocupar máximo un 15% del espacio en disco utilizado por el archivo BMP de origen.	Importante	Significativo
E3	Proveer formato de exportación para la señal	El usuario podrá exportar la señal extraída a un formato de archivo de texto plano que pueda ser utilizado en otras aplicaciones.	Importante	Relativo
D1	Calcular gráficas a partir del análisis de la señal extraída	El usuario podrá calcular la curva masa, histograma de precipitación y la curva de intensidad a partir de la señal extraída	Crítica	Relativo

Figura 28. *Listado de requisitos.*

4.1.2. RIESGOS CRÍTICOS

Un riesgo crítico es aquel que supone una barrera tal que podría afectar por completo la ejecución del proyecto a tal punto que el no ser sorteado rápidamente conllevaría a la detención total e incluso a la cancelación de este.

En la siguiente lista se muestran los principales riesgos críticos detectados en la fase de inicio en la ejecución

Descripción	Impacto	Contingencia
Perdida de información debido a daños en el equipo de desarrollo	Perdida total o parcial de la programación o documentación	Adquisición de una unidad quemadora para almacenamiento diario y permanente del trabajo realizado en la jornada

Descripción	Impacto	Contingencia
Dificultades en el tratamiento de temas avanzados en la programación sobre la herramienta seleccionada para el desarrollo	Retrazo o equivocaciones en la implementación debido a la alta pendiente de curva de aprendizaje que suponen los temas avanzados en Visual C++	Adquisición de documentación acerca de temas avanzado en Visual C++ y planeación para su estudio y práctica diaria antes de requerir su utilización
Conocimientos poco avanzado en temas de tratamiento de imágenes digitales y métodos numéricos	Retrazo en la ejecución del proyecto debido al tiempo requerido para avanzar en la temática	Se cuenta con la ayuda y asesoramiento de un experto en ambos temas que ayudará a que se reduzca el tiempo de aprendizaje

Figura 29. *Riesgos Críticos del Sistema*

4.1.3. MODELO DEL NEGOCIO

El modelo del negocio describe los procesos que se ejecutan en el sistema permitiendo identificar las actividades y el actor que las lleva a cabo. Este modelo tiene como objetivo identificar los casos de uso y las entidades del negocio, necesarias para comprender el funcionamiento del sistema.

A partir del modelo del negocio es posible definir inicialmente y de manera directa los casos de uso iniciales más importantes dentro de cada proceso del negocio.

Definir los actores es un paso esencial para obtener de manera correcta los requisitos. Para obtener los actores se debe partir del modelo de negocio y seguir los siguientes pasos:

- Identificar al menos a un usuario que pueda representar al actor candidato obteniendo así a los actores relevantes.

- Encontrar una coincidencia mínima entre los roles que desempeñan las instancias de los diferentes casos de uso en relación con el sistema, con el fin de evitar que dos o más actores sigan en esencia los mismos roles. De ser así, se debe crear un actor generalizado que tenga asignados los roles comunes a los actores que se superponen.
- Describir de una manera breve las funciones de cada actor y su papel en la utilización del sistema

Debido al contexto investigativo de este proyecto se a definido un solo actor, el actor “USUARIO”, quien gestiona las herramientas implementadas en la interfaz de usuario de manera libre con el fin de someter un pluviograma a los procedimientos necesarios para la extracción de la señal pluviométrica.

4.1.3.1. Casos de uso

Los casos de uso son fundamentales para definir una visión global del sistema e identificar los subsistemas que conforman el mismo, estos se agrupan dependiendo del paquete de análisis al que pertenezcan donde cada paquete presenta el modelo de casos de uso y una descripción general de cada caso de uso. Sólo se detallan aquellos casos de uso que se juzgue conveniente para la comprensión del funcionamiento general del sistema. En la siguiente gráfica se presenta el diagrama general de los casos de uso que hacen parte del contexto del sistema.

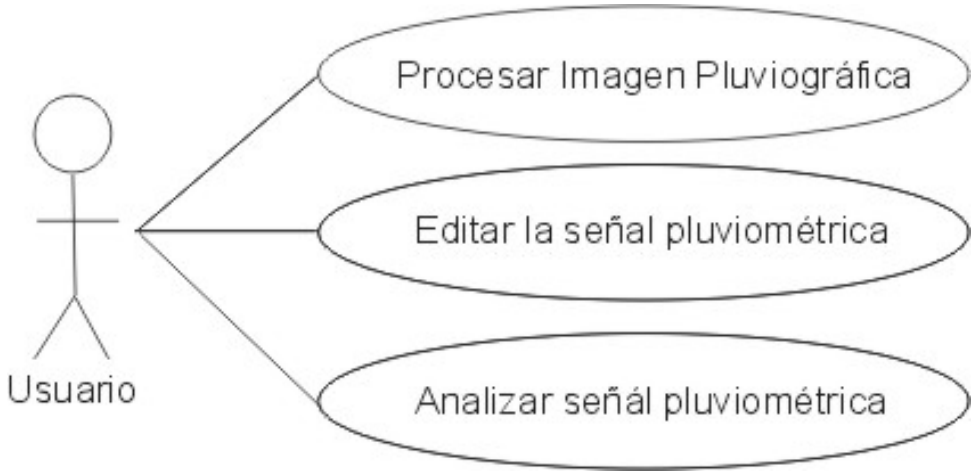


Figura 30. *Modelo general de casos de uso*

Procesar Imagen Pluviográfica	
Objetivo	Someter la imagen digital pluviográfica al los procesos necesarios encaminados a la extracción de la señal pluviométrica que contiene
Descripción	El Usuario carga una imagen en formato BMP y la somete a los procedimientos de tratamiento digital y métodos numéricos necesarios hasta concluir con una señal almacenada en un vector totalmente independiente de la imagen.
Riesgos	Son muchos los procesos intermedios ejecutados en este proceso y los cuales deben darse de manera secuencial, si alguno de estos llega a fallar o a arrojar información errónea la señal finalmente extraída puede ser poco representativa de la original.
Tiempo de ejecución	Depende del tamaño de la imagen y las especificaciones del equipo de cómputo

Figura 31. *Proceso del negocio: Procesar imagen pluviográfica*

Edición de la señal pluviométrica	
Objetivo	Someter una señal pluviométrica extraída a correcciones y ediciones manuales.

Edición de la señal pluviométrica	
Descripción	Una vez extraída una señal pluviométrica a vectores, esta puede ser modificada y editada con el fin de corregir errores provenientes en ella misma debido a la captura en el papel por parte del pluviómetro o errores ocasionados por los algoritmos durante el pre-procesamiento y la extracción.
Riesgos	Debido a que este es un procedimiento interactivo, el usuario esta en facultad de cambiar a voluntad la forma y los valores en el tiempo de la señal, luego podría en lugar de corregir errores dañar totalmente la señal ocasionando tener que repetir totalmente el proceso.
Tiempo de ejecución	Depende del usuario.

Figura 32. *Proceso del negocio: Edición de la señal pluviométrica*

Analizar señal pluviométrica	
Objetivo	Someter una señal pluviométrica extraída a análisis que arrojen información referente a la precipitación que representa
Descripción	Una vez extraída una señal pluviométrica esta puede ser procesada hasta obtener las gráficas de curva masa, histograma de precipitación e histograma de intensidad. Esto se puede dar inmediatamente después de la extracción o después, al cargar en la interfaz el archivo de proyecto que representa la señal extraida.
Riesgos	El resultado del análisis es totalmente dependiente del éxito del proceso de extracción, si la señal extraída no es representativa del pluviograma seguramente la información obtenida en este proceso no será coherente con el pluviograma original
Tiempo de ejecución	Depende de las especificaciones del equipo de cómputo

Figura 33. *Proceso del negocio: Análisis de la señal pluviométrica*

4.1.3.1.1. Caso de uso: Procesar imagen pluviográfica

El sistema permite al usuario cargar imágenes en formato BMP que define n pluviogramas objeto de estudio. Al cargar una imagen BMP el usuario está en disposición de aplicar a esta un conjunto de procedimientos que considere necesarios y que el sistema ofrece en miras a individualizar la porción de imagen que representa a la señal pluviométrica para proceder a su extracción hacia vectores

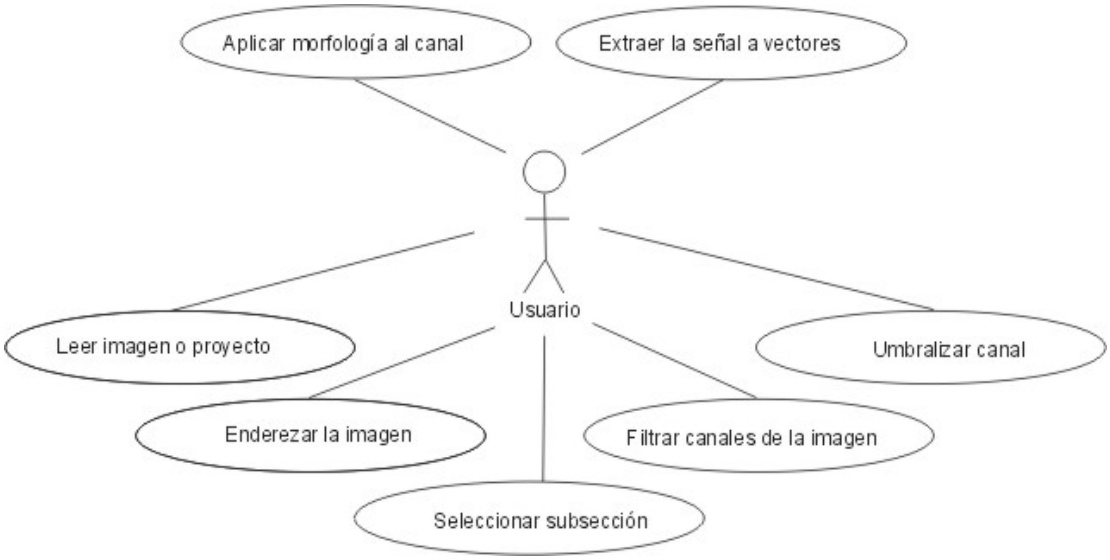


Figura 34. *Detalles del caso de uso procesar imagen pluviográfica*

Casos de uso identificados	Descripción
Leer imagen o proyecto	El usuario lee una imagen digital o un proyecto que representa un pluviograma.
Enderezar imagen	El usuario corrige inclinaciones de la imagen con respecto a la horizontal

Casos de uso identificados	Descripción
Seleccionar subsección	El usuario selecciona el area de la imagen que contiene la señal pluviométrica y la malla de registro.
Filtrar canales	El usuario somete los canales de la imagen cargada a un procedimiento de filtrado espacial con el fin de resaltar o atenuar algunas de sus características.
Umbralizar canal	El usuario umbraliza un canal para obtener una imagen binaria sobre la cual trabajar
Aplicar morfología	El usuario aplica algoritmos morfológicos matemáticos al canal binario con el fin de reparar o eliminar segmentos
Extraer señal	El usuario extrae la señal dentro de vectores que definen componentes conexas identificadas en el canal binario fianal

4.1.3.1.2. Caso de uso: Editar señal pluviométrica

El sistema permite al usuario modificar a voluntad y de manera interactiva la señal pluviométrica extraída mediante el ajuste de esta a curvas Spline o Bezier y la edición de los parámetros que definen estos elementos númericos.

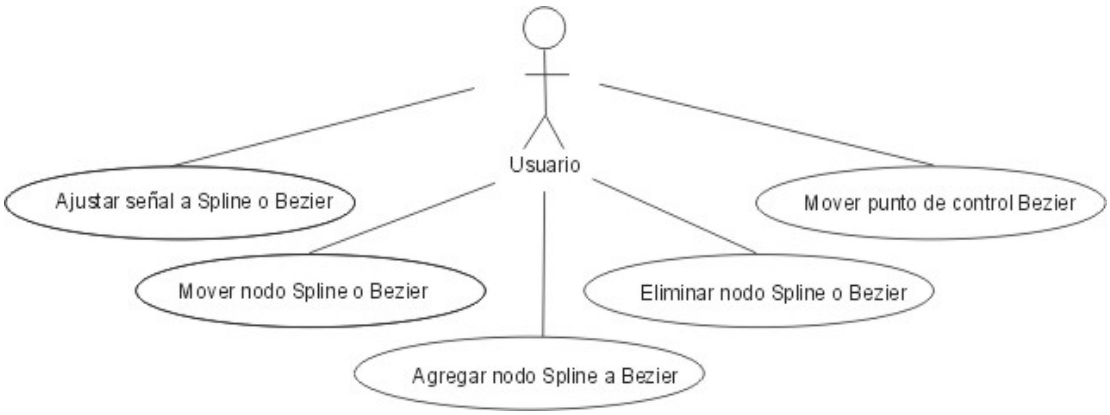


Figura 35. *Detalles del caso de uso editar señal pluviométrica*

Casos de uso identificados	Descripción
Ajustar señal a Spline o Bezier	El usuario podrá ejecutar una algoritmo de ajuste que intente convertir la señal extraída en vectores a una curva Spline o Bezier
Mover nodo Spline o Bezier	El usuario edita la señal pluviométrica moviendo los nodos que definen la curva Spline o Bezier que la representa
Agregar nodo Spline o Bezier	El usuario edita la señal pluviométrica agregando nodos que definen la curva Spline o Bezier que la representa.
Eliminar nodo Spline o Bezier	El usuario edita la señal pluviométrica borrando nodos que definen la curva Spline o Bezier que la representa.
Mover punto de control Bezier	El usuario edita la señal pluviométrica moviendo los puntos de control que definen la forma de un segmento de curva Bezier.

4.1.3.1.3. Detalles de los casos de uso

Al detallar los casos de uso se describe el flujo de sucesos en detalle, especificándose como comienza, como termina y como interactúa con los actores.

Caso de uso	Leer imagen
Precondición	La imagen a leer debe estar en formato BMP con resolución de color de 24 o paletizada de 8 bits no comprimida mediante ningún método.
Descripción o flujo de sucesos	El usuario selecciona un archivo en disco, los algoritmos establecen si la imagen puede ser procesada y se basan en su estructura para sacar información referente a la imagen y organizarla en los objetos propios.
Caminos alternativos	Si la imagen no puede ser procesada los algoritmos arrojan una excepción y el proceso de carga se interrumpe

Poscondiciones	Ninguna
Requisitos especiales	Debe estar establecida la estructura donde se almacenara la imagen

Figura 36. *Descripción detallada del caso de uso Leer Imagen*

Caso de uso	Enderezar imagen
Precondición	Debe haber una imagen cargada en las estructuras que la representan
Descripción o flujo de sucesos	El usuario escoge dos puntos sobre la imagen que definan una línea recta inclinada, luego los algoritmos calculan la inclinación de esta línea y enderezan la imagen tantos grados como este inclinada la línea.
Caminos alternativos	Si los algoritmos detectan que la línea demarcada no se encuentra inclinada ningún procedimiento de enderezamiento es ejecutado
Poscondiciones	Ninguna
Requisitos especiales	El sistema debe disponen de suficiente cantidad de memoria física y tiempo de procesador disponible para llevar a cabo este proceso.

Figura 37. *Descripción detallada del caso de uso Enderezar Imagen*

Caso de uso	Seleccionar subsección
Precondición	Las dimensiones de la subsección deben estar dentro de las dimensiones de la imagen original y estar en condiciones de definir un área.
Descripción o flujo de sucesos	El usuario define un área dentro de la imagen con la que desea trabajar, luego la imagen original es eliminada de la estructura que la contiene y reemplazada por la subsección seleccionada.

Caminos alternativos	Si las dimensiones de la subsección se encuentran fuera de las dimensiones de la imagen original la imagen original es conservada dentro de las estructuras.
Poscondiciones	Ninguna
Requisitos especiales	El sistema debe disponen de suficiente cantidad de memoria física y tiempo de procesador disponible para llevar a cabo este proceso.

Figura 38. *Descripción detallada del caso de uso Seleccionar Subsección*

Caso de uso	Filtrar imagen
Precondición	Debe haber una imagen cargada en las estructuras que la representan.
Descripción o flujo de sucesos	El usuario define una "matriz de convolución" a ser aplicada a uno de los canales de la imagen, luego esta se pasa como filtro al canal seleccionado y el resultado se despliega en pantalla.
Caminos alternativos	Ninguno
Poscondiciones	Dependiendo del tipo de filtro aplicado, el resultado puede estar fuera del rango de visualización del monitor, luego es posible que el rango dinámico de la imagen requiera ser ajustado.
Requisitos especiales	El sistema debe disponen de suficiente cantidad de memoria física y tiempo de procesador disponible para llevar a cabo este proceso.

Figura 39. *Descripción detallada del caso de uso Filtrar Imagen*

Caso de uso	Umbralizar canal
Precondición	Se debe seleccionar una canal sobre el cual trabajar.

Descripción o flujo de sucesos	El usuario selecciona el canal activo y establece una valor de umbralización que luego los algoritmos utilizan para ejecutar la operación.
Caminos alternativos	El usuario puede requerir un valor umbral sugerido que es calculado por el sistema, tarea para la cual los algoritmo requieren calcular el histograma del canal activo.
Poscondiciones	Como resultado de la umbralización el canal debe tener algún elemento con cada uno de los dos valores definidos por el procedimiento de umbralización
Requisitos especiales	El sistema debe disponen de suficiente cantidad de memoria física y tiempo de procesador disponible para llevar a cabo este proceso. El valor umbral debe estar dentro de la escala de visualización.

Figura 40. *Descripción detallada del caso de uso Umbralizar canal*

Caso de uso	Aplicar morfología
Precondición	La canal sobre el que aplicarán los algoritmos de morfología debe ser binario, generalmente producto del procedimiento de umbralización
Descripción o flujo de sucesos	El usuario especifica una matriz binaria que define un elemento estructurante que se aplicará sobre el canal, luego el usuario elige un algoritmo de morfología a aplicar y este se ejecuta sobre el canal activo.
Caminos alternativos	Ninguno
Poscondiciones	
Requisitos especiales	El sistema debe disponen de suficiente cantidad de memoria física y tiempo de procesador disponible para llevar a cabo este proceso.

Figura 41. *Descripción detallada del caso de uso Aplicar morfología*

Caso de uso	Extraer señal a vectores
Precondición	La canal sobre el que aplicarán los algoritmos de extracción debe ser binario.
Descripción o flujo de sucesos	El usuario ejecuta el procedimiento de extracción en el cual se etiquetan las componentes conexas del canal y se extraen individualmente a vectores. Luego el usuario define con la ayuda del sistema cuales segmentos quedarán en la señal final.
Caminos alternativos	La definición de las componentes conexas finales se puede realizar sin la ayuda de la aplicación.
Poscondiciones	Las componentes conexas deben ser integradas en una sola con el fin de poderla manipular.
Requisitos especiales	El sistema debe disponen de suficiente cantidad de memoria física y tiempo de procesador disponible para llevar a cabo este proceso.

Figura 42. *Descripción detallada del caso de uso Extraer señal a vectores*

Caso de uso	Ajustar señal a Spline o Bezier
Precondición	La señal pluviométrica debe haber sido extraída y fusionada en un solo vector
Descripción o flujo de sucesos	El usuario escoge el tipo de ajuste que quiere ejecutar, luego los algoritmos determinan la posición de los nodos que definen la curva ajustada mediante el calculo numérico de derivadas de primer y segundo orden, finalmente la curva ajustada es mostrada en pantalla.
Caminos alternativos	Ninguno
Poscondiciones	Ninguna

Requisitos especiales	El sistema debe disponen de suficiente cantidad de memoria física y tiempo de procesador disponible para llevar a cabo este proceso.

Figura 43. *Descripción detallada del caso de uso Ajustar señal a Spline o Bezier*

Caso de uso	Mover nodo Spline o Bezier
Precondición	La señal debe haber sido ajustada a curvas Spline o Bezier y la edición del nodo debe ser interactiva.
Descripción o flujo de sucesos	El usuario selecciona un nodo haciendo clic sobre él y lo mueve arrastrándolo. Si se mueve un nodo Bezier los puntos de control del nodo deben ser movidos con el nodo, finalmente la curva resultante luego de cada movimiento es recalculada y desplegada en pantalla
Caminos alternativos	Ninguno.
Poscondiciones	Ninguna
Requisitos especiales	El sistema debe disponen de suficiente cantidad de memoria física y tiempo de procesador disponible para llevar a cabo este proceso.

Figura 44. *Descripción detallada del caso de uso Mover nodo Spline o Bezier*

Caso de uso	Agregar nodo Spline o Bezier
Precondición	La señal debe haber sido ajustada a curvas Spline o Bezier y la edición del nodo debe ser interactiva. No se pueden colocar dos nodos Spline en el mismo lugar en el tiempo.
Descripción o flujo de sucesos	El usuario elige un lugar sobre la gráfica en pantalla sobre el cual colocar el nuevo nodo y haciendo clic sobre el lugar el nodo es adicionado, finalmente la nueva curva es calculada y la grafica desplegada en pantalla.

Caminos alternativos	Ninguno
Poscondiciones	Ninguna
Requisitos especiales	El sistema debe disponen de suficiente cantidad de memoria física y tiempo de procesador disponible para llevar a cabo este proceso.

Figura 45. *Descripción detallada del caso de uso Agregar nodo Spline o Bezier*

Caso de uso	Eliminar nodo Spline o Bezier
Precondición	La señal debe haber sido ajustada a curvas Spline o Bezier y la edición del nodo debe ser interactiva. La curva debe tener al menos 3 nodos.
Descripción o flujo de sucesos	El usuario elige el nodo a eliminar y lo borra, luego la nueva curva es calculada y desplegada en pantalla.
Caminos alternativos	Ninguno
Poscondiciones	Ninguna
Requisitos especiales	El sistema debe disponen de suficiente cantidad de memoria física y tiempo de procesador disponible para llevar a cabo este proceso.

Figura 46. *Descripción detallada del caso de uso Eliminar nodo Spline o Bezier*

Caso de uso	Mover punto de control Bezier
Precondición	La señal debe haber sido ajustada a curvas Bezier y la edición del nodo debe ser interactiva.

Descripción o flujo de sucesos	El usuario selecciona un nodo y una de sus puntos de control, luego de que este es movido arrastrándolo con el Mouse la curva es recalculada y desplegada en pantalla. El usuario puede elegir conservar la suavidad de la curva al especificar que se conserve un ángulo de 180º entre los dos puntos de control del nodo, luego uno punto de control es movido por el sistema a medida que el usuario mueve el otro.
Caminos alternativos	Ninguno
Poscondiciones	Los puntos de control solo pueden ser editados moviéndolo, estos no pueden ser borrados ni agregados.
Requisitos especiales	El sistema debe disponen de suficiente cantidad de memoria física y tiempo de procesador disponible para llevar a cabo este proceso.

Figura 47. *Descripción detallada del caso de uso Mover punto de control Bezier*

4.2. DESCRIPCIÓN GENERAL DEL PROCESO APLICADO AL PLUVIOGRAMA

En esta sección se hace una breve descripción del funcionamiento en conjunto de los algoritmos de tratamiento de imágenes digitales y métodos numéricos que hacen parte de este trabajo de grado y el proceso en el que se utilizan.

4.2.1. Captura de la imagen digital

La captura de la imagen digital es externa e independiente de los algoritmos implementados y utilizados en la interfaz desarrollada para el tratamiento de la imagen pluviométrica, sin embargo, es un paso muy importante dentro de todo el proceso. De una correcta adquisición de la imagen digital depende que los tratamientos que posteriormente se hagan sobre ella arrojen resultados satisfactorios.

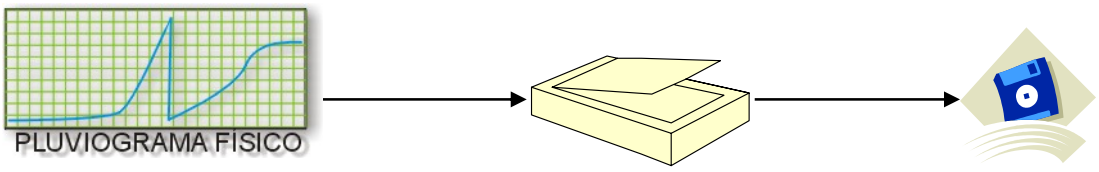


Figura 48. *Proceso de adquisición*

El proceso de adquisición tiene como objetivo obtener a partir del pluviograma físico una imagen digital de mismo a partir de su digitalización mediante un dispositivo de captura adecuado para la tarea, ya sea un scanner, una cámara digital o cualquier otro que sirva al propósito.

Como ya se mencionó, de una correcta captura dependen los resultados arrojados por los algoritmos implementados. Debido a esto se hace necesaria la utilización de dispositivos de calidad, que soporten alta resolución de imagen y de color si es posible.

Con esta intención, las condiciones de captura deben cumplir con los siguientes requisitos mínimos que definen un sencillo pero útil protocolo de captura para la imagen digital

Ambiente con buena iluminación: Se recomienda la utilización de luz alógena blanca y abstenerse de utilizar luz amarilla o focos parpadeantes como la luz de neón.

Ambiente exento de sombras en caso de utilizar luz natural para la captura con cámara digital.

El pluviograma debe estar en total contacto con la superficie para evitar ondas en el papel además de estar libre de arrugas, esto con el propósito de evitar sombras en el área al momento de la captura. Se recomienda para este propósito la utilización de un vidrio delgado anti-reflectivo sobre el pluviograma.

El dispositivo de captura debe soportar como mínimo resoluciones de color de 8 bits para generación de imágenes de 256 colores o niveles de gris, aunque se recomienda utilizar dispositivos que soporten profundidad de color de 24 bits.

La imagen resultan debe tener una resolución mínima de entre 78-80 pixeles de ancho por unidad de tiempo y de 39-41 pixeles de alto por unidad de medida de precipitación. Por ejemplo, la imagen digital de un pluviograma de 24 horas debería tener como mínimo 1896 pixeles de ancho y 400 pixeles de alto aproximadamente. Dentro de estas medidas no se consideran encabezados ni otras secciones del pluviograma físico carentes de información de la señal

La imagen capturada debe tener ningún o un bajo nivel de compresión cuando esta se hace mediante métodos que degradan la imagen, como es el caso de las imágenes en formato JPEG, esto se requiere debido a que la compresión de imágenes distorsiona la información contenida en ella dificultando su análisis.

La imagen digital final proporcionada debe estar en formato BMP con profundidad de color de 8 o 24 bits, no importa si se trata de una imagen a color a en escala de grises. Debido a que el dispositivo de captura puede devolver una imagen en un formato diferente, es posible que se requiera la utilización de un software externo para la conversión a formato BMP, muchos de estos pueden ser obtenidos gratuitamente vía Internet.

El pluviograma objeto de estudio debe estar en buenas condiciones físicas, exento de rupturas o rasgones además de poseer una línea nítida, definida y preferiblemente continua.

El pluviograma objeto de estudio debe contener solo un registro pluviométrico.

4.2.2. Lectura de la imagen y descripción de la información pictórica

El siguiente paso en el proceso luego de que la imagen es digitalizada es la lectura de esta y la organización y representación de la información pictórica contenida en

esta. Este paso y los algoritmos que intervienen en este son muy importantes debido a que cargan y describen internamente a la imagen digital de tal manera que permiten su manipulación.

Aunque en el manual técnico se hará una descripción mas a fondo de las clases y algoritmos implementados, se hace necesario mencionar en esta sección y de manera superficial aquellos que intervienen en este proceso.

Para empezar, tenemos a la clase “CCANAL”. Esta clase define de manera general una matriz dinámica de datos y un amplio grupo de funciones encaminadas a su manipulación. Esta clase es utilizada entre otras cosas para representar los datos de un canal (Rojo, Verde, Azul, Tono, Saturación, Luminosidad o Valor) de una imagen, mascaras o datos de operaciones intermedias entre canales.

Por otra parte tenemos a la clase “CIMAGEN”. Esta clase representa una imagen en cualquiera de sus formatos de representación RGB, HSI o HSV. La clase CImagen está internamente compuesta de tres objetos instanciados de la clase CCanal que representan cada uno de los tres canales de la imagen y de varias funciones encaminadas a manipularlos en conjunto, por ejemplo cuando queremos cambiar el formato de representación de RGB a HSI, cuando queremos rotar la imagen o extraer de ella una subsección de ella, todos estos procedimientos y otros requieren algoritmos que manipulen los tres canales a la vez.

Así, cuando una imagen en formato BMP es leída, la información referente a sus tres canales R, G y B es almacenada en los tres objetos CCanal pertenecientes a un objeto instanciado de la clase CImagen.

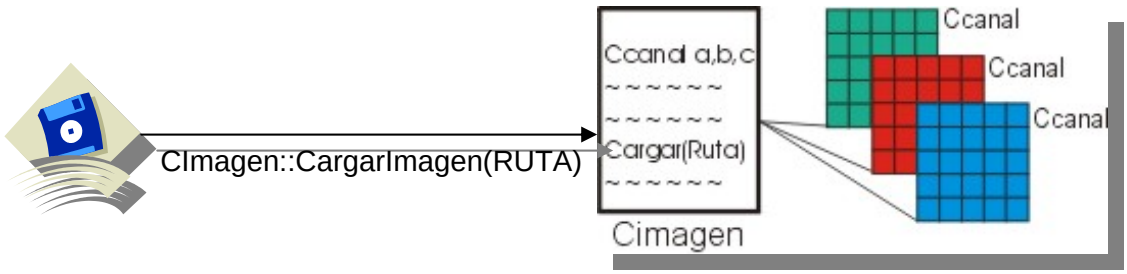


Figura 49. *Lectura de la imagen y descripción de la información pictórica.*

Una imagen digital es cargada dentro de un objeto CImagen mediante la ejecución de su función miembro “CargarImagen”, esta es una función polimorfíca en la que una de sus versiones recibe la ruta en disco de la imagen. Esta función implementa internamente un algoritmo que basado en el formato de archivo BMP, saca la información de este y luego la organiza dentro del objeto. En el capítulo dedicado al manual técnico se hace una profunda descripción del formato de archivo BMP y de las herramientas de Visual C++ 6.0 dedicadas a su manipulación.

4.2.3. Procesamiento de la información pictórica

Una vez que la información pictórica de la imagen se tiene a disposición en una estructura que permite y facilita su manipulación se procede a su procesamiento. El procesamiento de la imagen ahora almacenada consiste en la aplicación de varios algoritmos sobre el objeto CImagen y sus 3 miembros CCanal, implementados en funciones miembros de las clases y funciones de librerías independientes.

Dentro de las tareas ejecutadas por los algoritmos implementados dentro de las funciones disponemos entre otras las siguientes:

- Rotar una imagen o canal.
- Extraer una subsección de una imagen o canal.
- Cambiar el formato de representación de una imagen entre RGB, HSI y HSV.
- Aplicar filtros espaciales a un canal mediante la ejecución de la convolución entre dos canales.
- Calcular el histograma de un canal.
- Analizar un histograma para producir un umbral sugerido.
- Umbralizar un canal.
- Aplicar algoritmos básicos de morfología binaria sobre una canal (dilatación, erosión, apertura, cierre).

- Calcular el esqueleto de una imagen binaria mediante la trasformación de su eje medio.
- Etiquetar las componentes conexas dentro de una imagen binaria.

Aunque las clases y librerías de funciones implementadas en este proyecto cubren algunas otras tareas, las anteriormente mencionadas son las de mayor importancia dentro del mismo.

4.2.3.1. Rotar una imagen o un canal

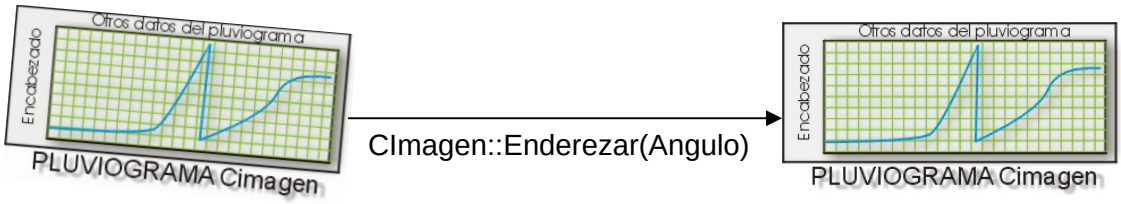


Figura 50. *Enderezado de una imagen.*

Luego de que se tiene una la información pictórica de una imagen organizada en un objeto CImagen es posible que debido a las condiciones de captura, esta se encuentre ligeramente inclinada con respecto a un eje horizontal imaginario. Es muy importante para la extracción de la información, que dicha inclinación sea disminuida o eliminada en lo posible, de manera que su ángulo de inclinación tienda a los 0º (cero grados).

4.2.3.2. Extracción de una subsección de la imagen

Una vez se tiene una imagen derecha, con un ángulo con respecto a la horizontal que tienda a cero, se debe definir la sección dentro de la imagen que será objeto del tratamiento, esto se debe a que algunas de las partes del papel de pluviograma contienen información que no son de nuestro interés en el ámbito del procesamiento de la imagen, tal es el caso de los encabezados, título o pies de pagina, además de componentes totalmente externos al pluviograma que se captan dentro de la imagen

durante el proceso de captura, como por ejemplo secciones de la mesa o lugar donde se postre el pluviograma pasa su captura.

Toda esta información pictórica extra se debe eliminar para que no forme parte de los posteriores tratamientos, en caso contrario modificarían sustancialmente el resultado durante cada paso siguiente. En general, para obtener una subsección de una imagen se deben definir las dimensiones de la subsección así como su origen.

CImagen::Subsección()

Figura 51. *Extracción de una subsección de una imagen.*

4.2.3.3. Cambiar el formato de representación

Luego de que se tiene definida la sección de la imagen con la que se desea trabajar, existen un par de procedimiento que se pueden o no aplicar a la imagen. Uno de ellos es la transformación del formato de representación original de la imagen, que en este caso es el RGB. Este formato de representación puede cambiarse al HSI o HSV. Una de las razones para hacer este cambio es la de poder trabajar con el canal del color, el canal H (Hue) de la imagen. Este canal contiene la información referente al color de la imagen y puede ser muy útil debido a que aunque en algunas imágenes, es difícil distinguirse las señal del fondo y la malla por su intensidad, puede ser más fácil distinguirse por su color, es decir, una malla verde puede tener la misma intensidad que una señal pluviométrica azul, pero definitivamente tienen diferente color y esto es mucho mas apreciable en el canal del tono que en cualquier otro.

4.2.3.4. Aplicación de filtros espaciales.

Otro de los procesos que pueden aplicarse a la imagen es un filtrado espacial al canal seleccionado para su análisis, en este proceso se pueden aplicar filtros de paso-alto, paso-bajo cualquier otro que pueda definirse por la convolución entre dos canales.

Dichos filtros deben estar encaminados a resaltar características de la imagen que puedan facilitar su posterior tratamiento y que contribuyan a la distinción de la señal pluviométrica de los demás elementos dentro de la imagen.

4.2.3.5. Calculo del histograma, el umbral y umbralización.

Una vez definido el canal que se desea exponer al tratamiento posterior, el siguiente paso es umbralizarlo con el objetivo de distinguir totalmente la señal de los demás elementos de la imagen como premier paso hacia la extracción final de esta. Con dicha intención en mente el histograma del canal debe ser calculado y luego sometido a un algoritmo de análisis que ejecuta sobre este el procedimiento de "PUNK" para el cálculo automático de un valor sugerido como umbral, dicho valor puede ser luego modificado a voluntad para finalmente aplicar la umbralización final sobre el canal.

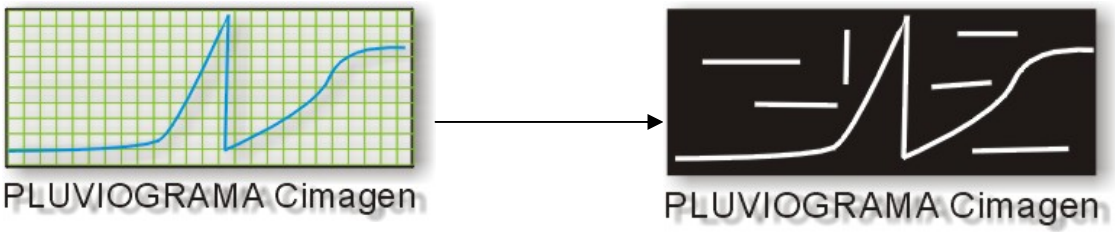


Figura 52. *Umbralización del canal.*

4.2.3.6. Aplicación de algoritmos básicos de morfología

El siguiente paso hacia la extracción de la señal es la utilización de algoritmos de morfología matemática, la aplicación de estos procedimientos esta básicamente encaminada a la consecución de dos objetivos:

Reparar segmentos de líneas dañadas durante el proceso de umbralización.
Eliminación de segmentos de líneas no pertenecientes a la señal pluviométrica (segmentos pertenecientes a la malla o rayones provenientes de comentarios hechos sobre el papel entre otros).

Por otra parte, para la extracción de la señal se requiere disponer en lo posible de líneas del mínimo ancho viable (un pixel). Debido a que generalmente la señal pluviométrica posee varios pixeles de ancho (cuando esta se encuentra bien definida), un algoritmo de transformación del eje medio es ejecutado sobre el canal para obtener el esqueleto de los segmentos contenidos en este. Este procedimiento es útil además cuando se presentan manchones en los segmentos de la señal debidos a la humedad de la zona en que se toma el pluviograma u otros agentes externos o internos que puedan malograr el registro de la precipitación

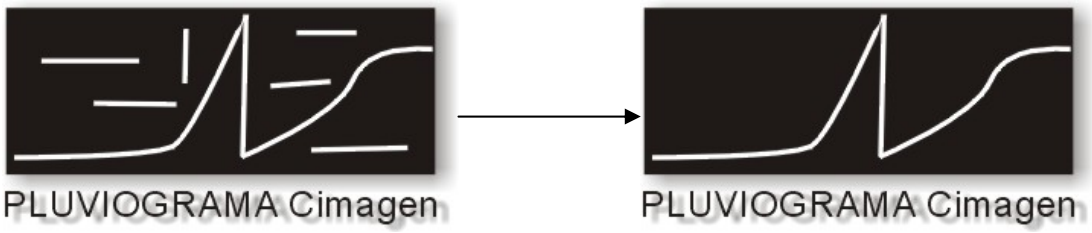


Figura 53. *Aplicación de morfología matemática a los datos de la imagen.*

4.2.3.7. Etiquetado de componentes conexas

El siguiente paso en el tratamiento de la imagen una vez umbralizada y ligeramente corregida por los algoritmos de morfología es el de etiquetado de componentes conexas. Es este paso se clasifican los puntos que componen la imagen según su 8-conectividad en segmentos conectados entre si y se asigna a cada elemento conexo un número o etiqueta que lo distingue de los demás

Figura 54. *Etiquetado de las componentes conexas del canal*

4.2.4. Extracción de la señal pluviométrica

Una vez disponemos de una canal cuyo contenido ya no es una imagen binaria, sino un conjunto de trazos etiquetados en sus componentes conexas, estamos en capacidad de extraer los primeros datos de la señal pluviométrica. Para llevar a cabo esta tarea es necesario antes proveer información a la interfaz acerca de las dimensiones del pluviograma original y el espacio que ocupa este en el dispositivo de contexto (la pantalla) con el objetivo de realizar un escalado proporcional que permita definir el valor de la señal en un instante de tiempo dado dependiendo de la posición que ocupe dentro de la vista. Con esta información y las componentes conexas del canal, las rutinas implementadas para la extracción de la señal pluviométrica están en capacidad de construir un "vector" de tiempo contra unidad de precipitación por cada componente conexa dentro del canal, colocando a disposición del usuario, estos vectores representativos de los trazos individuales dentro de la imagen, los cuales servirán para decidir finalmente cuales harán parte de la señal final y cuales no.

Cada vector es ordenado de menor a mayor por los valores que toman las x (el tiempo), y utilizando algunos criterios se eliminan las posiciones repetidas en el tiempo, dejando solo un punto por cada posición en x.

Una vez seleccionados los trazos que harán partes de la señal final, estos son “fundidos” o concatenados en un solo vector mediante líneas rectas que unen cada trazo independiente con el siguiente

Disponemos entonces de un trazo representativo de la señal pluviométrica que podemos ahora modificar para corregir posibles errores en sus valores provenientes desde el pluviograma físico u ocasionados en algunas e las etapas de procesamientos anteriormente descritas.

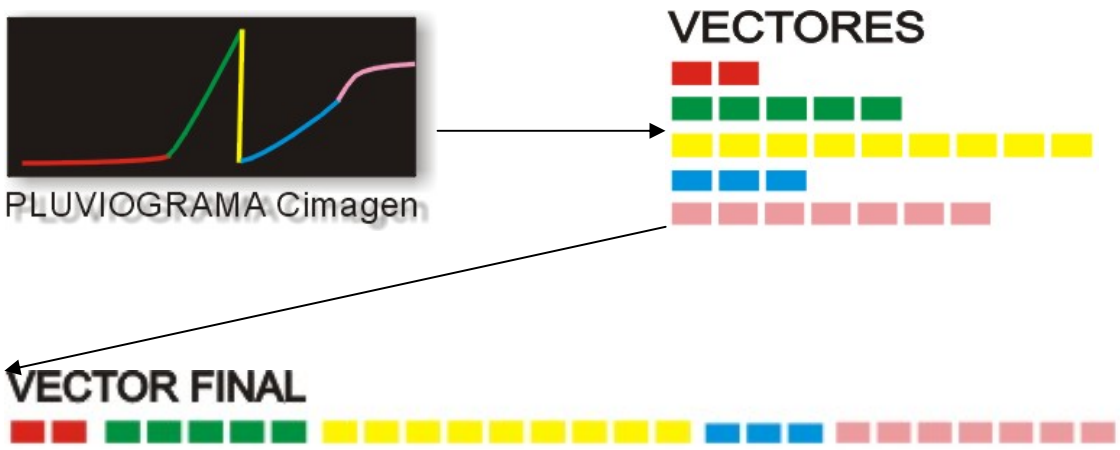


Figura 55. *Extracción de la señal pluviométrica a partir de componentes conexas.*

4.2.5. Corrección de la señal pluviométrica

Ahora que disponemos de un vector representativo de la señal pluviométrica extraída, estamos en disposiciones de modificarla modificando los valores en el tiempo y/o unidades de precipitación contenidas dentro del vector.

Con dicho propósito fueron implementados un conjunto de algoritmos que trabajan con procedimientos numéricos encaminados no solo a modificar esta información, sino además a proveer un útil conjunto de herramientas para la manipulación gráfica, interactiva y sobre todo coherente de esta información. Dichos algoritmos fueron encapsulados en dos jerarquías de clases independientes, una encaminada a hacer una representación de la señal mediante curvas Spline (trazador cúbico) y

otra a hacer una representación se la señal mediante secuencias de curvas Bezier (curvas paramétricas). Ambas alternativas ofrecen entre otras cosas, la posibilidad de modificación conservando la suavidad de las curvas que definen la señal y la interpolación de valores dentro de la misma.

En cualquiera de los casos la señal puede ser modificada alterando los parámetros que definen el trazador cúbico o la curva paramétrica que la representa, lo cual se puede hacer mediante una interfaz grafica implementada con la ayuda que a este propósito prestan los miembros de las clases, datos y funciones, que encapsulan la abstracción de datos y comportamiento de un Spline a un Bezier.

4.2.6. Almacenamiento de la información

Una vez se tiene la señal corregida, esta debe ser guardada para su posterior manipulación, ya sea para fines de análisis, modificación o impresión. Entre las alternativas tomadas en cuenta para este propósito se tuvieron entre otras, bases de datos y archivos planos. Finalmente se optó por crear un formato de archivo propio de la interfaz en la cual se aplicaron estos algoritmos, basado en la funcionalidad de seriación provista por Visual C++ a través de librerías MFC, para más información acerca de la seriación en Visual C++ refiérase al capítulo que trata el manual técnico en este mismo libro.

Este archivo de aplicación, es un archivo binario que guarda el vector final, la curva Spline o la curva Bezier representativa de la señal así como información extra que termina de definir el pluviograma como por ejemplo la fecha, intervalo de captura, estación origen, etc.

Además, y con el fin de conservar la portabilidad de los resultados finales arrojados por el proceso de extracción, la señal pluviométrica se puede exportar a un archivo de texto plano que contiene ‘N’ líneas, una por cada registro en la señal y dos columnas separadas por un tabulador, la primera guarda los datos del tiempo y la segunda guarda los datos de precipitación. De esta manera es posible someter estos datos a casi cualquier tipo de procesamiento o análisis en lenguajes

especializados, por ejemplo el MatLab, el cual puede leer fácilmente este archivo y reproducir el pluviograma en base a los datos guardado

5. MANUAL TECNICO

En este capítulo se hace una documentación de la programación implementada en este proyecto (clases, librerías de funciones, tipados de usuario etc.). Inicialmente se hace una explicación del la estructura del formato de archivo BMP y las estructuras provistas por Visual C++ para su manipulación, luego se da una descripción de las clases desarrolladas en este proyecto así como de las clases utilizadas de las MFC que tuvieron una mayor relevancia en el desarrollo, se sigue con la documentación acerca de la librerías de funciones y finalmente se trata el tema de la seriación como técnica utilizada para el almacenamiento permanente de información.

5.1. MAPAS DE BITS INDEPENDIENTES DEL DISPOSITIVO

Si decimos que un mapa de bits es independiente del dispositivo es porque contiene una tabla de color. Un mapa de bits independiente del dispositivo (DIB) no es un objeto GDI. Por lo tanto, al GDI no puede almacenar un DIB. Lo que hace la GDI es obtener el DIB y convertirlo en un DDB compatible con algún dispositivo de salida; en este caso se pierde la información de color del DIB. El propósito principal de los DIB es permitir intercambiar mapas de bits entre programas.

Un mapa de bits independiente del dispositivo utiliza un formato que permite varias resoluciones de color. En contraste con un mapa de bits dependiente del dispositivo (DDB), un DIB utiliza un formato externo y cuando se carga, aparece en el sistema igual que un objeto que hubiera sido creado por una aplicación utilizando CreateBitmap, CreateCompatibleBitmap, CreateBitmapIndirect. o CreateDIBitmap. Un DIB normalmente se transporta en meta archivos (*meta files* – un meta archivo es una descripción vectorial de la imagen y no una representación digital de la misma), en ficheros BMP, o en el portapapeles (formato de datos CF_DIB).

5.1.1. Formato de un DIB

Un DIB consta de dos partes: una cabecera que describe el formato de los bits y los *bits* del mapa de bits (el array de bits empieza por la línea inferior de píxeles, en lugar de por la superior).

Cabecera	Array de bits que forman la imagen

La cabecera contiene el formato de color, una tabla de colores y el tamaño del mapa de bits. La estructura actual de un DIB soporta seis resoluciones de color: 1 bit (blanco y negro). 4 bits (16 colores), 8 bits (256 colores), 16 (65536 colores), 24 bits (16 millones de colores) y 32 bits. En los DIB de resolución 1 bit, 4 bits y 8 bits, los píxeles están definidos por índices de una tabla de colores. Por ejemplo, en un DIB de 16 colores cada byte del mapa de bits almacena dos píxeles (cada 4 bits seleccionan una de las 16 entradas de una tabla de colores). En la resolución de 24 bits cada pixel está descrito por un valor de 24 bits (valor RGB), 1 byte para cada uno de los colores rojo, verde y azul.

Con DIB, cada dispositivo visualiza la imagen en función de su resolución de color. Una aplicación puede guardar una imagen en un DIB y posteriormente visualizarla independientemente del dispositivo de salida; por lo tanto, una aplicación no requiere crear una versión de cada imagen por cada tipo de dispositivo.

La extensión por omisión de un fichero que almacena un DIB es BMP. Un fichero de este tipo consta de una estructura BITMAPFILEHEADER seguida de otra estructura BITMAPINFOHEADER, de la tabla de colores y del propio mapa de bits. La información de estas estructuras se indica a continuación.

5.1.1.1. BITMAPFILEHEADER

La estructura BITMAPFILEHEADER contiene información acerca del tipo y tamaño del fichero que contiene el mapa de bits, así como de la colocación del mismo dentro del fichero.

Miembro	Tipo	Descripción
bfType	WORD	Tipo del fichero. Debe ser “BM”.
bfSize	DWORD	Tamaño del fichero en bytes.
BfReserved1	WORD	Reservado. Debe ser 0.
bfREserved2	WORD	Reservado. Debe ser 0.
bfOffBits	DWORD	Desplazamiento desde el comienzo del fichero hasta los bits del mapa de bits.

5.1.1.2. BITMAPINFOHEADER

La estructura BITMAPINFOHEADER contiene información acerca de las dimensiones y el formato de color del DIB.

Miembro	Tipo	Descripción
biSize	DWORD	Tamaño, en bytes de la estructura.
biWidth	LONG	Ancho del mapa de bits en píxeles.
biHeight	LONG	Alto del mapa de bits en píxeles.
biPlanes	WORD	Siempre es1. Para definir la resolución de color, el DIB cuenta con el dato biBitCount.
biBitCount	WORD	Resolución de color (1, 4, 8, 16, 24 ó 32)
biCompression	DWORD	Tipo de compresión.
biSizeImage	DWORD	Tamaño en bytes del mapa de bits.
biXpelsPerMeter	LONG	Píxeles/m. en el eje X.
biYpelsPerMeter	LONG	Pixeles/m. en el eje Y.
biClrUsed	DWORD	Número exacto de colores en la tabla.
biClrImportant	DWORD	Número de colores que son importantes.

Cada bloque de biBitCount bits define un índice correspondiente a un elemento de la tabla de colores para resoluciones de 1, 4 u 8 bits, o un valor RGB que define directamente el color para resoluciones de 16, 24 o 32 bits,

El dato biCompression especifica el tipo de compresión. La compresión reduce el almacenamiento requerido. Puede ser uno de los cuatro valores siguientes: BI_RGB, BI_RLE4, BI_RLE8 O BI_BITFIELDS. La opción más común y útil BI_RGB, define un DIB en el que todo es como se ve, el mapa de bits no esta comprimido. Las opciones BI_RLE4 y BI_RLE8 especifican que el DIB de resolución 4 u 8 bits se almacenará comprimido utilizando un formato RLE (runlength encoding) y BI_BITFIELDS especifica que el mapa de bits no está comprimido y que la tabla de color esta formada por tres dobles palabras que especifican las componentes rojo, verde y azul, respectivamente de cada pixel; cuando un DIB se comprime es preciso especificar el valor biSizeImage; un valor biSizeImage distinto de cero indica a la aplicación el tamaño en bytes que necesita para cargar el mapa de bits; para BI_RGB, biSizeImage puede ser cero. Calcular el tamaño de un mapa de bits no es complejo.

BiSizeImage = (((biwidth*biBitCount)+31)&-31>>3) * biHeight

La alineación para cada línea es DWORD (llenar cada línea hasta completar grupos de 4 bytes).

El valor biClrUsed proporciona una forma de obtener tablas de colores más pequeñas. Cuando este valor es 0, el número de colores en la tabla depende del campo biBitCount: 1 indica 2 colores, 4 indica 16, 8 indica256, y para el modo de compresión BI_RGB, 16, 24 y 32 indican que no hay tabla de colores. Un valor distinto de 0 especifica el número exacto de colores en la tabla. Así, por ejemplo, si un DIB de 8 bits de resolución de color utiliza solo 26 colores, entonces solo necesita definirse en la tabla esos 26 colores, y biClrUsed se pone a 26. Este campo no puede ser utilizado durante una operación GetDIBits. Para un DIB de 16, 24 o 32 bits de resolución, un valor distinto de 0 indica la existencia de una tabla de colores que la aplicación puede utilizar como referencia de colores.

Resumiendo, si biClrUsed es cero, el mapa de bits utiliza el máximo número de colores (2^biBitCount). Si es distinto de cero y biBitCount es menor que 16, biClrUsed especifica el número real de colores; y si biBitCount es mayor o igual que

16, entonces biClrUsed especifica el tamaño de la tabla de color utilizada para optimizar las paletas de color Windows.

El valor biClrImportant especifica que los primero biClrImportant colores de la tabla son importantes para el DIB. Si el resto de los colores no están disponibles, la imagen todavía retiene su significado en un amanera aceptable. Cuando este valor es 0, todo los colores son importantes; o, más bien, no se ha considerado su importancia relativa.

5.1.1.3. BITMAPINFO

La estructura BITMAPINFO combina la estructura BITMAPINFOHEADER y la tabla de color para proporcionar una definición completa de las dimensiones y de los colores del DIB.

Miembro	Tipo
bmiHeader	BITMAPINFOHEARDE
bmiColors	Array de tipo RGBQUAD/datos de tipo DWORD

El miembro bmiColors puede ser un array de enteros sin signo de 16 bits que especifique los índices correspondientes a la paleta lógica actual, o un array de tipo RGBQUAD con los valores RGB explícitos. En el primer caso, una aplicación que utilice el mapa de bits debe invocar a las funciones DIB, tales como CreateDIBitmap, CreateDIBPatternBruh y CreateDIBSection. Con el valor DIB_PAL_COLORS para el perímetro wUsage. En el case de que el mapa de bits vaya a ser transferido a otra aplicación, el miembro bmiColors no debe contener índices sino valores RGM.

Un array de estructura RGBQUAD define una tabla de colores. Cada estructura define un valor de color RGB. Por ejemplo, si el campo biClrUsed vale 0, un DIB de 1 bit de resolución de color requiere 2 estructuras, un DIB de 4 bits requiere 16 estructuras y de 8 bits, 256 estructuras. La información que almacena cada estructura es:

Miembro	Tipo	Descripción
rgbBlue	BYTE	Intensidad de azul
rgbGreen	BYTE	Intensidad de verde
rgbRed	BYTE	Intensidad de rojo
rgbReserved	BYTE	Reservado. Valor 0

Como ya hemos indicado anteriormente, el tamaño de la tabla de colores depende del valor del biBitCount (y puede ser sobreescrito utilizando el campo biClrUsed). Para biClrUsed cero y biBitCount 1, 4 u 8, bmiColors apunta a un array de 2, 16 o 256 entradas respectivamente; si biCompression vale BI_RGB, que es lo normal, y biBitCount es 16, 24 u 32, bmiColors es NULL. Según esto, el número de colores y el tamaño en bytes de la tabla de colores puede calcularlos así:

```
 If (!(nNumColores = biClrUsed))
[
        if (biBitCount < 16)
          nNumColores = 1 << biBitCount:
]
 nTamTablaClr = nNumColores * sizeof(RGBQUAD):
```

5.2. DOCUMENTACION DE CLASES

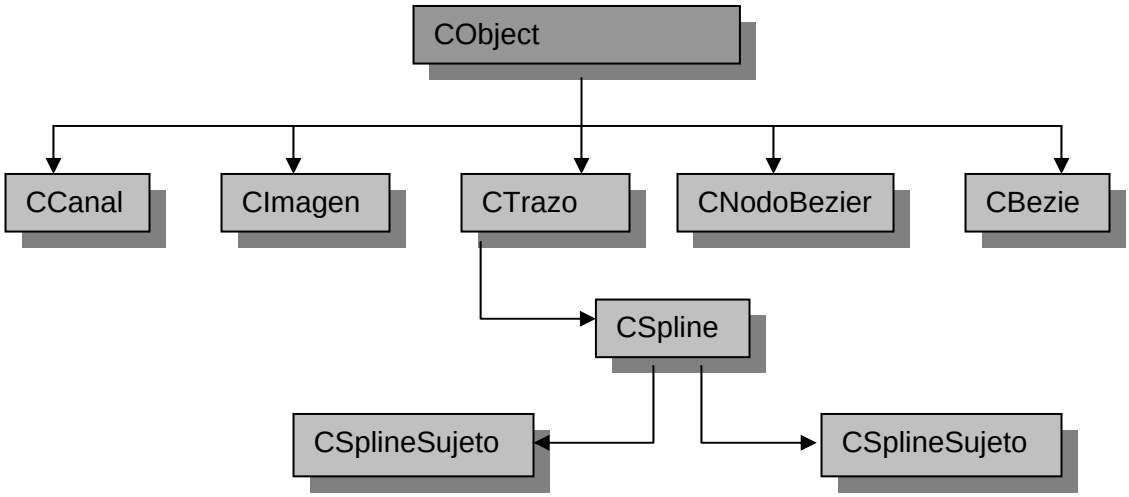


Figura 56. *Diagrama jerárquico general de las clases implementadas, no se incluyen las pertenecientes la las librerías MFC*

5.2.1. CObject

La clase CObject es una clase abstracta de la jerarquía de clases de la biblioteca MFC (Microsoft Foundament Clases). La definición de esta clase puede verla en el fichero afx.h

La clase CObject es una clase raíz para la mayoría de las clases de la biblioteca MFC. Esta clase contiene muchas características útiles, como seriacíon, diagnósticos o información sobre clases, que podemos incorporar a nuestros programas simplemente derivando una clase de ella. El coste de derivar una clase de CObject es mínimo y a cambio, dicha clase derivada tendrá además, cuatro funciones virtuales que veremos a continuación y un objeto **CRuntimeClass.** La funcionalidad de CObject la podemos agrupar de la siguiente forma:

Creación y borrado de objetos.
new. Para la versión *Release* de un programa, el operador **new** opera una función parecida a **malloc**. En la versión *Debug*, el operador **new** participa en un plan de verificación diseñado para detectar lagunas de memoria.

Si utilizamos la línea de código

```
#define new DEBUG_NEW
```

antes de cualquier implementación en un fichero .cpp, entonces se utilizara la segunda versión de new, almacenando el nombre del fichero y el número de línea, en el bloque destinado para ser posteriormente informados.

Aun cuando no utilicemos DEBUG_NEW en modo Debug, todavía conseguiremos detectar lagunas de memoria, pero sin el número de línea y sin el nombre del archivo fuente mencionados anteriormente.

Soporte para seriación (guardar uno o más objetos en un disco, los cuales, más tarde, pueden ser recuperados y restaurados en memoria):

IsSerializable. Devuelve un valor distinto de cero si el objeto es seriable; en otro caso devuelve cero. Para que los objetos de una clase se puedan seriar, la declaración de la clase debe incluir la macro DECLARE_SERIAL y su implementación IMPLEMENT_SERIAL.

Serialize. Es una función virtual que permite leer o escribir un objeto de/en un fichero en el disco vinculado a un objeto **CArchive.** La función **Serialize** debe ser redefinida para cada clase cuyos objetos desee seriar. Esta función debe invocar primero a la función **Serialize** de la clase base.

Diagnósticos.
Assert Valid. Es una función virtual que permite verificar los supuestos acerca de la validez del estado interno de un objeto de una clase derivada de **CObject.** Cuando escribamos nuestra propia clase derivada, tendremos que redefinir esta función para proporcionar los servicios de diagnósticos que creamos necesarios.

Dump. Es una función virtual que permite descargar el contenido de un objeto en otro de tipo **CDumpContext** a efectos de depuración.

Obtener información de un objeto durante ejecución.
GetRuntimeClass. Hay una estructura **CruntimeClass** para cada clase derivada de **CObject** que contiene información sobre el nombre de la clase, tamaño del objeto, clase base, etc. Esta información es virtual y devuelve un puntero a la estructura **CRuntimeClass** asociada con la clase de objeto que recibe el mensaje **GetRuntimeClass.**

IsKindOf. Verifica si un objeto es de la clase especificada o de una clase derivada de ella. Esta función se puede utilizar con clases que incluyan las macros DECLARE_DYNAMIC o DECLARE_SERIAL. Ver más adelante la macro RUNTIME_CLASS

La sintaxis correspondientes a estas funciones, así como ejemplos de utilización de las mismas, puede verlas en su pantalla utilizando la ayuda en línea correspondiente a la clase (busque por el nombre de la clase).

Para utilizar esta funcionalidad, simplemente tiene que derivar su clase de la clase **CObject.** Por ejemplo,

```
Class CMiClase :  public Cobject
{
      //declaración de la clase
};
```

Cuando derivamos una clase de **CObject** podemos escoger entre cuatro niveles de funcionalidad:

Funcionalidad básica. Solo proporciona diagnósticos relacionados con la gestión de memoria.
Nivel 1. más soporte para obtener información de objetos durante la ejecución.
Nivel 2. más creación dinámica de objetos.
Nivel 3. más seriación.

Una clase derivada de **CObject** que luego vaya a ser utilizada como clase base, debería incluir por lo menos soporte para obtener información y para la seriación, si pensamos que en un futuro esta característica puede ser útil.

Para escoger el nivel de funcionalidad adecuado fíjese en el cuadro que se muestra a continuación. Las macros elegidas debe escribirse en la declaración y en la implementación de la clase, respectivamente.

Macro	Información sobre clases	Creación Dinámica	Seriación

Funcionalidad básica	No	No	No
DECLARE_DYNAMIC			
IMPLEMENT_DYNAMIC	Si	No	No
DECLARE_DYNCREATE			
IMPLEMENT_DYNCREATE	Si	Si	No
DECLARE_SERIAL			
IMPLEMENT_SERIAL	Si	Si	Si

Las macros DECLARE_DYNAMIC e IMPLEMENT_DYNAMIC permiten obtener durante la ejecución información acerca de objetos de una determinada clase. La macro DECLARE_DYNAMIC hay que incluirla en la declaración de la clase (fichero.*h*) y la macro IMPLEMENT_DYNAMIC en la implementación (fichero *.cpp*) para que sea evaluada solo una vez. Por ejemplo.

```
// Fichero .h
class CMiclase : public CObject
[
DECLARE_DYNAMIC( CMiclase )
Public:
CMiclase() [ ]:
//sigue la declaración de la clase
 ]:
// Fichero .cpp
IMPLEMENT_DYNAMIC( CMiclase, CObject )
// ...
```

Ahora puede acceder a la información de CMiclase durante la ejecución. Para ello, utilice la función miembro **IsKindOf** de la clase **CObject** y la macro **RUN. TIME_CLASS.** Por ejemplo,

```
If ( MiObjeto.IsKindOf(RUNTIME_CLASS( CMiclase )) )
```

CMiclase OtroObjeto = MiObjeto:

La función **IsKindOf** devuelve **TRUE** si el objeto que recibe el mensaje pertenece a la clase específica o a una clase derivada de ella.

La macro RUNTIME_CLASS devuelve un puntero a una estructura **CRuntimeClass** asociada con la clase especificada. Los miembros de esta estructura contiene información tal como nombre de la clase, tamaño del objeto, clase base, etc. La estructura **CRuntimeClass** está definida en el fichero *afx.h.*

CRuntimeClass * pStCMiclase = RUNTIME_CLASS(CMiclase):

Las macros DECLARE_DYNCREATE e IMPLEMENT_DYNCREATE permiten crear objetos de una clase derivada de **CObject** durante la ejecución por ejemplo, cuando se leen objetos del disco durante un proceso de seriación.

Las macros DECLARE_SERIAL e IMPLEMENT_SERIAL generan el código necesario para que los objetos de la clase derivada de **CObject** que las incluya, puedan seriarse (acción de leer o escribir objetos en un fichero en el disco).

5.2.2. CArchive

La clase **CArchive** es una clase perteneciente a las liberarías de las MFC y no tiene una clase base. Su función es permitir almacenar objetos de forma permanente, normalmente en un disco, de forma que persistan cuando ellos sean eliminados de la memoria. Más tarde, estos objetos podrán ser recuperados y reconstruidos de nuevo en la memoria. Este proceso recibe el nombre de *seriación*. La definición de esta clase puede verla en el fichero *afx.h*

Según lo expuesto, un objeto **CArchive** está asociado con un fichero en disco por lo tanto, antes de crear el objeto **CArchive,** hay que crear un objeto **CFile** que representa al fichero que almacena o que almacenará los objetos, y después

conectar el objeto **CArchive** al objeto **CFile.** También se debe especificar si la operación que se va a realizar es leer (**load**) o escribir (**store**).

La funcionalidad de esta clase está soportada por el dato miembro **m_pDocument** y por varios conjuntos de funciones miembro, que podemos clasificar de la forma siguiente:

Un puntero al objeto **CDocument** que está siendo seriado. El sistema inicializa **m_pDocument** para que apunte al documento cuando un usuario emite la orden *Abrir Archivo o Guardar.* Un uso común de este puntero es permitir a todos los objetos que están siendo seriados acceder, si es necesario, al documento.
Un constructor **CArchive** y un destructor **~CArchive.**
La función **Close** que cierra el fichero asociado a un objeto **CFile.**
Entrada/salida:
Flush. Envía los datos aún no escritos contenidos en el *buffer* del objeto **CArchive.**
operator >> y operator <<. El operador de extracción carga del objeto **CArchive** especificado a su izquierda, el objeto especificado a su derecha; y el operador de inserción almacena el objeto especificado a su derecha en el objeto **CArchive** especificado a su izquierda. El operando de la derecha es un objeto si es de un tipo predefinido, o un puntero a un objeto si es de un tipo definido por el usuario (una clase).

Read. Lee un número especificado de bytes del objeto **CArchive.**
Write. Escribe un número especificado de bytes en el objeto **CArchive.**

Estado
GetFile. Obtiene el puntero al objeto **CFile** que esta ligado al objeto **CArchive.** que recibe este mensaje.
IsLoading. Indica si el objeto **CArchive** se está utilizando para cargar datos.
IsStoring. Indica si el objeto **CArchive** se está utilizando para guardar datos.

Entrada/salida de objetos
ReadObject. Invoca a la función **Serialize** de un objeto para cargar datos.

WriteObject. Invoca a la función Serialize de un objeto para guardar datos.

La sintaxis corresponde a estas funciones, así como ejemplos de utilización de las mismas, puede verla en su pantalla utilizando la ayuda en línea correspondiente a la clase (busque por el nombre de la clase).

5.2.3. CFile

La clase **CFile** es la clase base de las clases que proporcionan las MFC para manipular ficheros en el disco. La clase **CFile** trabaja conjuntamente con la **CArchive** para soportar la *seriación* (proceso de almacenar y recuperar objetos de un fichero). La definición de esta clase puede verla en el fichero *afx.h.*

La funcionalidad de esta clase está soportada por varios conjuntos de funciones miembro. Estas funciones se clasifican de la forma siguiente:

Un constructor **CFile** Construye un objeto y lo asocia con el fichero especificado.
La función **Open** que abre un fichero y lo vincula al objeto **CFile** que recibe lel mensaje. Se utiliza con el constructor sin argumentos. Esta función devuelve **FALSE** si ocurre un error o **TRUE** en caso contrario.
La función **Close** que cierra el fichero asociado a un objeto **CFile** lo que implica romper el vínculo existente entre ambos y borrar el objeto **CFile**
Entrada/salida:
Flush. Envía los datos aún no escritos, al fichero.
Read. Lee del fichero, a partir de la posición actual.
Write. Escribe al fichero, a partir de la posición actual.
Posición:
Seek. Permite colocar el puntero de L/E en una determinada posición dentro del fichero. Esto permite el acceso aleatorio.
SeekToBegin. Coloca el puntero de L/E al principio del fichero.
SeekToEnd. Coloca el puntero de L/E al final del fichero.
GetLength. Obtiene la longitud del fichero.

SetLength. Cambia la longitud del fichero. El fichero sería aumentado o truncado a la nueva longitud.

Bloqueo:

LockRange. Bloquea un número de bytes en el fichero para prevenir que otros procesos accedan a ellos.

UnLockRanger. Desbloquea un número de bytes en el fichero.

Estado:

GetPosition. Retorna la posición de L/E actual dentro del fichero.

Getstatus. Recupera el estado del fichero asociado con el objeto **CFile** que recibe el mensaje y lo almacena en una estructura de tipo **CFileStatus.**

Funciones **static:**

Rename. Cambia el nombre del fichero especificado.

Remove. Borra el fichero especificado.

SetStatus. Obtiene el estado del fichero especificado.

SetStatus. Asigna el estado al fichero especificado.

La sintaxis correspondiente a estas funciones, así como ejemplo de utilización de las mismas, puede verla en su pantalla utilizando la ayuda en línea correspondiente a la clase (busque por el nombre de la clase).

5.2.4. CObArray

La clase CObArray pertenece a las librerías MFC y soporta colecciones de punteros a CObject. Este tipo de colecciones son similares a los *arrays* de C, con la diferencia de que pueden variar su tamaño dinámicamente. El índice correspondiente al primer elemento de la colección es O. Con respecto al límite superior, podemos decidir entre fijarlo o permitir a la colección expandirse cuando se añada un objeto pro encima del límite actual. En este último caso, la memoria extra necesaria se asigna contiguamente a partir del límite superior, añadiendo elementos nulos si es necesario.

La clase CObArray está declarada en el fichero afxcoll.h y su funcionalidad está soportada por varios conjuntos de funciones miembro clasificadas de la forma siguiente:

Un constructor CObArray y un destructor ~ CObArray.

Límites

GetSize. Devuelve el número de elementos del array.

GetUpperBound. Retorna el índice superior del array. El índice inferior es cero, y el superior es GetSize – 1.

SetSize. Fija el número de elementos para un array vacío o existente. Si el nuevo tamaño es más pequeño que el actual, el array se trunca al nuevo tamaño y se libera la memoria no utilizada y si es más grande se amplía en lo necesario, asignando más memoria contigua.

Liberar memoria:

FreeExtra. Libera la memoria no utilizada por encima del límite superior actual.

RemoveAll. Libera toda la memoria ocupada por todos los elementos del array, pero no libera la memoria ocupada por los objetos apuntados.

Acceso a los elementos del array (los elementos son punteros a CObArray):

GetAt. Retorna el elemento del array especificado. Un índice fuera de límites solo es detectado en la versión *Debug.*

SetAt. Reemplaza el elemento del array especificado. El índice que especifica la posición del elemento debe estar dentro de los límites del array. SetAt no incrementa el tamaño del array; para esto tiene que utilizar la función SetAtGrow. Un índice fuera de límites solo es detectado en la versión *Debug.*

Insertar/eliminar elementos:

SetAtGrow. Pone el elemento del array en la posición especificada. Si el índice especificado es mayor que el límite superior, el array se agranda automáticamente en el número necesario de elementos.

Add. Añade un nuevo elemento al final del array.

InsertAt. Inserta un elemento (o todos los elementos de otro array) a partir de una posición especificada.

RemoveAt. Borra uno o más elementos del array a partir de una posición especifica. Se libera la memoria utilizada por los punteros del array, pero no la

ocupada por los objetos apuntados. Después del proceso, el array queda reorganizado.

Operadores:

Operator [ ]. Este operador puede utilizarse tanto al lado izquierdo como al lado derecho del operador de asignación (con arrays no const). Significa esto que es un buen sustituto para las funciones SetAt y GetAt.

A continuación se muestra un ejemplo que le enseña como utilizar la clase CObArray:

```
#include <iostream.h>
#include <afxcoll.h>

class CMiclase : public CObject
[
   // ...
]:
void main()
[
   COArray a:
         a.SetSize(10): // el array a tiene 10 elementos con
 valor NULL
 cout << a.GetSize() <<end]: / escribe 10
            a.Add(new CMiclase): // añadir un elemento al final del
      array y asignar     // asignarle un objeto CMiclase
      cout <<a.GetSize() <<end]: //escribe 11
   a.SetAtGrow(30, NULL): //agrandar el array hasta el
   elemento              // de indice 30 inclusive. Los
   elementos             // añadidos tiene valor Null
         cout <<a.GetSize() <<end]: //escribe 31
         // ...
// liberar la memoria ocupada por los objetos asignados   al array
         for (int i = 0 <a.GetSize(): i++)
```

```
        delete a[i]:
// liberar la memoria ocupada por los elementos del  array
a.RemoveAll():
 ]
```

5.2.5. CCanal

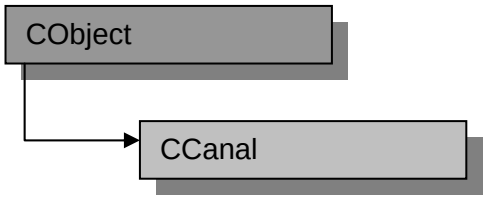


Figura 57. *Diagrama jerárquico de la clase CCanal.*

La clase **CCanal** encapsula la abstracción de datos que se hizo de un canal de una imagen, independientemente del formato en que esta se encuentre representada (RGB, HSI o HSV).

La clase CCanal funciona básicamente como una matriz de datos que puede cambiar de dimensiones de manera dinámica, de esta manera, la utilización de la clase Canal no está restringida solo al uso de esta como el canal de una imagen y por el contrario se convierte en el corazón de los algoritmos diseñados e implementados en este proyecto por ser objetivo y resultado de la aplicación de estos.

La clase CCanal, al igual que todas las demás clases utilizadas en este proyecto, se encuentra heredada de la clase CObject, la cual ya fue analizada y explicada en un anterior apartado de este capítulo. Dicha relación fue creada con el objetivo de brindar a la clase, ventajas de la seriación, creación dinámica de objetos y depuración avanzada para la detección de lagunas durante la implementación, lo cual resulta muy importante ya que como veremos a continuación, es muy importante el manejo de la memoria en esta clase.

- PROPIEDADES
    - m_pc
    - m_Demensiones
    - m_Tipo
- MÉTODOS
    - CCanal
    - Crear
    - Destruir
    - GetDimensiones
    - GetTipo
    - SetTipo
    - GetVal
    - SetVal
    - IsCanal
    - IsDimenciones
    - subsección
    - GetMayor
    - GetMenor
    - Redondear
    - Invertir
    - CrearBuffer
    - CrearBitmap
    - Serialize

```
# include <Canal.h>
```

5.2.5.1. Datos miembros de la clase

5.2.5.1.1. CCanal::m_pC

Este dato miembro es el corazón de la clase y como se mencionó anteriormente, es el corazón de los algoritmos implementados. Este funciona básicamente como un puntero que apunto a un vector de punteros de datos de tipo *DECIMAL*, de esta manera, una matriz dinámica queda definida por este dato miembro.

En la siguiente figura podemos ves la estructura que establece este miembro:

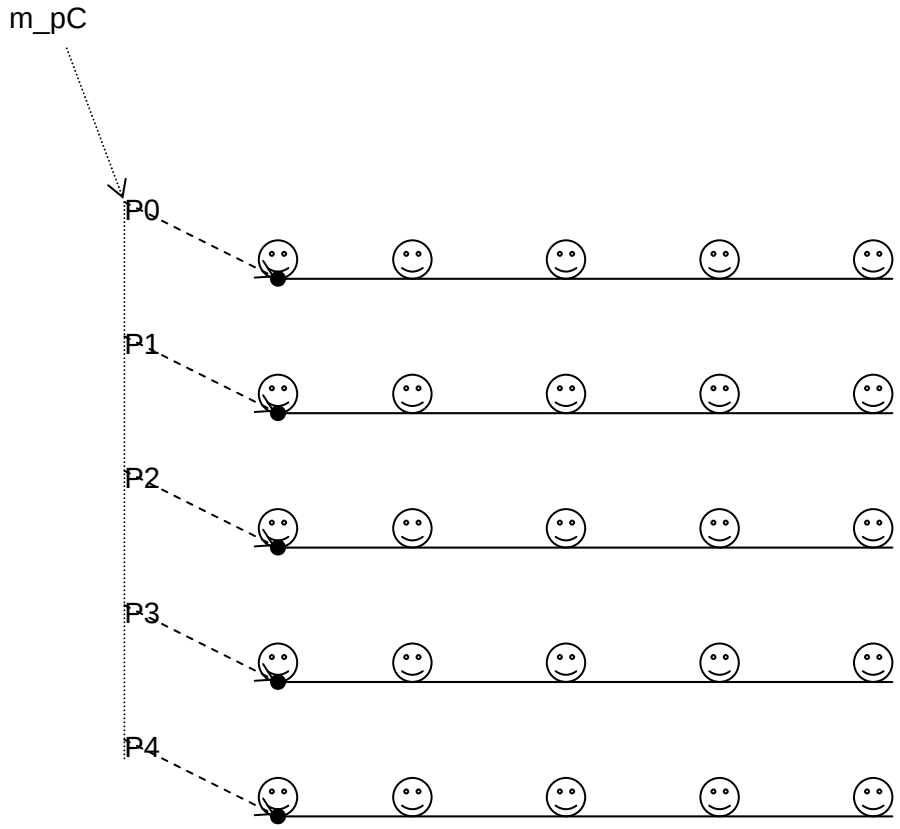


Figura 58. *Imagen representativa del dato miembro m_pDC.*

Utilizando la asignación de memoria dinámica sobre los punteros mediante el operador **new**, el cual ya vimos anteriormente, podemos definir una matriz bidimensional del tamaño según nuestras necesidades.

Por otra parte, la utilización directa de punteros para la creación de esta matriz en lugar de utilizar otros tipos de estructuras aporta a los algoritmos una gran velocidad de ejecución, ya que se accede directamente a la memoria sin necesidad de una interfaz que le quite agilidad al proceso. Por supuesto, el manejo de la memoria de una manera tan directa requiere ciertas responsabilidades por parte del programador, entre otras, la precaución de acceder a índices pertenecientes a los segmentos de memoria reservados, la inicialización de los segmentos y la liberación de la memoria en el momento de dejar de utilizarla con el fin de evitar molestas lagunas.

5.2.5.1.2. CCanal::m_Dimensiones

Este dato miembro contiene las "*dimensiones del canal"* o el tamaño de la matriz representada por esta clase, es un dato de vital importancia all momento de querer liberar la memoria utilizada por los objetos instanciados de esta clase y al querer acceder a la información de la matriz sin riesgo de violar las fronteras de los segmentos de memoria reservados para dicho propósito.

El objeto CSize pertenece a las MFC, su funcionamiento es básico y representa un tamaño en dos dimensiones, este objeto tiene dos datos miembros públicos, ***cx*** y ***cy***, para nuestra clase, el primero representa las dimensiones del canal a lo largo del eje X y el segundo representa las dimensiones del canal a lo largo del eje Y.

5.2.5.1.3. CCanal::m_Tipo

La definición de este tipo de datos se encuentra en el fichero ***Tipados.h***
Este es un tipo de dato enumerado definido por el usuario, su definición es la siguiente:

enum TIPO_CANAL {CANAL_RGB,CANAL_R,CANAL_G,CANAL_B,CANAL_H,CANAL_S,CANAL_V,CANAL_ND,CANAL_I,CANAL_CA,CANAL_CB,CANAL_CC};

El objetivo de este miembro de la clase es identificar a la matriz como una canal de tipo RGB, un canal Rojo, Verde o Azul cuando la imagen se encuentra en formato RGB, un canal de Tono (H), Saturación (S) o Intensidad (I) cuando la imagen se encuentra en formato HSI, un canal de Tono (H), Saturación (S) o Valor (V) cuando la imagen se encuentra en formato HSV, un canal A, B o C cuando la imagen se encuentra en un formato desconocido o un canal NO DEFINIDO (ND), cuando el objeto se pretende utilizar para propósitos generales como por ejemplo una máscara, o resultado de alguna operación.

5.2.5.2. Funciones miembros de la clase.

5.2.5.2.1. CCanal::CCanal

CCanal():
CCanal(CSize Dimensiones, TIPO_CANAL Tipo=CANAL_ND);
Throw(CmemoryException);
CCanal(CSize Dimensiones, Decimal val, TIPO_CANAL Tipo=CANAL_ND);
Throw(CmemoryException);
CCanal(const CCanal &canal);
Throw(CmemoryException);

Parámetros

Dimensiones

Dimensiones de la matriz que representará el canal.

Tipo

Tipo de canal que será representado por el objeto.

Val

Valor que se coloca en todas las celdas de la matriz.

CCanal

Canal que se desea copiar al nuevo objeto

Observaciones

El constructor por defecto Coloca las dimensiones del canal a 0, el miembro m_pDC a NULL y el tipo de canal a CANAL_ND o no definido. Cuando no se especifica el valor a colocar en las celdas de la matriz, estas se inicializan en cero.

Una excepción de tipo CMemoryException es lanzada cuando no hay suficiente memoria para crear la matriz dinámica que define el canal.

5.2.5.2.2. CCanal::Crear

BOOL Crear(CSize Dimensiones=CSize(0,0), TIPO_CANAL Tipo=CANAL_ND);
Throw(CMemoryException);

Retorno

True si el canal se crea satisfactoriamente, False en cualquier otro caso.

Parámetros

Dimensiones

Dimensiones de la matriz que representará el canal.

Tipo:

Tipo de canal que será representado por el objeto.

Observaciones

Una excepción de tipo CMemoryException es lanzada cuando no hay suficiente memoria para crear la matriz dinámica que define el canal.

5.2.5.2.3. CCanal::Destruir

BOOL Destruir();

Retorno

True en caso de destruirse el objeto satisfactoriamente, False en cualquier otro caso.

Observaciones

Esta función tiene como objetivo liberar la memoria reservada por el objeto y colocar todos sus miembros a los valores por defecto.

5.2.5.2.4. CCanal::GetDimensiones

CSize GetDimensiones();

Retorno

Las dimensiones del canal.

5.2.5.2.5. CCanal::GetTipo

TIPO_CANAL GetTipo()

Retorno

El tipo del canal.

5.2.5.2.6. CCanal::SetTipo

void SetTipo(TIPO_CANAL tipo)

Parámetros

Tipo

Tipo de canal el cual se desea establecer en el objeto referenciado.

5.2.5.2.7. CCanal::GetVal

Decimal GetVal(const Entero &x,const Entero &y)

Retorno

El valor de la matriz almacenado la posición especificada o un valor ***NaN*** en caso de no poder hacer referencia a esa posición.

Parámetros

x, y

La posición en X y Y de elemento que se quiere obtener de la matriz.

Observaciones

Obtiene el valor de la matriz en una posición dada. Los parámetros se pasan a la función como una referencia con el fin de obtener un mejor desempeño al no tener que crear copias internas de los objetos como sucede cuando se pasan parámetros por valor.

Con el fin de brindar un mejor desempeño, no se realiza ningún tipo de verificación dentro de esta función, por lo tanto queda como responsabilidad del programador las verificaciones pertinentes.

5.2.5.2.8. CCanal::SetVal

void SetVal(const Entero &x,const Entero &y,Decimal Num)

Parámetros

x, y

Posición de la matriz en la cual se desea establecer un valor.

Num

Valor de que desea establecer en la matriz.

Observaciones

Establece el valor de la matriz en una posición dada. Con el fin de brindar un mejor desempeño, no se realiza ningún tipo de verificación dentro de esta función, por lo tanto queda como responsabilidad del programador las verificaciones pertinentes. Los parámetros se pasan a la función como una referencia con el fin de obtener un

mejor desempeño al no tener que crear copias internas de los objetos como sucede cuando se pasan parámetros por valor.

5.2.5.2.9. CCanal::IsCanal

BOOL IsCanal()

Retorno

True en caso de ser un canal con información, False en cualquier otro caso.

Observaciones

Se define una canal con información a aquel canal cuyo miembro m_pDC tiene memoria reservada.

5.2.5.2.10. CCanal::IsDimenciones

BOOL IsDimenciones(const Entero &x, const Entero &y);

Retorno

True en caso de que las coordenadas dadas como parámetros se encuantren dentro de las dimensiones del canal, False en cualquier otro caso.

Parámetros

x, y

Coordenadas que se desean evaluar.

Observaciones

Los parámetros son pasados por referencia para obtener un mejor desempeño en la ejecución de la función.

Esta función es muy útil al realizar verificaciones en el momento en que se quiere establecer un valor dentro de la matriz u obtener un valor de ella debido a que las funciones que realizan estas tareas no hacen dicha verificación internamente.

5.2.5.2.11. CCanal::SubSeccion

CCanal SubSeccion(Entero x1, Entero y1, Entero x2, Entero y2, SUB_SECCION flag=SS_PUNTUAL);
AfxThrowUserException( );
Thorw(CmemoryException);

Retorno

Un canal que contiene un subsección del canal original.

Parámetros

x1, y1, x2, y2

Coordenadas que definen la subsección que se desea extraer del canal.

flag

Parámetro que define la forma en que será interpretados los parámetros x1, y1, x2, y2, en cualquier caso, x1, y1 son las coordenadas de la esquina superior izquierda de la subsección que se desea extraer y los parámetros x2, y2 se interpretan dependiendo del valor que tome el parámetro flag así:

SS_PUNTUAL: Coordenada inferior derecha de la subsección que se desea extraer del canal.

SS_LONGITUDINAL: Ancho de la subsección que se desea extraer del canal.

Observaciones

Extrae una subsección de una canal. La función arroja una excepción de memoria cuando no hay memoria suficiente para crear la subsección y excepciones de usuario cuando las coordenadas que definen la subsección no son válidas.

5.2.5.2.12. CCanal::GetMayor

Decimal GetMayor();

Retorno

El mayor valor dentro de la matriz.

5.2.5.2.13. CCanal::GetMenor

Decimal GetMenor();

Retorno

El menor valor dentro de la matriz.

5.2.5.2.14. CCanal::Redondear

void Redondear();

Observaciones

Esta función redondea todos los valores dentro del canal, es bastante útil al momento de querer utilizar el canal para desplegar una imagen en pantalla luego de que esta es manipulada por un filtro, el cual puede dejar valores no enteros dentro del canal lo cual lo dejaría no apto para su visualización.

5.2.5.2.15. CCanal::Invertir

void Invertir();

Observaciones

Esta función realiza una inversión RGB de los valores dentro del canal, es decir, el valor de cada posición dentro e la matriz será 255-val luego de la ejecución de esta función

5.2.5.2.16. CCanal::CrearBuffer

unsigned char *CrearBuffer(BOOL FalsoColor=FALSE);.

Thorw(CmemoryException);
AfxThrowUserException( );

Retorno

Puntero a un buffer en memoria que contiene una imagen independiente del dispositivo en formato BMP creada a partir de la información contenida en la matriz del canal.

Parámetros

FalsoColor

Parámetro que define si la imagen que contendrá el buffer será una imagen a escala de grises o una imagen en falso color.

Observaciones

En cualquier caso, la imagen que se crea dentro del buffer corresponde a una imagen RGB, es decir de tres canales. Cuando se crea una imagen a escala de grises los tres canales son idénticos entre si, cuando se crea una imagen en falso color a cada valor distinto de la matriz del objeto le es asignado un color RGB aleatorio el cual será colocado dentro del buffer.

La función arroja una excepción de usuario en caso de no existir información dentro del objeto para crear el buffer o en caso de que sus valores no sean aptos para la creación de una imagen RGB, caso que se da por ejemplo cuando la matriz contiene valores negativos o mayores a 255.

También arroja una excepción de tipo CmemoryException en caso de no haber memoria suficiente para la creación del buffer.

5.2.5.2.17. CCanal::CrearBitmap

CBitmap *CCanal::CrearBitmap(CDC *pDC,BOOL FalsoColor);

Retorno

Objeto CBitmap (Mapa de bits dependiente del dispositivo) creado a partir de la información dentro del canal.

Parámetros

FalsoColor

Parámetro que define si el BitMap será a escala de grises o en falso color.

Observaciones

La función crea una mapa de bits dependiente del dispositivo a partir de la información que contiene la matriz del canal. Debido a que disponemos de solo una matriz, la imagen que se genera dentro del bitmap puede ser solo a escala de grises o en falso color.

Esta función usa internamente a la función **CrearBuffer**, razón por la cual arroja las mismas excepciones.

5.2.5.2.18. CCanal::Serialize

virtual void Serialize(CArchive &ar);

Observaciones

Esta es una función virtual sobrescrita de la clase base de CCanal, la clase CObject, tiene como objetivo seriar el objeto.

5.2.5.2.19. CCanal:: operator=

CCanal &operator=(const CCanal &canal);
throw(CMemoryException);
CCanal &operator=(const Decimal &num);

Retorno

Un canal que es una copia de otro o un canal con todos los elementos de su matriz establecidos a un valor determinado.

Parámetros

canal:

Referencia a un objeto CCanal que se desea copiar.

num:

Valor que se desea copiar en todos las elementos de la matriz.

5.2.5.3. Funciones amigas de la clase

5.2.5.3.1. operator+

friend CCanal operator+(CCanal &CanalA, CCanal &CanalB);

AfxThrowUserException( );

friend CCanal operator+(CCanal &CanalA, Decimal Num);

Retorno

Un canal que es el resultado de la suma miembro a miembro de los elementos de las matrices de dos canales distintos o un canal que es el resultado de sumarle a todos los elementos de la matriz un valor dado.

Parámetros

CanalA:

Primer operador de la suma.

CanalB:

Canal que será el segundo operador de la suma de dos canales miembro a miembro.

Num:

Valor que será el segundo operador de la suma entre un canal y un valor dado.

5.2.6. CImagen

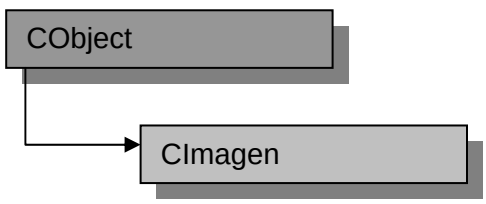


Figura 59. *Diagrama jerárquico de la clase Imagen.*

La clase CImagen define una estructura de datos para trabajar con imágenes cuyo formato de representación puede ser definido por tres canales de información y provee funciones miembro para acceder y transformar los datos almacenados en estos canales.

Para usar un objeto de la clase CImagen, constrúyalo, y entonces llame a sus funciones miembros.

- PROPIEDADES
    - m_Canal_A;
    - m_Canal_B;
    - m_Canal_C;
    - m_Dimensiones;
    - m_Tipo;
- MÉTODOS
    - CImagen
    - Crear
    - Destruir
    - CargarImagen
    - SetVals
    - SetVal_A
    - SetVal_B
    - SetVal_C
    - GetVals
    - GetVal_A
    - GetVal_B
    - GetVal_C
    - IsImagen
    - IsDimensiones
    - GetDimensiones

- SetCanales
- SetCanal
- SetTipo
- GetTipo
- GetCanal_A
- GetCanal_B
- GetCanal_C
- Subsección
- CrearBuffer
- CrearBitmap
- VisualizarImg

```
# include <Imagen.h>
```

5.2.6.1. Datos miembros de la clase

5.2.6.1.1. CImagen::m_Canal_A, m_Canal_B, m_Canal_C;

```
CCanal                  m_Canal_A;
CCanal                  m_Canal_B;
CCanal                  m_Canal_C;
```

Observaciones

Estos tres datos miembros de la clase representan los tres canales que definen la imagen que será almacenada por el objeto CImagen, dependiendo del formato de representación que se disponga para la imagen estos canales pueden representar en su orden los canales Rojo, Verde y Azul si la imagen se encuentra en formato RGB, Tono, Saturación e Intensidad si la imagen se encuentra en formato HSI o el Tono, Saturación y Valor si la imagen se encuentra en formato HSV.

5.2.6.1.2. CImagen::m_Dimenciones

```
CSize                   m_Dimenciones;
```

Observaciones

Este dato miembro contiene las "*dimensiones de la imagen"*, es un dato de vital importancia al momento de querer liberar la memoria utilizada por los objetos instanciados de esta clase y al querer acceder a la información de la imagen sin riesgo de violar las fronteras de los segmentos de memoria reservados para dicho propósito.

El objeto CSize pertenece a las MFC, su funcionamiento es básico y representa un tamaño en dos dimensiones, este objeto tiene dos datos miembros públicos, ***cx*** y ***cy***, para nuestra clase, el primero representa las dimensiones de la imagen a lo largo del eje X y el segundo representa las dimensiones de la imagen a lo largo del eje Y.

5.2.6.1.3. CImagen::m_Tipo

TIPO_IMAGEN m_Tipo;

Observaciones

Este dato miembro representa el formato en que la imagen se encuentra representada y puede tomar uno de los siguientes valores:

IMAGEN_RGB: Imagen en formato RGB.

IMAGEN_HSV: Imagen en formato HSV.

IMAGEN_HSI: Imagen en formato HSI.

IMAGEN_ND: Imagen en formato no definido.

5.2.6.2. Funciones miembros de la clase

5.2.6.2.1. CImagen::CImagen

CImagen();

CImagen(CCanal &c1, CCanal &c2, CCanal &c3);

CImagen(CSize Dimensiones);

CImagen(const CImagen &imagen);
CImagen(CFile *file);
CImagen(CString ruta);
throw(CImagenException);

Parámetros

c1, c2, c3:

Estos tres parámetros definen los tres canales que se desean establecer como canales de la imagen.

Dimensiones:

Dimensiones de la imagen que se desea crear.

imagen:

Objeto CImagen que se desea copiar la objeto que se va a crear.

file:

Objeto CFile que contienen un fichero cargado desde el disco.

ruta:

Ruta en disco o red de la imagen que se desea cargar en el objeto CImagen.

Observaciones.

El constructor por omisión crea un objeto CImagen de tipo no definido y sin dimensiones. Cuando las dimensiones son especificadas se reserva memoria para almacenar información en los tres canales de la imagen.

Cuando un objeto CFile es especificado, el contenido de este es pasado a un buffer en memoria donde la información es interpretada y almacenada en los tres canales al tiempo que se establecen las dimensiones y el formato de representación de la imagen.

Si una ruta de archivo en disco o red es especificada, un objeto CFile es creado a partir de esta y el procedimiento de carga es el descrito anteriormente.

5.2.6.2.2. CImagen::Crear

void Crear(CSize Dimensiones, TIPO_IMAGEN tipo);

Parámetros

Dimensiones:

Dimensiones de la imagen.

tipo:

Formato de representación de la imagen.

Observaciones

Esta funcione crea un objeto CImagen con las dimensiones y el formato de representación especificados de tal manera que la información anteriormente almacenada en el objeto es eliminada liberando la memoria anteriormente asignada y reservando nueva memoria para los tres canales de la imagen.

5.2.6.2.3. CImaen::Destruir

void Destruir();

Observaciones

Esta función libera la memoria asignada a los canales de un objeto CImagen y coloca las dimensiones de este y el tipo de representación a 0,0 y no definido respectivamente.

5.2.6.2.4. CImagen::CargarImagen

void CargarImagen(CString ruta);
void CargarImagen(CFile *pFich);
throw(CImagenException);

Parámetros

pFich:

Puntero a un objeto CFile que contienen un fichero cargado desde el disco.

ruta:

Ruta en disco o red de la imagen que se desea cargar en el objeto CImagen.

Observaciones

Esta es una función de iniciación que tiene como objetivo principal llenar las estructuras de datos que hacen parte de un objeto CImagen con información de proveniente de un fichero en disco.

5.2.6.2.5. CImagen::SetVals

void SetVals(const Entero &x, const Entero &y, const Decimal &c1, const Decimal &c2, const Decimal &c3)

Parámetros

x, y:

Coordenadas de la posición en donde se desea establecer la tripleta de valores para los canales de la imagen.

c1, c2, c3:

Los tres valores que se establecerán en los canales a, b y c de la imagen.

Observaciones

La función no realiza ningún tipo de verificación de datos como por ejemplo que las coordenadas se encuentren dentro de las dimensiones de la imagen, este tipo de verificaciones se omite con el fin de dar un mejor desempeño a la ejecución de esta función.

5.2.6.2.6. CImagen::SetVal_A, SetVal_B, SetVal_C

void SetVal_A(const Entero &x, const Entero &y, const Decimal &c_A)
void SetVal_B(const Entero &x, const Entero &y, const Decimal &c_B)
void SetVal_C(const Entero &x, const Entero &y, const Decimal &c_C)

Parámetros

x, y:

Coordenadas de la imagen en donde se desea establecer el valor para un canal específico.

c_A, c_B, c_C:

Valor que se desea establecer en el canal especificado.

Observaciones

Estas tres funciones establecen valores en solo uno de los canales a, b o c respectivamente. Ninguna de ellas realiza una verificación de datos como por ejemplo que las coordenadas se encuentren dentro de las dimensiones de la imagen, este tipo de verificaciones se omite con el fin de dar un mejor desempeño a la ejecución de esta función.

5.2.6.2.7. CImagen::GetVals

void GetVals(const Entero &x, const Entero &y, Decimal &c1,Decimal &c2, Decimal &c3)

Parámetros

x, y:

Coordenadas de la imagen en donde de desea obtener los valores de los canales.

c1, c2, c3:

Parámetros pasador por referencia en donde se devolverán los valores solicitados.

Observaciones

La función no realiza ningún tipo de verificación de datos como por ejemplo que las coordenadas se encuentren dentro de las dimensiones de la imagen, este tipo de verificaciones se omite con el fin de dar un mejor desempeño a la ejecución de esta función.

5.2.6.2.8. CImagen::GetVal_A, GetVal_B, GetVal_C

void GetVal_A(const Entero &x, const Entero &y, Decimal &c_A)

void GetVal_B(const Entero &x, const Entero &y, Decimal &c_B)

void GetVal_C(const Entero &x, const Entero &y, Decimal &c_C)

Parámetros

x, y:

Coordenadas de la imagen en donde se desea obtener el valor de canal específico.

c_A, c_B, c_C:

Parámetro pasado por referencia en donde se devolverá el valor solicitado.

Observaciones

Estas tres funciones obtienen valores en solo uno de los canales a, b o c respectivamente. Ninguna de ellas realiza una verificación de datos como por ejemplo que las coordenadas se encuentren dentro de las dimensiones de la imagen, este tipo de verificaciones se omite con el fin de dar un mejor desempeño a la ejecución de esta función.

5.2.6.2.9. CIamgen::IsImagen

BOOL IsImagen()

Retorno

True en caso de ser una imagen con información, False en cualquier otro caso.

Observaciones

Se define una imagen con información a aquella imagen cuyos canales posean memoria reservada para el almacenamiento de información.

5.2.6.2.10. CImagen::IsDimensiones

BOOL IsDimensiones(const Entero &x, const Entero &y);

Retorno

True en caso de que las coordenadas dadas como parámetros se encuentren dentro de las dimensiones de la imagen, False en cualquier otro caso.

Parámetros

x, y

Coordenadas que se desean evaluar.

Observaciones

Los parámetros son pasados por referencia para obtener un mejor desempeño en la ejecución de la función.

Esta función es muy útil al realizar verificaciones en el momento en que se quiere establecer un valor dentro de los canales de la imagen u obtener un valor de ella debido a que las funciones que realizan estas tareas no hacen dicha verificación internamente.

5.2.6.2.11. CImagen::GetDimensiones

CSize GetDimensiones();

Retorno

Las dimensiones del canal en un objeto CSize.

5.2.6.2.12. CImagen::SetCanales

void SetCanales(CCanal &c1, CCanal &c2, CCanal &c3);
throw(CImagenException);

Parámetros

c1, c2, c3:

Referencias a tres objetos CCanal que se desean establecer como canales del objeto.

Observaciones

Esta función establece los canales de una imagen a los canales que se le pasa como parámetros, internamente se verifica que los canales tengan información válida y las mismas dimensiones.

Dependiendo del tipo de canales que se estén estableciendo (R, G, B, H, S, I, V o no definido) le es asignada a la imagen un formato.

5.2.6.2.13. CImagen::SetCanal

void SetCanal(CCanal &canal, Entero NumCanal);

Parámetros

Canal:

Referencia al canal que se desea establecer en el objeto.

NumCanal:

Numero de canal en el cual se desea colocar el canal que se pasa como parámetro, si es 1 (uno) se establece el canal A, si es 2 (dos) se establece el canal B y si es 3 (tres) se establece el canal C. Si el valor de NumCanal no esta entre los definidos ningún canal es establecido.

Observaciones

Luego de establecer el canal, los tres canales de la imagen son evaluados para reasignar un formato a la imagen.

5.2.6.2.14. CImagen::SetTipo

void SetTipo(TIPO_CANAL a, TIPO_CANAL b, TIPO_CANAL c);
void SetTipo(TIPO_IMAGEN tipo);

Parámetros

a, b, c:

Estos parámetros corresponden a los tres tipos de canales distintos.

tipo:
Tipo de formato que se desea establecer para la imagen.

Observaciones
Esta función establece el valor del dato miembro m_Tipo. El primer prototipo de función evalúa tres tipos de canales distintos y si estos definen un formato de imagen dicho formato es establecido como el formato de representación de la imagen.

Es importante destacar que estas funciones solo establecen el valor del dato miembro m_Tipo pero en ningún momento hace manipulación de los datos de los canales con fines de cambiar el verdadero formato de representación.

5.2.6.2.15. CImagen::GetTipo

TIPO_IMAGEN GetTipo();

Retorno:
Esta función retorna el valor del miembro m_Tipo, es decir, el formato de representación de la imagen.

5.2.6.2.16. CImagen::GetCanal_A, GetCanal_B, GetCanal_C

CCanal GetCanal_A();
CCanal GetCanal_B();
CCanal GetCanal_C();

Retorno
Estas tres funciones retornan respectivamente una copia de los canales A, B y C del objeto CImagen.

5.2.6.2.17. CImagen::SubSeccion

CImagen SubSeccion(Entero x1, Entero y1, Entero x2, Entero y2, SUB_SECCION flag=SS_PUNTUAL);

Retorno

Un objeto CImagen que contienen un subsección de la imagen contenida en el objeto CImagen que invoca la función.

Parámetros

x1, y1, x2, y2

Coordenadas que definen la subsección que se desea extraer de la imagen.

flag

Parámetro que define la forma en que será interpretados los parámetros x1, y1, x2, y2, en cualquier caso, x1, y1 son las coordenadas de la esquina superior izquierda de la subsección que se desea extraer y los parámetros x2, y2 se interpretan dependiendo del valor que tome el parámetro flag así:

SS_PUNTUAL: Coordenada inferior derecha de la subsección que se desea extraer de la imagen.

SS_LONGITUDINAL: Ancho de la subsección que se desea extraer de la imagen.

Observaciones

La función arroja una excepción de memoria cuando no hay memoria suficiente para crear la subsección y excepciones de usuario cuando las coordenadas que definen la subsección no son válidas.

5.2.6.2.18. CImagen::CrearBuffer

unsigned char *CrearBuffer();.

throw(CImagenException);

Retorno

Puntero a un buffer en memoria que contiene una imagen independiente del dispositivo en formato BMP creada a partir de la información contenida en los canales del objeto CImagen.

Observaciones

En cualquier caso, la imagen que se crea dentro del buffer corresponde a una imagen RGB, es decir de tres canales.

La función arroja una excepción en caso de no existir información dentro del objeto para crear el buffer o en caso de que sus valores no sean aptos para la creación de una imagen RGB, caso que se da por ejemplo cuando la matriz contiene valores negativos o mayores a 255.

También arroja una excepción en caso de no haber memoria suficiente para la creación del buffer.

5.2.6.2.19. CImagen::CrearBitmap

CBitmap *CrearBitmap(CDC *pDC);

Retorno

Objeto CBitmap (Mapa de bits dependiente del dispositivo) creado a partir de la información dentro de los tres canales del objeto CImagen.

Observaciones

La función crea una mapa de bits dependiente del dispositivo a partir de la información que contiene la matriz del canal.

Esta función usa internamente a la función **CrearBuffer**, razón por la cual arroja las mismas excepciones.

5.2.6.2.20. CImagen::VisualizarImg

void VisualizarImg(CDC *pDC, CPoint origen, CSize Tam, DWORD modo=SRCCOPY);
void VisualizarImg(CDC *pDC, CPoint origen=CPoint(0,0));
throw(CImagenException);

Parámetros

pDC:

Puntero al dispositivo de contesto donde se quiere visualizar la imagen.

origen:

Coordenadas del origen donde se desplegara la imagen.

tam:

Tamaño que tendrá la imagen en pixeles al ser desplegada en la pantalla

modo:

Especifica la operación de copia a ser ejecutada. Los códigos de operación de copia definen como el GDI combina colores en operaciones de salida que involucran una brocha especificada, un posible bitmap de origen, y un bitmap de destino. La siguiente es una lista de códigos de operaciones de copia y us respectivas descripciones:

BLACKNESS Convierte toda la salida a negro.

DSTINVERT Invierte el bitmap de destino.

MERGECOPY Combina el patrón y el bitmap de destino usando el operador lógico AND.

MERGEPAINT Combina el bitmap de origen invertido con el bitmap de destino usando la operación lógica OR.

NOTSRCCOPY Copia del inverso del bitmap de origen en el bitmap de destino.

NOTSRCERASE Invierte el resultado de combinar el bitmap de destino y origen usando el operador lógico OR.

PATCOPY Copia el patrón al bitmap de destino.

PATINVERT Combina el bitmap de destino con el patrón usando la operación lógica XOR.

PATPAINT Combina el inverso del bitmap de origen con el patron usando la operación lógica OR. Combina el resultado de esta operación con el bitmap de destino usando el operador lógico OR.

SRCAND Combina los pixeles del bitmap de destino y origen usando la operación lógica AND.

SRCCOPY Copia el bitmap de origen en el bitmap de destino.

SRCERASE Invierte el bitmap de destino y combina el resultado con el bitmap de origen Usando la operación lógica AND.

SRCINVERT Combina los pixeles del bitmap de destino y origen Usando la operación lógica XOR.

SRCPAINT Combina los pixeles del bitmap de origen y destino usando la operación lógica OR.

WHITENESS Convierte toda la salida a blanco.

Observaciones

El primer prototipo de la función es mas flexible porque permite aplicar distintos modos de copiado sobre el dispositivo de salida y además especificar el tamaño que tendrá la imagen al visualizarce, pero estas opciones extras hacen a esta función lenta y menos eficiente en ejecución.

El segundo prototipo de la función realiza un volcado rápido de la imagen sobre el dispositivo de visualización con las mismas dimensiones que posee la imagen y con modo de copiado ***SRCCOPY***

5.2.6.2.21. CImagen::Serialize

void Serialize(CArchive &ar);

Parámetros

ar:

Referencia al objeto CArchive donde va a seriar la el objeto CImagen;

Observaciones

Esta función tiene como objeto guardar en disco el objeto CImagen (SERIAR), internamente primero guarda las dimensiones y el formato e la imagen, por último llama a las funciones de seriación de los tres canales los cuales se guardan a si mismos.

5.2.7. CTrazo

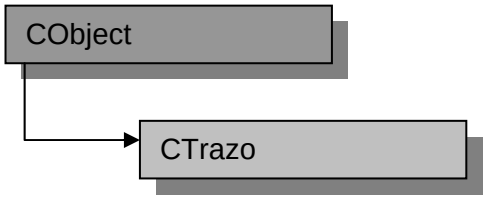


Figura 60. *Diagrama jerárquico de la clase Imagen.*

La clase CTrazo es una sencilla clase que encapsula una estructura de datos que se comporta como un vector dinámico. Su dato miembro principal es un objeto instaciado de una clase definida por la plantilla CArray. Además de las funcionalidades aportadas por esta plantilla, la clase CTrazo dispone de otras funciones miembro útiles en la creación de diferentes tipos de trazos como por ejemplo curvas spline y bezier.

Para utilizar la clase CTrazo, cree una instancia de esta clase y use sus funciones miembros y las de su dato miembro para definir el comportamiento de la estructura.

- PROPIEDADES
  - m_Valores
- MÉTODOS
  - CTrazo
  - LiberarTrazo
  - Graficar
  - Ordenar
  - EliminarRepetidos

- BuscarValor

include <Trazo.h>

5.2.7.1. Datos miembro de la clase

5.2.7.1.1. CTrazo::m_Valores

CArray<Punto, Punto> m_Valores;

Observaciones

Este dato miembro define la estructura dinámica que contendrá las valores del vector, la plantilla establece los miembros de array como datos de tipo punto. Una mejor descripción de la clase CArray se realiza en un siguiente anterior apartado de este mismo capítulo.

5.2.7.2. Funciones miembro de la clase

5.2.7.2.1. CTrazo::CTrazo

CTrazo();

Observaciones

Esta función es el constructor de la clase

5.2.7.2.2. Ctrazo::LiberarTrazo

void LiberarTrazo();

Observaciones

Esta función tiene como objetivo liberar la memoria utilizada por el miembro m_Valores del objeto.

5.2.7.2.3. CTrazo::Graficar

void Graficar(CDC *pDC, COLORREF color, int Modo=0,int Ancho=1,int Seleccionado=-1);

Parámetros

pDC:

Dispositivo cliente de visualización en donde se desea graficar el trazo.

color:

Color en el cual se desea graficar el trazo.

Modo:

Valor que define la forma en que se graficarán los valores de trazo, cuando modo vale 0 (cero), el trazo se dibuja como líneas continuas de un punto a otro, cuando vale 1 (uno), el trazo se dibuja como pequeños círculos en cada punto del trazo no unidos por líneas.

Ancho:

Ancho de la línea con la cual se dibujará el trazo.

Seleccionado:

Punto dentro del trazo que será dibujado de un color diferente de los demás para que resalte sobre los otros, este parámetro solo se tiene en cuenta cuando modo vale 1.

Observaciones

La función tiene como objetivo graficar los puntos dentro del trazo en forma de líneas continuas o puntos independientes.

5.2.7.2.4. Ctrazo::Ordenar

void Ordenar(int Modo=0);

Parámetros

Modo:

Parámetro que define el tipo de ordenamiento que se aplicara a los puntos del trazo. Como el trazo esta compuesto por una serie de puntos, cada uno con coordenadas

X y Y, el trazo puede ser ordenado por cualquiera de estos valores o en combinación de ambos dependiendo del valor que tome el parámetro así:

0: Ascendentemente por la X
1: Descendentemente por la X
2: Ascendentemente por la Y
3: Descendentemente por la Y
4: Ascendentemente por la X y por la Y
5: Descendentemente por la X y por la Y
4: Ascendentemente por la X y Descendentemente por la Y

5.2.7.2.5. CTrazo::EliminarRepetidos

void EliminarRepetidos(int modo=0);

Parámetros

Modo:

Esta función elimina los nodos repetidos por la X cuando modo vale 0 o por la y cuando nodo vale 1.

Observaciones

La función siempre deja el nodo cuyo valor del miembro contrario X o Y es mayor.

5.2.7.2.6. CTrazo::BuscarValor

int BuscarValor(Punto PBuscado, double H, Punto &PEncontrado)
int BuscarValor(CPoint PBuscado,double H,Punto &PEncontrado)

Retorno

La posición dentro del vector en la cual se encuentra en punto buscado, -1 en caso de no encontrarlo.

Parámetros

PBuscado:

Punto que se desea buscar dentro del trazo.

H:

Radio umbral dentro del cual se puede evaluar a un punto como encontrado. Para que un punto sea encontrado no es necesario que este exactamente la misma pareja de valores X y Y dentro del trazo, basta con que este dentro de la circunferencia demarcada con centro en PBuscado y de radio H.

PEncontrado:

Punto encontrado, se retorna este valor debido a que puede no ser encontrado exactamente el mismo punto dentro del trazo pero si uno que este dentro de la circunferencia descrita anteriormente.

Observaciones

Para saber si un punto se encuentra dentro de la circunferencia descrita por el centro PBuscado y radio H, se evalúa la distancia cartesiana entre el punto buscado y cada uno de los puntos del trazo, si esta distancia es menor o igual a H el punto se califica como encontrado, de existir varios puntos que estén dentro de la circunferencia se retorna el punto cuya distancia al punto buscado sea menor.

5.2.7.2.7. CTrazo::Serialize

void Serialize(CArchive &ar)

Parámetros

ar:

Referencia al objeto CArchive donde va a seriar la el objeto CTrazo;

Observaciones

Esta función tiene como objeto guardar en disco el objeto CTrazo (SERIAR).

5.2.8. CSpline

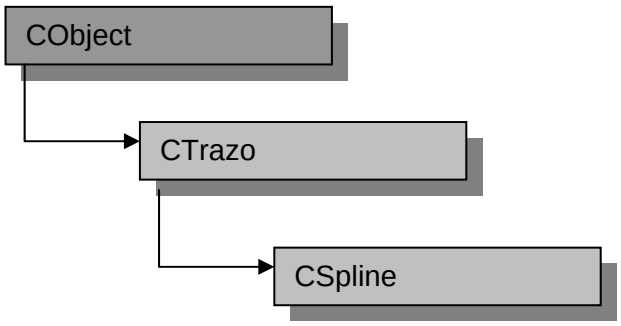


Figura 61. *Diagrama jerárquico de la clase CSpline.*

La clase CSpline es una abstracción genérica de la representación de una curva de trazador cúbico. Un trazador cúbico esta definido generalmente por sus nodos y por cuatro coeficientes por cada nodo excepto para el último, los cuales definen una ecuación cúbica que pasa por dos nodos consecutivos.

El miembro m_Valores de la clase base representa la curva propiamente dicha y es al final lo que se debe calcular y graficar una vez obtenidos los coeficientes.

Para más información acerca de trazadores cúbicos, refiérase al capítulo dedicado al marco teórico de este libro.

- PROPIEDADES
  - m_A
  - m_B
  - m_C
  - m_D
  - m_Nodos
- MÉTODOS
  - CSpline
  - LiberarSpline
  - TrazoANodos
  - SetNodos
  - CalcularSpline

include <Spline.h>

5.2.8.1. Datos miembros de la clase

5.2.8.1.1. CSpline::m_A, m_B, m_C, m_D

CArray<double,double> m_A, m_B, m_C, m_D;

Estos cuatro miembros de la clase son cuatro vectores dinámicos que se utilizan para guardar los coeficientes de cada ecuación cúbica.

5.2.8.1.2. CSpline::m_Nodos

CTrazo m_Nodos;

Este miembro define los nodos del trazador cúbico, estos nodos son la base para calcular los coeficientes que darán origen a las ecuaciones cúbicas con las cuales se calculará la curva final.

m_Nodos es declarado de tipo CTrazo y no CArray debido a que para la manipulación del Spline en interfaces gráficas se suele necesitar de procedimientos de ordenamiento y búsqueda, los cuales dispone la implementación de esta clase.

5.2.8.2. Funciones miembros de la clase

5.2.8.2.1. CSpline::LiberarSpline

void LiberarSpline();

Observaciones

LiberarSpline tiene como objetivo liberar todos los vectores dinámicos utilizados por el Spline y la correspondiente memoria reservada por ellos.

5.2.8.2.2. CSpline::TrazoANodos

BOOL TrazoANodos(CTrazo *array);

Retorno

True en caso de poder realizarse una aproximación a la curva pasada como parámetro, False en cualquier otro caso.

Parámetros

array:

Un vector de puntos que definen una curva continua.

Observaciones

TrazoANodos tiene como objeto realizar un ajuste de curva a un trazador cúbico encontrando un conjunto de nodos que al ser utilizados para calcular el Spline, el resultado sea los mas parecido posible a la curva original. Esta función afecta directamente al miembro m_Nodos.

5.2.8.2.3. CSpline::SetNodos

BOOL SetNodos(CArray<Punto,Punto> *nodos)

Retorno

True en caso de poder establecer el miembro m_Nodos, False en cualquier otro caso.

Parámetros

Nodos:

Array de puntos que representan los nodos que se quieren colocar en el miembro m_Nodos de la clase.

5.2.8.2.4. CSpline::CalcularSpline

BOOL CalcularSpline(double Inicio, double Fin, double H);
BOOL CalcularSpline(double H=1);

Retorno

True en caso de poder calcular la curva Spline, False en cualquier otro caso.

Parámetros

Inicio:

Valor inicial en X desde el cual se empezara a culcular la curva Spline.

Fin:

Valor final hasta el cual se calculará la curva Spline.

H:

Intervalo entre valores de X para los cuales será calculada la curva Spline.

Observaciones

Cuando se utiliza el primer prototipo de la función, la curva Spline es calculada entre dos valores de x dados para todos los puntos que se encuentren a una distancia H entre sí, por ejemplo si Inicio es 5, Fin es 15 y H es 2, la curva Spline es calculada para los valores de x: 5, 7, 9, 11, 13 y 15.

Cuando se utiliza el segundo prototipo de la función, el valore de Inicio y fin son los valores mínimo y máximo en x de los puntos que se encuentran en el miembro m_Nodos.

5.2.8.2.5. CSpline::Serialize

void Serialize(CArchive &ar)

Parámetros

ar:

Referencia al objeto CArchive donde va a seriar la el objeto CSpline;

Observaciones

Esta función tiene como objeto guardar en disco el objeto CSpline (SERIAR). Para seriar un Spline, lo único que tengo que guardar son sus nodos, ya que todo lo demás puede ser calculado a partir de estos.

5.2.9. CSplineNatural

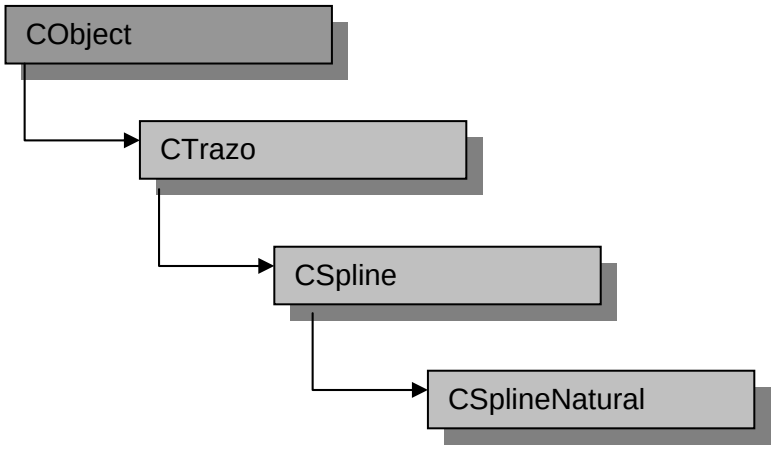


Figura 62. *Diagrama jerárquico de la clase CSplineNatural.*

Esta clase representa un trazador cúbico natural, para mas información acerca del trazador cúbico refiérase al la sección dedicada al marco teórico de este libro.

Para utilizar la clase CSplineNatural, instancie un objeto de esta clase y utilice sus funciones miembro o las de sus clases bases para calcularla curva Spline o manipularla en una interfaz gráfica.

- PROPIEDADES
  - Solo las derivadas
- MÉTODOS
  - CSplineNatural
  - TrazoANodos
  - CalcularCoeficientes

include <SplineNatural.h>

5.2.9.1. Funciones miembro de la clase

5.2.9.1.1. CSplineNatural::TrazoANodos

BOOL TrazoANodos(CTrazo *array);

Retorno

True en caso de poder realizarse una aproximación a la curva pasada como parámetro, False en cualquier otro caso.

Parámetros

array:

Un vector de puntos que definen una curva continua.

Observaciones

TrazoANodos tiene como objeto realizar un ajuste de curva a un trazador cúbico encontrando un conjunto de nodos que al ser utilizados para calcular el Spline, de manera que la curva resultante sea los mas parecida posible a la curva original. Esta función afecta directamente al miembro m_Nodos.

5.2.9.1.2. CSplineNatural::CalcularCoeficientes

BOOL CalcularCoeficientes();

Retorno

True en caso de poder calcular los coeficientes que definen las ecuaciones cuadráticas que se utilizan para calcular los valores interpolados de la curva, False en cualquier otro caso.

Observaciones

Esta función calcula los coeficientes que definen el Spline, estos coeficientes se almacenan en los miembros m_A, m_B, m_C y m_D heredados de la clase base. Siempre que se llame a esta función cualquier valor guardado en estos vectores es borrado.

5.2.10. CSplineSujeto

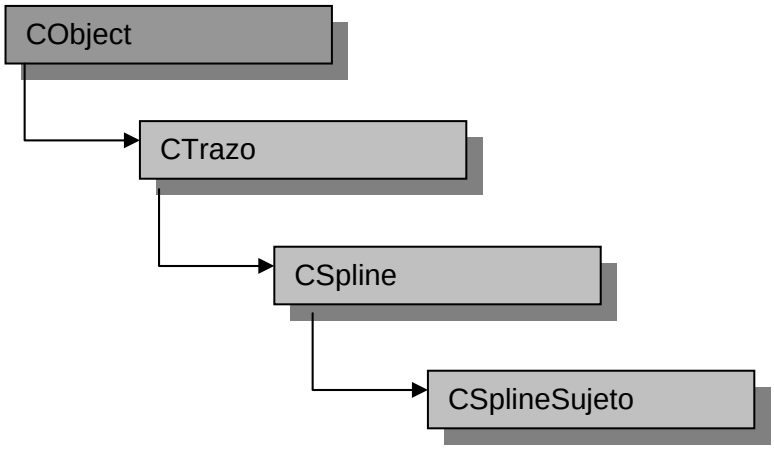


Figura 63. *Diagrama jerárquico de la clase CSplineSujeto.*

Esta clase representa un trazador cúbico sujeto, para mas información acerca del trazador cúbico refiérase al la sección dedicada al marco teórico de este libro.

Para utilizar la clase CSplineSujeto, instancie un objeto de esta clase y utilice sus funciones miembro o las de sus clases bases para calcularla curva Spline o manipularla en una interfaz gráfica.

- PROPIEDADES
  - m_FP0
  - m_FPN
- MÉTODOS
  - CSplineSujeto
  - TrazoANodos
  - SetFP0

- SetFPN
- GetFP0
- GetFPN

include <SplineSujeto.h>

5.2.10.1. Datos miembro de la clase

5.2.10.1.1. CSplineSujeto::m_FP0, m_FPN

double m_FP0, m_FPN;

Esos dos datos miembro representan la derivada en el primer y ultimo punto que debe tener la curva Spline interpolada a partir de los nodos que se especifiquen en el miembro m_Nodos. Un trazador natural se diferencia de uno sujeto precisamente en la existencia de estados dos parámetros extras que definen la curva Spline

5.2.10.2. Funciones miembro de la clase

5.2.10.2.1. CSplineSujeto:: SetFP0, SetFPN

void SetFP0(double val);
void SetFPN(double val);

Parámetros

FP0, FPN:

Estos dos parámetros representan los valores de las derivadas de los dos extremos del Spline que se desean establecer como parámetros de calculo para la curva que se pretende calcular.

5.2.10.2.2. CSplineSujeto:: GetFP0, GetFPN

double GetFP0();
double GetFPN();

Retorno

El valor de los datos miembro m_FP0 y m_FPN respectivamente.

5.2.11. CNodoBezier

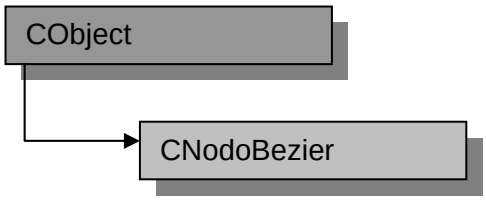


Figura 64. *Diagrama jerárquico de la clase CNodoBezier.*

La clase CNodoBezier está diseñada para representar una abstracción de datos conveniente de un nodo perteneciente a aun curva de tipo Bezier. Esta clase esta compuesta principalmente por cuatro miembros que definen el nodo, estos son: un nodo central, 2 puntos de control y un vector que contiene el segmento de curva calculada para el nodo.

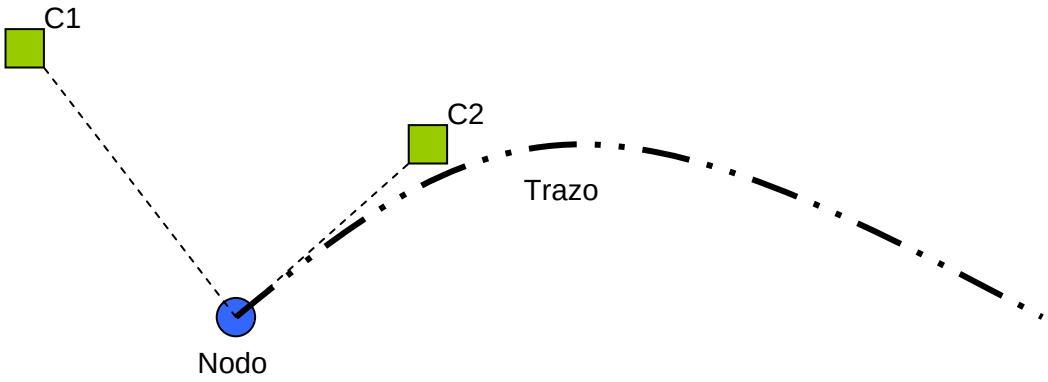


Figura 65. *Representación gráfica de la abstracción de datos de un nodo Bezier.*

Es importante destacar que el trazo Bezier esta definida para un Nodo Bezier por su propio nodo, el punto de control C2 y además otro nodo y otro punto de control externos al nodo en cuestión, generalmente el nodo y el punto de control C1 de otro Nodo Bezier.

Para información detallada acerca de la construcción y calculo de curvas paramétricas Bezier por favor refiérase al capítulo dedicado al marco teórico de este libro.

- PROPIEDADES
    - m_Trazo;
    - m_Nodo
    - m_C1
    - m_C2;
    - m_NodoSuavizado;
    - m_NumPasos
- MÉTODOS
    - CNodoBezier
    - LiberarValores
    - SetNodo
    - SetNodo
    - GetNodo
    - SetC1
    - GetC1
    - SetC2
    - GetC2
    - SetNumPasos
    - GetNumPasos
    - Mover
    - SetSuavizado
    - GetSuavizado
    - Graficar
    - GraficarNodo
    - GraficarAspas
    - GraficarCurva
    - BuscarAspa
    - CalcularBezier

include <NodoBezier.h>

5.2.11.1. Datos miembro de la clase

5.2.11.1.1. CNodoBezier::m_Trazo

CTrazo m_Trazo;

Observaciones

Este dato miembro tiene como objetivo almacenar la curva paramétrica calculada para el nodo Bezier en cuestion.

5.2.11.1.2. CNodoBezier::m_Nodo

Punto m_Nodo;

Observaciones

Este dato miembro representa contiene las coordenadas en X y Y del Nodo Bezier.

5.2.11.1.3. CNodoBezier::m_C1, m_C2

Punto m_C1, m_C2;

Observaciones

Estos dos datos miembro representan las coordenadas de los dos puntos de control pertenecientes al Nodo Bezier, las cuales servirán como parámetros para el calculo de la curva del propio Nodo y de otro nodo externo

5.2.11.1.4. CNododBezier::m_NodoSuavizado

BOOL m_NodoSuavizado;

Observaciones

Esta variable es útil al momento de la manipulación del Nodo Bezier a través de una interfaz gráfica. Cuando toma el valor TRUE, la funciones encargadas de manipular las coordenadas de los puntos de control del Nodo Bezier obligan a que siempre se conserve un ángulo de 180º entre dichos puntos de control, de manera que siempre que se mueva uno de los puntos de control, el otro también lo hará con el objetivo de cumplir con esta condición pero conservando siempre la misma distancia con respecto al nodo. Este tipo de manipulación se hace necesaria siempre que queramos evitar ***efectos de esquinas*** es un Nodo Bezier buscando en su lugar una ***curva suavizada.***

5.2.11.1.5. m_NumPasos::CNodoBezier

DWORD m_NumPasos;

Observaciones

Al cálculo de una curva paramétrica Bezier esta dada por dos ecuaciones, una para X y otra para Y, ambas en función de una variable continua ***T*** que toma valores entre 0 y 1. Para calcular la curva Bezier se hace necesario discretizar esta variable, dividiendo el intervalo que va de 0 a 1 en un número de segmentos dado por el miembro m_NumPasos.

Este dato miembro es útil en la manipulación mediante interfaz gráfica de un Nodo Bezier. Cuando se desea mover algún elemento del Nodo Bezier como por ejemplo su nodo o sus puntos de control y al mismo tiempo ver los resultados en pantalla en todo momento, es posible dar una valor bajo a esta variable con el fin de dar una previsualización de la curva mientras se modifica. La disminución del número de pasos se hace necesario debido a que una gran cantidad de Pasos para la variable ***T*** conlleva a una gran cantidad de cálculos que, dependiendo de la máquina, haría demasiado lento el proceso de visualización. Luego de terminar la modificación de la curva es posible dar un valor alto al número de pasos para calcular una curva mas detallada.

5.2.11.2. Funciones miembro de la clase

5.2.11.2.1. CNodoBezier::CNodoBezier

CNodoBezier();

CNodoBezier(Punto N, BOOL Suavizado=FALSE);

CNodoBezier(double Nx, double Ny, BOOL Suavizado=FALSE);

CNodoBezier(Punto N, Punto C2,BOOL Suavizado=FALSE);

CNodoBezier(double Nx, double Ny, double C2x,double C2y, BOOL Suavizado=FALSE);

CNodoBezier(Punto N, Punto C1, Punto C2, BOOL Suavizado=FALSE);

CNodoBezier(double Nx, double Ny, double C1x, double C1y, double C2x, double C2y, BOOL Suavizado=FALSE);

Parámetros

N:

Punto que define las coordenadas del miembro m_Nodo del objeto.

Nx, Ny:

Coordenadas que definen el miembro m_Nodo.

C1:

Punto que define las coordenadas del miembro m_C1.

C1x,C1y:

Coordenadas que definen el miembro m_C1.

C2:

Punto que define las coordenadas del miembro m_C2.

C2x,C2y:

Coordenadas que definen el miembro m_C2.

Suavizado:

Parámetro que define el valor de dato miembro m_NodoSuavizado.

Observaciones

En todos los constructores, el miembro m_NumPasos es establecido a 500. El valor por defecto de m_NodoSuavizado es False.

5.2.11.2.2. CNodoBezier:LiberarValores

void LiberarBezier();

Observaciones

Esta función libera la memoria utilizada por el miembro m_Trazo eliminando los valores guardados por él.

5.2.11.2.3. CNodoBezier::SetNodo

void SetNodo(Punto &P);
void SetNodo(double x, double y);

Parámetros

P:

Punto que define las coordenadas del miembro m_Nodo del objeto.

x, y:

Coordenadas que definen el miembro m_Nodo.

5.2.11.2.4. CNodoBezier::GetNodo

Punto GetNodo();

Retorno

Esta función retorna un punto que contiene las coordenadas de la posición del miembro m_Nodo.

5.2.11.2.5. CNodoBezier::SetC1

void SetC1(Punto &P);
void SetC1(CPoint &P);
void SetC1(double x, double y);

Parámetros

P:

Punto u objeto CPoint que define las coordenadas del miembro m_C1 del objeto.

x, y:

Coordenadas que definen el miembro m_C1.

Observaciones

Esta función establece la posición del miembro m_C1, cuando m_NodoSuavizado es igual a TRUE, la función también modifica la posición del miembro m_C2 para que guarde un ángulo de 180º con el miembro m_C1 de la forma en que se explico en apartados anteriores.

5.2.11.2.6. CNodoBezier::GetC1

Punto GetC1();

Retorno

Un punto con las coordenadas de la posición del miembro m_C1 del objeto.

5.2.11.2.7. CNodoBezier::SetC2

void SetC2(Punto &P);

void SetC2(CPoint &P);

void SetC2(double x, double y);

Parámetros

P:

Punto u objeto CPoint que define las coordenadas del miembro m_C2 del objeto.

x, y:

Coordenadas que definen el miembro m_C2.

Observaciones

Esta función establece la posición del miembro m_C2, cuando m_NodoSuavizado es igual a TRUE, la función también modifica la posición del miembro m_C1 para que guarde un ángulo de 180º con el miembro m_C2 de la forma en que se explico en apartados anteriores.

5.2.11.2.8. CNodoBezier::GetC2

Punto GetC2();

Retorno

Un punto con las coordenadas de la posición del miembro m_C2 del objeto.

5.2.11.2.9. CNodoBezier::SetNumPasos

void SetNumPasos(int NumPasos);

Parámetros

NumPasos:

Valor que se establecerá para el miembro m_NumPasos del objeto.

5.2.11.2.10. CNodoBezier::GetNumPasos

int GetNumPasos();

Retorno

Esta función retorna el valor del miembro m_NumPasos.

5.2.11.2.11. CNodoBezier::Mover

void Mover(double x, double y);

void Mover(Punto &P);

void Mover(CPoint &P);

Parámetros

P:

Punto que define las coordenadas del miembro m_Nodo del objeto.

x, y:

Coordenadas que definen el miembro m_Nodo.

Observaciones

Esta función mueve junto con el miembro m_Nodo, los miembro m_C1 y m_C2 una distancia igual a la que se desplaza el nodo y en la misma dirección.

5.2.11.2.12. CNodoBezier::SetSuavizado

void SetSuavizado(BOOL Suavizar);

Parámetros

Suavizar:

Parámetro que define el valor que se establecerá en el miembro m_NodoSuavizado

5.2.11.2.13. CNodoBezier::GetSuavizado

BOOL GetSuavizado();

Retorno

Esta función retorna el valor del miembro m_NodoSuavizado.

5.2.11.2.14. CNodoBezier::GraficarNodo

void GraficarNodo(CDC *pDC, COLORREF Color);

Parámetros

pDC:

Puntero al dispositivo de contexto donde se desea graficar el miembro m_Nodo del objeto.

Color:

Color en que se desea graficar el nodo.

Observaciones

Esta función solo grafica el miembro m_Nodo de la clase y lo hace exactamente en las coordenadas que este miembro establece.

5.2.11.2.15. CNodoBezier::GraficarAspas

void GraficarAspas(CDC *pDC, COLORREF ColorPunto, COLORREF ColorAspa, int Aspa=0);

Parámetros

pDC:

Puntero al dispositivo de contexto donde se desea graficar los puntos de control definidos por los miembros m_C1 y m_C2.

ColorPunto:

Color en que se dibujarán los puntos de control.

ColorAspa:

Color en que se dibujaran las líneas que unen los puntos de control al nodo.

Aspa:

Parámetro que define el punto de control que se dibujará, si vale 0 se dibujan ambos, si vale uno se dibuja solo m_C1 y si vale 2 se dibuja solo m_C2.

5.2.11.2.16. CNodoBezier::BuscarAspa

int BuscarAspa(Punto PBuscado, double H, Punto &PEncontrado);

int BuscarAspa(CPoint PBuscado, double H, Punto &PEncontrado);

int BuscarAspa(double BuscadoX, double BuscadoY, double H, Punto &PEncontrado);

Retorno

La función retorna el número del punto de control que se encuentra mas cercano a las coordenadas establecidas por los demás parámetros, es decir, 1 si el mas cercano es el miembro m_C1, 2 si es el miembro m_C2 o 0 si ningunos de los dos puntos de control cumplen con las condiciones establecidas.

Parámetros

PBuscado:

Punto que contiene las coordenadas que se desean buscar en alguno de los dos puntos de control.

BuscadoX, BuscadoY:

Coordenadas que se desean buscar en alguno de los dos puntos de control.

H:

Radio umbral de búsqueda.

pEncontrado:

Referencia a un Punto u objeto CPoint que contiene del punto de control más cercano a las coordenadas requeridas.

5.2.11.2.17. CNodoBezier::CalcularBezier

BOOL CalcularBezier(Punto N, Punto C);

BOOL CalcularBezier(double Nx, double Ny, double Cx, double Cy);

Retorno

True en caso de poder calcular la curva paramétrica Bezier, False en cualquier otro caso.

Parámetros

N:

Punto que define las coordenadas del segundo nodo necesario para calcular curvas paramétricas Bezier.

C:

Punto que define las coordenadas del segundo punto de control necesario para calcular curvas paramétricas Bezier.

Nx, Ny:

Coordenadas del segundo nodo necesario para calcular curvas paramétricas Bezier.

Cx, Cy:

Coordenadas del segundo punto de control necesario para calcular curvas paramétricas Bezier.

Observaciones

Para calcular curvas paramétricas Bezier es necesario establecer cuatro parámetros, dos nodos y dos puntos de control. El primer nodo y el primer punto de control están dados por los miembros m_Nodo y m_C1 del objeto, estas funciones reciben información que le permiten establece los parámetros faltantes para calcular la curva.

5.2.12. CBezier

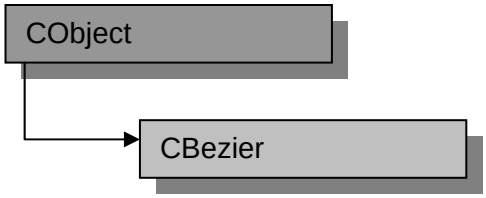


Figura 66. *Diagrama jerárquico de la clase CBezier.*

Esta clase representa la abstracción una de las abstracciones de datos que se puede hacer de una curva Bezier formada por secuencias de nodos Bezier.

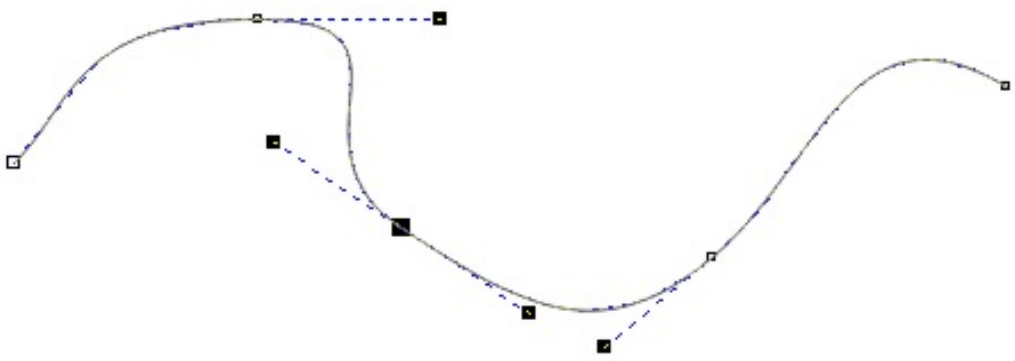


Figura 67. *Secuencia de nodos Bezier que definen una curva paramétrica.*

- PROPIEDADES
  - m_Nodos
- MÉTODOS
  - CBezier
  - LiberarBezier
  - BuscarNodo
  - BuscarAspa
  - GraficarCurvas
  - GraficarNodos
  - GraficarAspas
  - CalcularBezier
  - BuscarMasCercano
  - TrazoANodos

5.2.12.1. Datos miembro de la clase

5.2.12.1.1. CBezier::m_Nodos

CObArray m_Nodos;

Observaciones

Este dato miembro es un vector de punteros a objetos de derivados de la clase CObject. Específicamente en esta clase, cada puntero de este vector apunta a objetos de tipo CNodoBezier estableciendo de esta forma una secuencia de nodos con las cuales se puede calcular una curva paramétrica de N nodos.

5.2.12.1.2. CNodoBezier::BuscarNodo

int BuscarNodo(Punto Buscado,double H,Punto &Encontrado);

int BuscarNodo(CPoint Buscado,double H,Punto &Encontrado);
int BuscarNodo(double BuscadoX, double BuscadoY, double H, Punto &Encontrado);

Retorno

La posición dentro del array m_Nodos que ocupa el nodo que cumple con las condiciones de búsqueda establecidas por los parámetros pasados a la función, en caso de no encontrar ningún nodo que cumpla con estos, la función devuelve –1.

Parámetros

PBuscado:

Punto u objeto CPoint que contiene las coordenadas del nodo que se desea buscar.

BuscadoX, BuscadoY:

Coordenadas del nodo que se desea buscar.

H:

Valor que define el radio de búsqueda.

PEncontrado:

Punto que contiene las coordenadas el nodo encontrado que cumple con las condiciones de búsqueda.

5.2.12.1.3. CBezier::BuscarAspa

int BuscarAspa(int PosNodo, Punto Buscado, double H, Punto &Encontrado);
int BuscarAspa(int PosNodo, CPoint Buscado, double H, Punto &Encontrado);
int BuscarAspa(int PosNodo, double BuscadoX, double BuscadoY, double H,Punto &Encontrado);

Parámetros

PosNodo:

Posición del nodo donde se desea buscar un punto de control que cumpla con las condiciones especificadas por los demás parámetros dados.

Observaciones

Estas funciones utilizan internamente la función BuscarAspa de la clase CNodoBezier, luego los demás parámetros pasados a estas funciones se comportan de igual manera a los pasados a la función de CNodoBezier.

5.2.12.1.4. CBezier::GraficarNodos

void GraficarNodos(CDC *pDC, COLORREF Color, int Seleccionado=-1, COLORREF ColorSeleccionado=RGB(0,0,0));

Parámetros

pDC:

Puntero al dispositivo de contexto donde se visualizarán los nodos de la curva Bezier.

Color:

Color en que se dibujarán los nodos.

Seleccionado:

Posición de un nodo en especial, el cual se desea dibujar resaltado de los demás. Si el valor de este parámetro es –1, ningún nodo se dibujara como seleccionado.

ColorSeleccionado:

Color del nodo seleccionado.

Observaciones

Esta función dibuja la serie de nodos de la curva Bezier en las coordenadas especificadas por el miembro m_Nodo de cada uno de los objeto CNodoBezier que conforman el array de nodos m_Nodos perteneciente a la clase CBezier.

5.2.12.1.5. CBezier::GraficarAspas

BOOL GraficarAspas(CDC *pDC, COLORREF ColorPunto, COLORREF ColorAspa, int PosNodo=-1, int Seleccionado=0, COLORREF ColorPuntoSeleccionado=RGB(0,0,0), COLORREF ColorAspaSeleccionado=RGB(0,0,0));.

Parámetros

pDC:

Puntero al dispositivo de contexto donde se visualizarán los nodos de la curva Bezier.

ColorPunto:

Color con que se dibujaran los puntos de control de cada uno de los nodos de la curva Bezier.

ColorAspa:

Color con que se dibujarán cada una de las líneas (aspas) que unen los puntos de control al nodo de cada nodo Bezier,

PosNodo:

Posición de un nodo específico que contiene uno de los puntos de control seleccionado, cuando este parámetro vale –1, ningún punto de control es seleccionado.

Seleccionado:

Dado un nodo seleccionado, este parámetro indica el aspa que se desea dibujar como seleccionada seleccionar, sus valores pueden ser 0, 1 o 2 para ambos puntos de control, el punto de control 1 o el punto de control 2 respectivamente

5.3. DOCUMENTACIÓN DE LIBRERIAS

5.3.1. Librería CanalOperaciones

La librería CanalOperaciones contiene todo un conjunto de algoritmos encapsulados en funciones independientes que se encargan de manipular objetos de tipo CCanal y arrojar otro objeto como resultado de estas manipulaciones, en general el objeto original no es modificado.

- FUNCIONES
    - Transformar
    - CambiarRango
    - Ecualizar
    - LimitarValores

- Rotar
- GetHistograma
- AplicarMascara
- Etiquetar
- PuntoUmbralPunk
- Umbralizar
- Dilatación
- Erosion
- Apertura
- Cierre
- EsqueletoMat

5.3.1.1. Transformar

CCanal Transformar(CCanal *canal, Punto Funcion[]);

CCanal Transformar(CCanal *canal, Decimal Funcion[],Entero base);

AfxThrowUserException( );

Parámetros

Función[]:

Vector de puntos que definen una función de transformación para los valores dentro del canal.

base:

Valor que se suma a todos los elementos del canal al momento de hacer la transformación con el fin de dar una base diferente de cero a la transformación.

Observaciones

La función verifica internamente que los valores x de los puntos que componen la función de transformación sean crecientes y tenga por lo menos 3 puntos, si estas condiciones no se dan, o el canal no posee información válida, la función arroja una excepción de usuario.

5.3.1.2. CambiarRango

CCanal CambiarRango(CCanal *canal, Decimal rango, CR_TIPO Tipo_Cambio);

AfxThrowUserException( );

Parámetros

rango:

Máximo valor que tendrán los elementos del nuevo canal.

Tipo_Cambio:

Tipo de operación que será aplicada sobre el canal para obtener el cambio de rango dinámico en el canal, estos pueden ser CR_PROPORCIONAL o CR_LOGARITMICO, valores que especifican un cambio de rango en los valores de manera proporcional o de manera logarítmica.

Observaciones:

Esta función ejecuta un cambio de rango dinámico en los valores del canal, este puede ser proporcional o logarítmico. En caso de que el canal de entrada no posea información válida, la función arroja una excepción de usuario.

5.3.1.3. Ecualizar

CCanal Ecualizar(CCanal *canal);

AfxThrowUserException( );

Observaciones

Esta función ejecuta un algoritmo de ecualización del histograma del canal, independientemente del tipo de canal que se trate, si es un canal R, G, B, S, I o V, la ecualización se ejecuta sobre 256 niveles, si es un canal H, se ejecuta sobre 359 niveles y si es sobre un canal ND, es buscado el mayor elemento del canal para obtener el número de niveles a ecualizar.

5.3.1.4. LimitarValores

CCanal LimitarValores(CCanal *canal,Decimal menor, Decimal mayor);

AfxThrowUserException( );

Parámetros

menor:

Valor que define el menor valor permitido dentro de canal.

mayor:

Valor que define el mayor valor permitido dentro de canal.

Observaciones

Esta función tiene como objetivo restringir los valores contenidos en el canal, de manera que los valores por debajo de menor, serán convertidos al valor especificado por menor y los valores por encima de mayor serán convertidos al valor especificado por mayor.

5.3.1.5. Rotar

CCanal Rotar(CCanal *canal,CPoint origen,Decimal grados);

AfxThrowUserException( );

Parámetros

orignen:

Punto que define las coordenadas de referencia para la rotación, es decir, el eje de rotación.

grados:

Número de grados que se rotarán los valores del canal en el sentido horario.

Observaciones

Esta función tiene como objetivo devolver un canal que es el resultado de aplicar una rotación horaria al canal que se ingresa como parámetro.

5.3.1.6. GetHistograma

Histograma *GetHistograma(CCanal *canal, Entero NumMarcas=256, BOOL ExtenderMarcas=TRUE);

AfxThrowUserException( );

Parámetros

NumMarcas:

Es el número de marcas de clase en que será dividido el histograma.

ExtenderMarcas:

Si ExtederMarcas es FALSE se divide del mínimo al máximo del canal en el número de marcas de clase, si es TRUE las marcas de clase van desde 0 hasta NumMarcas-1.

Observaciones

Esta función obtiene el histograma de un canal, el histograma está representado por un vector dinámico de una estructura de dos miembros, la marca de clase y su frecuencia.

typedef struct Histograma
{
Entero frecuencia;
Decimal marca;
}Histograma;

5.3.1.7. AplicarMascara

CCanal AplicarMascara(CCanal *canal, CCanal *mascara);
Decimal AplicarMascara(const Entero &x, const Entero &y, CCanal *canal, CCanal *mascara)
AfxThrowUserException( );

Retorno

El primer prototipo de la función retorna un canal que es producto de la convolución entre los dos canales que se pasan como parámetros.
El segundo prototipo devuelve un valor decimal que es producto de aplicar la convolución del segundo canal sobre un punto específico del primero.

Parámetros

mascara:
Canal que define la matriz que se hará operar en convolución con el primer canal que se pasa como parámetro.
x y:
Coordenadas especificas donde se quiere aplicar la convolución.

Observaciones

Esta función tiene como objetivo aplicar una operación de convolución entre las matrices que definen los canales que se pasan como parámetros a la función. Esta es muy útil en la aplicación de filtros espaciales de paso alto, paso bajo y detección de bordes entre otros.

Para más información acerca de filtros espaciales refiérase al capítulo relacionado con el marco teórico de este mismo libro.

5.3.1.8. Etiquetar

CCanal Etiquetar(CCanal *canal);

CCanal Etiquetar(CCanal *canal, CObArray *segmentos);

AfxThrowUserException( );

Parámetros

segmentos:

Array CObject de objetos de typo CArray con elementos de tipo CPoint, donde cada CArray define el conjunto de puntos que hacen parte de un segmento conexo dentro del canal.

Observaciones

La función ejecuta un algoritmo de etiquetado sobre los elementos del canal, marcando en el canal resultante, a cada segmento conexo una marca diferente y a cada elemento del segmento la misma marca

5.3.1.9. PuntoUmbralPunk

Entero PuntoUmbralPunk(Histograma *Hist);

AfxThrowUserException( );

Retorno

La función retorna el valor de umbralización para el histograma proporcionado.

Parámetros

Hist:

Histograma objeto del análisis.

Observaciones

Esta función ejecuta al algoritmo que busca un punto de umbral sobre el histograma proporcionado. El procedimiento ejecutado es el algoritmo de Punk, este algoritmo funciona sobre histograma bimodales o cuya multimodalidad pueda ser interpretada en forma de bimodal. La umbralización y este algoritmo son temas tratados en el marco teórico de este libro.

5.3.1.10. Umbralizar

CCanal Umbralizar(CCanal *canal, Decimal Umbral, Decimal val, BOOL modo=TRUE);

CCanal Umbralizar(CCanal *canal, Decimal Umbral, Decimal Min=0, Decimal Max=255,BOOL modo=TRUE);

AfxThrowUserException( );

Parámetros

Umbral:

Valor que se utilizará para umbralizar los valores del canal.

Val:

Valor que tendrán los elementos umbralizados en el primer prototipo de la función.

Min:

Mínimo valor que tendrán los elementos umbralizados en el segundo prototipo de la función.

Max:

Máximo valor que tendrán los elementos umbralizados en el segundo prototipo de la función.

Modo:

Variable que define el sentido de la umbralización.

Observaciones

Estas funciones ejecutan un algoritmo de umbralización sobre el canal pasado como parámetro. En el primer prototipo de la función, los valores por encima del umbral toman el valor especificado por “val” y los demás elementos no son modificados siempre que “modo” sea igual a TRUE, en caso contrario los valores umbralizados serán los menores a “val”.

En el segundo prototipo de la función, los valores por encima de “val” son establecidos al valor especificado por “Máximo” y los valores menores que “val” son establecidos al valor dado por “Mínimo” siempre que “Modo” sea igual a TRUE, en caso contrario el sentido de la umbralización se invierte.

5.3.1.11. Dilatacion

CCanal Dilatacion(CCanal *canal, CCanal* estructura);

BOOL Dilatacion(Entero &x, Entero &y, CCanal *canal, CCanal*estructura);

AfxThrowUserException( );

Retorno

El primer prototipo de la función devuelve el canal resultante de la dilatación sobre el canal pasado como parámetro. El segundo prototipo de la función devuelve TRUE en caso de poder ejecutar la dilatación sobre el canal especificado.

Parámetros

estructura:

Canal que definen el elemento estructurante que se aplicara en la operación morfológica sobre el canal.

Observaciones

Esta función ejecuta la operación morfológica básica de dilatación sobre el canal especificado. La diferencia primordial entre los dos prototipos de la función radica en la forma de devolver el resultado, el primer prototipo crea un objeto que guarda el resultado, mientras que el segundo modifica el canal que se le pasa como parámetro, devolviendo el resultado de la operación sobre este mismo. Debido a que el segunda prototipo no requiere la creación de otro objeto y la ejecución del

algoritmo se realiza sobre el mismo canal, su ejecución es más rápida pero se pierde la información del canal de entrada.

5.3.1.12. Erosion

CCanal Erosion (CCanal *canal, CCanal* estructura);
BOOL Erosion (Entero &x, Entero &y, CCanal *canal, CCanal*estructura);
AfxThrowUserException( );

Retorno

El primer prototipo de la función devuelve el canal resultante de la erosión sobre el canal pasado como parámetro. El segundo prototipo de la función devuelve TRUE en caso de poder ejecutar la erosión sobre el canal especificado.

Parámetros

estructura:

Canal que definen el elemento estructurante que se aplicara en la operación morfológica sobre el canal.

Observaciones

Esta función ejecuta la operación morfológica básica de erosión sobre el canal especificado. La diferencia primordial entre los dos prototipos de la función radica en la forma de devolver el resultado, el primer prototipo crea un objeto que guarda el resultado, mientras que el segundo modifica el canal que se le pasa como parámetro, devolviendo el resultado de la operación sobre este mismo. Debido a que el segunda prototipo no requiere la creación de otro objeto y la ejecución del algoritmo se realiza sobre el mismo canal, su ejecución es más rápida pero se pierde la información del canal de entrada.

5.3.1.13. Apertura

CCanal Apertura(CCanal *canal,CCanal* estructura);
AfxThrowUserException( );

Parámetros

estructura:

Canal que definen el elemento estructurante que se aplicara en la operación morfológica sobre el canal.

Observaciones

Esta función ejecuta un algoritmo que realiza una operación morfológica de apertura sobre el canal especificado y con el elemento estructurante pasado como parámetro a través del parámetro “estructura”.

5.3.1.14. Cierre

CCanal Cierre(CCanal *canal,CCanal* estructura);

AfxThrowUserException( );

Parámetros

estructura:

Canal que definen el elemento estructurante que se aplicara en la operación morfológica sobre el canal.

Observaciones

Esta función ejecuta un algoritmo que realiza una operación morfológica de cierre sobre el canal especificado y con el elemento estructurante pasado como parámetro a través del parámetro “estructura”.

5.3.1.15. EsqueletoMAT

CCanal EsqueletoMAT(CCanal *canal);

Observaciones

Esta función implementa un algoritmo de esqueletización mediante le procedimiento de transformación del eje medio (M.A.T - Medial Axis Transformation),

5.3.2. Librería ImagenOperaciones

Esta librería contiene la implementación de algoritmos encaminados a la manipulación de objetos de tipo CImagen y la información contenida dentro de los tres canales que lo componen de manera simultánea. Por el momento esta librería esta compuesta de cuatro funciones que transforman el formato de representación de la imagen contenida en el objeto.

En general, a las funciones que hacen parte de esta liberaría se les pasa como parámetros una referencia al objeto CImagen que será motivo de análisis o manipulación. Aunque se pasa una referencia al objeto los datos de este generalmente no son modificados, la referencia se pasa para que la función no cree un objeto temporal, copia del que se pasa como parámetro, lo cual consumiría un tiempo considerable. El resultado de la manipulación del objeto es devuelto en la forma de otro objeto CImagen.

- FUNCIONES
    - ImagenRGB2HSI
    - ImagenHSI2RGB
    - ImagenRGB2HSV
    - ImagenHSV2RGB

5.3.2.1. ImagenRGB2HSI

CImagen ImagenRGB2HSI(CImagen &imagen);
AfxThrowUserException();

Observaciones

Esta función ejecuta un algoritmo que transforma los datos RGB contenidos en los tres canales del objeto CImagen al formato HSI, estos datos son luego colocados en los canales A, B, C del objeto CImagen que será devuelto por la función.

5.3.2.2. ImagenHSI2RGB

CImagen ImagenHSI2RGB(CImagen &imagen, Entero modo=0);
AfxThrowUserException();

Parámetros

Modo:

Valor que define el tipo de ajuste en los valores HSI para dar un valor RGB dentro del rango visualizable en el monitor. Debido a que la tripleta de valores RGB resultado de convertir ciertos valores HSI puede estar fuera del rango 0-255, se hace necesario modificar los valores HSI de manera que su conversión al formato RGB este dentro del rango permitido para visualización. Los valores permitidos para este parámetro son 0 para variar la saturación hasta obtener un valor RGB válido, 1 para variar la luminosidad hasta obtener un valor RGB válido o cualquier otro para no variar los valores HSI.

Observaciones

Esta función ejecuta un algoritmo que transforma los datos HSI contenidos en los tres canales del objeto CImagen al formato RGB, estos datos son luego colocados en los canales A, B, C del objeto CImagen que será devuelto por la función.

Como ya se mencionó, algunos colores RGB contenidos a partir de colores HSI pueden ser no representables en monitores por estar fuera del rango 0-255, mas específicamente porque alguno de sus valores R, G o B suelen estar por encima de 255, en este caso, antes de ejecutar la conversión a RGB, uno de los valores S o I del color HSI es disminuido hasta asegurar que su conversión en RGB será válida. El valor H no es modificado ya que esto modificaría drásticamente el resultado final.

5.3.2.3. ImagenRGB2HSV

```
CImagen ImagenRGB2HSV(CImagen &imagen);
AfxThrowUserException();
```

Observaciones

Esta función ejecuta un algoritmo que transforma los datos RGB contenidos en los tres canales del objeto CImagen al formato HSV, estos datos son luego colocados en los canales A, B, C del objeto CImagen que será devuelto por la función.

5.3.2.4. ImagenHSV2RGB

CImagen ImagenHSV2RGB(CImagen &imagen);

AfxThrowUserException();

Observaciones

Esta función ejecuta un algoritmo que transforma los datos HSV contenidos en los tres canales del objeto CImagen al formato RGB, estos datos son luego colocados en los canales A, B, C del objeto CImagen que será devuelto por la función.

5.4. SERIACION

La seriación es el proceso de escribir o de leer el contenido de uno o más objetos a y desde un fichero. Normalmente la operación de enviar una serie de objetos a un fichero en disco para hacerlos persistentes recibe el nombre de *seriación,* y la operación de leer o recuperar su estado del fichero en disco para reconstruirlos en memoria recibe el nombre de *deseriación.* El esquema gráfico que responde a este proceso es el siguiente:

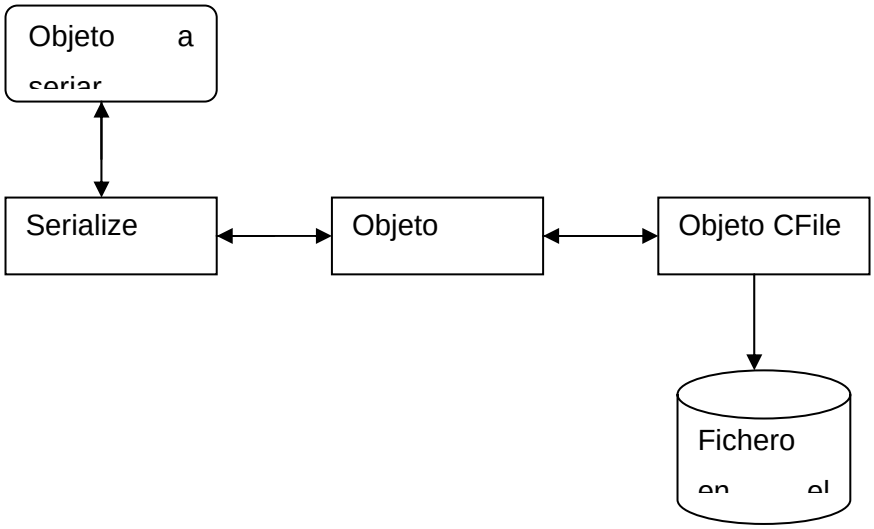


Figura 68. *Diagrama de seriación*

Para poder seriar los objetos de una clase:

Derive la clase de **CObject.**
Ponga la macro DECLARE_SERIAL dentro de la declaración de la clase (fichero .h) y la macro IMPLEMENTE_SERIAL en la que la implementación de la misma (fichero .cpp) para que sea procesada solo una vez.
Defina un constructor sin argumentos.
Redefina la función miembro **Serialize**

El siguiente ejemplo, pone en práctica loas pasos enunciados anteriormente para diseñar una clase de forma que sus objetos pueden seriarse:

```
// fichero.h
Class CMiclase : public CObject
[
        DECLARE_SERIAL( CMiclase )
        Public:
          CString m_nombre:
          Int     m_numero:


          CMiclase()[]: //constructor sin argumento
          CMiclase(CString nom. Int num=0):
          Void Serialize( CArchive &ar ):
          // sigue la declaración de la clase
]:
////////////////////////////////////////////////////////// fichero .cpp
IMPLEMENT_SERIAL( CMclase. CObject. 2 )
// ...
```

El tercer argumento de esta macro es un número de versión ($n \geq 0$) que se refiere a la implementación de la clase; se asigna a la clase cuando la macro IMPLEMENT_SERIAL se procesa. El número de versión será codificado en el fichero para permitir al programa que realice la deseriación identificar y manejar los datos creados por versiones anteriores de programa.

Al redefinir la función miembro **Serialize,** primero llamamos a la versión de **Serialize** de la clase base pata asegurar que la parte heredada del objeto también es seriada. A continuación se insertan o se extraen las variables miembro de nuestra clase. El operador de inserción <<, y el operador de extracción,>>.

```
Void CMiclase::Serialize( CArchive &ar )
[
  //Llamar a la versión Serialize de la clase base
  CObject::Serialize( ar ):

  //Insertar o extrarer los datos miembro de nuestra clase
  if ( ar.IsStoring() )
    ar <<m_nombre << (WORD)m_numero:
  else
  [
    WORD w:
    ar >> m_nombre >> w:
    m_numero = (int)w:
   ]
]
```

la clase **CArchive** sólo puede seriar objetos de tipos de tamaño fijo: por ejemplo, BYTE, WORD, DWORD o LONG. Cualquier otro tipo entero debe ser convertido explícitamente a uno de estos tipos. Por ejemplo, **int** es un tipo de datos de 16 bits en Windows 3.x y de 32 bits en Windows NT. Por lo tanto, para utilizar un tamaño fijo e independiente del sistema operativo, tendríamos que realizar una conversión explícita a WORD.

Para saber sí el objeto **CArchive** se está utilizando para guardar datos (storing), que es lo mismo que escribir en el fichero, o para cargar datos (loading), que es lo mismo que leer del fichero, hay que enviar al objeto **CArchive,** en nuestro caso a ar,

el mensaje **IsStoring.** Si se está utilizando para guardar datos, la función devuelve un valor **TRUE;** en otro caso devuelve **FALSE.**

Una vez que tenemos una clase, podemos seriar objetos de esa clase a y desde un objeto de la clase **CArchive** siempre tiene que estar asociado con un fichero.

Para crear un objeto **CArchive** y asociarlo con un fichero, declare un objeto **CFile** que represente al fichero abierto y páselo como argumento al constructor de **CArchive** por ejemplo,

```
CFile Mifichero:
    Mifichero.Open("datos",CFile::modeWrite I CFile::modeCreate)
    CArchive ar( &MiFichero. CArchive::store ):
```

Observe que el fichero ha sido abierto para escribir, **modeWrite,** y que el objeto **CArchive** ha sido construido para almacenar datos, **store.**

Una vez que tenemos un objeto **CArchive** capaz de almacenar datos, podemos seriar objetos de CMiclase, así.

```
CMiclase *pObjetol = new CMiclase( "Francisco" ):
ar << pObjetol:
ar.Close():
MiFichero.Close():
Delete pObjetol:
```

Para deseriar un objeto, el proceso sería inverso al anteriormente descrito. Esto es, crearíamos un objeto **CFile** que represente al fichero del cual tenemos que leer los datos y lo conectaríamos a un objeto **CArchive** construido para extraer datos:

```
CFile MiFichero:
MiFichero.Open( "datos", CFile::modeRead ):
CArchive ar( &MiFichero, CArchive::load ):
```

Una vez que tenemos un objeto **CArchive** capaz de cargar datos, podemos deseriar objetos de CMiclase, así:

```
CMiclase *pObjetol:
ar>>pObjetol://el sistema construye un objeto dinámicamente
ar.Close():
MiFichero.Close():
```

La deseriación debe hacerse en le mismo orden que se hizo la seriación, lo que suponemos que el programa no debe perder de vista el orden en el que fueron seriados los objetos. Para evitar este inconveniente, lo mejor es colocar todos los objetos en una colección y seriar la colección como un único objeto.

Recuerde que las macros DECALRE_SERIAL e IMPLEMENT_SERIAL generan el código necesario no sólo para que los objetos de una clase derivada de **CObject** puedan seriarse, sino también para que durante este proceso se construyan dinámicamente. Por ejemplo, el objeto CMiclase referenciado por pObjectol es construido dinámicamente en el mismo proceso de deseriación

6. MANUAL DE USUARIO

El siguiente es un manual de usuario que describe el funcionamiento y las herramientas disponibles dentro de la interfaz gráfica implementada con el fin de integrar y manipular los algoritmos que llevarán a cabo los procesos aplicados a la imagen digital del pluviograma durante su manipulación.

6.1. INTERFAZ GENERAL

La aplicación que integra los algoritmos de procesamiento de imágenes y métodos numéricos dispone de una interfaz gráfica principal que integra las herramientas que dan acceso a la manipulación de estos algoritmos. ES de resaltar que esta interfaz y en general la aplicación, no provee de la manipulación de todas las rutinas implementadas ni de todas las opciones y parámetros de los que se pueden disponer en estas, solamente permite el manejo de un subconjunto de rutinas que demostraron ser muy útiles en la consecución de las metas planteadas a lo largo de este proyecto.

La interfaz gráfica principal está compuesta de cuatro elementos básicos:

- Menú principal.
- Barras de herramientas
- Barra de estado
- Vista

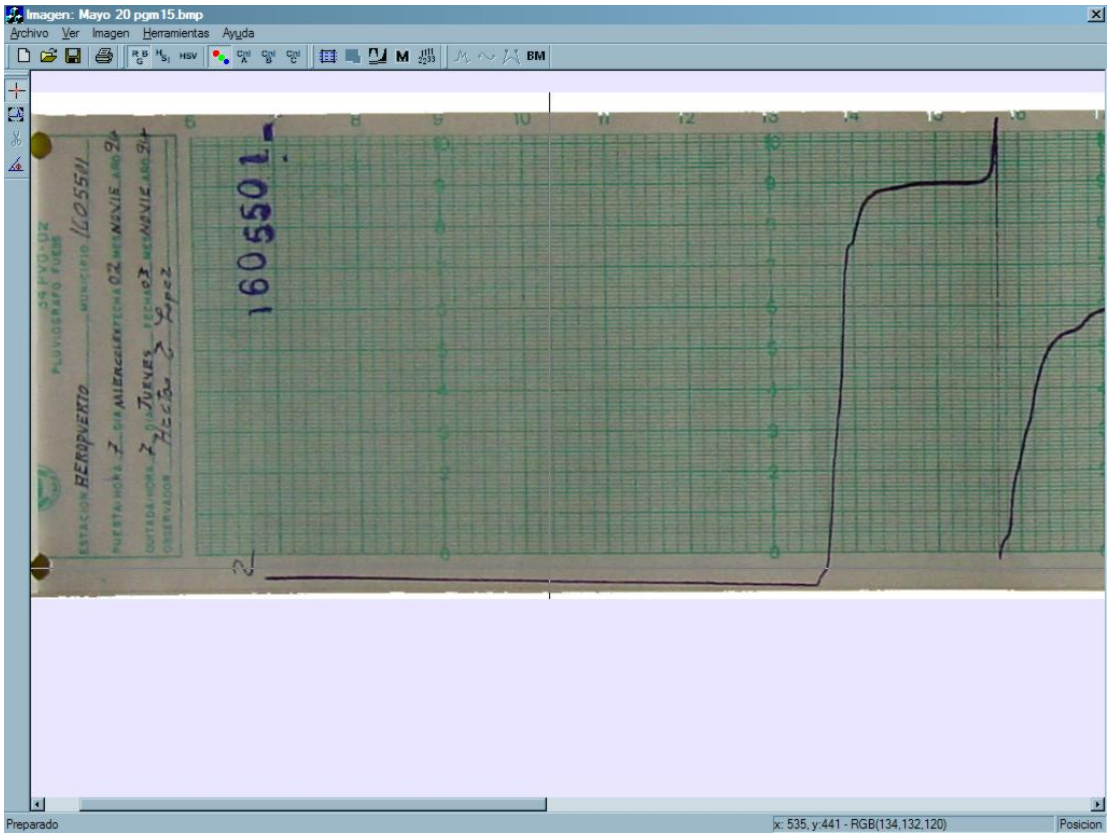


Figura 69. *Interfaz principal de la aplicación.*

Aunque sencilla, dentro de esta interfaz disponemos de todas las herramientas necesarias para la manipulación del pluviograma hasta la extracción de la señal pluviométrica. Se evito sobrecargar esta interfaz con cualquier otro tipo de elemento, como por ejemplo ventanas fraccionadas o solapas, para poner a disposición del usuario de una vista amplia donde observar los cambios surgidos dentro de la imagen y los elementos generados dentro del proceso así como los resultados de su manipulación.

6.2. MENÚ PRINCIPAL

Dentro del menú principal disponemos de las siguientes opciones básicas

- Archivo.

- Ver.
- Imagen.
- Herramientas.
- Ayuda.

6.2.1. Submenú Archivo

Dentro del submenú archivo podemos encontrar opciones que permiten el manejo de ficheros dentro de la aplicación, dentro de las cuales podemos observar las siguientes.

- Nuevo.
- Abrir.
- Guardar Como.
- Imprimir.
- Salir.

6.2.1.1. Nuevo

Esta opción del menú permite crear una vista nueva dentro de la cual cargar un nuevo fichero de la imagen digital de un pluviograma con el fin de someterlo a todo el proceso. Cuando esta opción es elegida, todos los objetos credos dentro de la aplicación son reseteados y es liberada la memoria consumida por ellos con el fin de dar paso al nuevo marco de trabajo.

Cuando se desea cargar una nueva imagen no es estrictamente necesario utilizar antes esta opción, al orden Abrir del menú Archivo realiza implícitamente esta tarea.

6.2.1.2. Abrir

Esta opción permite cargar dentro de la aplicación diferentes tipos de archivos, algunos propios de la aplicación, que describen un pluviograma objeto de estudio y análisis.

Entre los diferentes tipos de archivos que podemos cargar tenemos los siguientes:

- .BMP: Archivos de mapas de bits de 8 y 24 Bytes de resolución de color.
- .IMB: Proyecto de la aplicación.
- .IMT: Archivo de texto plano con la definición básica del pluviograma.

Cual sea el tipo de fichero elegido, este dará origen a un pluviograma representado en la vista de la aplicación.

6.2.1.3. Guardar Como

Esta opción permite guardar un proyecto en curso en la forma de ficheros propios de la aplicación. Para que esto sea así, es necesario que al pluviograma analizado se le haya hecho al menos un ajuste a trazos, curvas Spline o curvas Bezier.

- IMB: Formato de archivo en donde se guarda el pluviograma en forma de trazos, curva Spline o Bezier. Si el proyecto actual dispone de estos tres tipos de ajustes, la aplicación guardará aquellos que se encuentren señalados en la barra de herramientas de trazos. Este tipo de archivo tiene la ventaja de ser muy pequeño comparado con la imagen del pluviograma original y por tanto es bastante portable.

- IMT: Formato de archivo que guarda la definición del pluviograma en forma de texto plano donde cada fila corresponde a un punto de registro dentro del pluviograma, la primera columna de valores se corresponde con el tiempo y la segunda con las unidades de precipitación. Este formato de archivo fue creado con la intención de brindar portabilidad entre aplicaciones capaces de procesar e interpretar archivos planos

6.2.1.4. Imprimir

Esta selección presenta algunas opciones básicas y generales por medio de las cuales se puede disponer de una imagen del pluviograma a partir de los datos generados por el análisis, para lo cual es necesario haber realizado un ajuste previo de la señal a trazos, curvas Spline o Curvas Bezier.

6.2.2. Submenú Ver

Dentro de este submenú se encuentran distintas opciones que permiten visualizar o no los siguientes elementos dentro del marco de trabajo

Barras de herramientas

- Barra de herramientas principal.
- Barra de utilidades.
- Barra de formato y canales
- Barra de diálogos
- Barra de curvas e imágenes

Canales

- Ver Tricenal: Dispone de la visualización de los tres canales de una imagen en formato RGB para dar origen a una imagen a color, esta opción no está disponible cuando el formato de representación es diferente al RGB
- Ver Canal A: Dependiendo del formato, permite visualizar el canal R o H en la forma de una imagen a escala de grises.
- Ver Canal C: Dependiendo del formato, permite visualizar el canal G o S en la forma de una imagen a escala de grises.
- Ver Canal D: Dependiendo del formato, permite visualizar el canal B, S o I en la forma de una imagen a escala de grises.

Diálogos

- Unidades, medidas y escalas
- Filtro espacial
- Umbralización
- Morfología binaria
- Componentes conexas

Una descripción de estos diálogos y su utilización será descrita en las siguientes secciones de este capítulo.

Curvas y Imágenes

- Ver Trazos: Si el pluviograma ha sido ajustado a una secuencia de trazos no ajustables, esta opción permite ver dicha representación en pantalla.
- Ver Spline: Si el pluviograma ha sido ajustado a una curva Spline, esta opción permite ver dicha representación en pantalla.
- Ver Bezier: Si el pluviograma ha sido ajustado a una curva Bezier, esta opción permite ver dicha representación en pantalla.
- Ver Imagen/Canal: Esta opción permite visualizar en pantalla la imagen de componentes conexas que da origen al ajuste por trazos y que es la base para cualquier otro tipo de ajuste, es muy útil al momento de comparar los ajustes con la aproximación original.

6.2.3. Submenú Imagen

Dentro de este submenú se disponen de herramientas que permiten aplicar algunas manipulaciones básicas a la imagen o el canal activo. El canal activo es siempre el que se muestra en la vista de la aplicación, por lo tanto puede ser uno de los canales de la imagen o el resultado de la manipulación de cualquiera de ellos.

Cuando se muestra una imagen de tres canales en formato RGB, el canal activo siempre es el canal del color Rojo.

6.2.3.1. Opciones Formato RGB, Formato HSI y Formato HSV.

Estas tres opciones permites convertir desde y hacia cualquiera de los tres formatos de imágenes RGB, HSI y HSV.

6.2.3.2. Ajustar señal a Curva Spline

Esta opción permite a partir de una señal ajustada a un trazo, realizar un ajuste de esta a una curva Spline y disponerla para su manipulación interactiva en la vista.

6.2.3.3. Ajustar señal a Curva Bezier

Esta opción permite a partir de una señal ajustada a un trazo, realizar un ajuste de esta a una curva Bezier y disponerla para su manipulación interactiva en la vista.

6.2.3.4. Liberar Curvas

Esta opción libera los recursos utilizados por los ajustes de curvas a Spline y Bezier eliminando por ente dicha representación hasta que se vuelva a escoger alguna de las opciones de ajuste.

6.2.4. Submenú Herramientas

6.2.4.1. Seleccionar

Elegida esta herramienta se está en disposición de manipular cualquiera de los elementos dentro de la vista que así lo permitan, como por ejemplo, trazos o curvas ajustadas, nodos Spline, Bezier o Aspas de ajuste de nodos Bezier. Cuando no se está manipulando ningún elemento, información referente al la posición del cursor

es desplegada en la barra de herramientas como por ejemplo las coordenadas y el color.

6.2.4.2. Selección rectangular

Esta opción pone a disposición del usuario un rectángulo de selección que permite escoger una subsección del mapa de bits original para luego ser recortado.

6.2.4.3. Cortar

Permite recortar una subsección del mapa de bits original delimitada por el rectángulo de selección desplegado por la herramienta Selección Rectangular, a partir de este momento toda la información de la imagen fuera de este rectángulo se pierde

6.2.4.4. Enderezar imagen

Esta herramienta permite girar la imagen con el fin de enderezarla con respecto a la horizontal. Esta es una opción muy útil cuando la imagen del pluviograma se encuentra un poco girada. Esta operación se lleva a cabo convirtiendo una línea recta unida por dos puntos en pantalla en una línea horizontal cuyo ángulo de inclinación tiende a cero, luego, toda la imagen es rotada al igual que esta línea.

Para enderezar la imagen se recomienda seguir los siguientes pasos:

- Hacer clic en la herramienta enderezar del submenú herramientas en el correspondiente botón de la barra de herramientas.
- Identificar una línea recta horizontal dentro del pluviograma que tenga la mayor extensión posible, no importa si esta línea no es perfectamente horizontal, el objetivo de esta herramienta es enderezarla.
- Hacer clic al inicio y luego al final de esta línea.
- Espere mientras se realiza la operación, esto puede tardar varios segundos

Debido a que esta es una operación pesada en cuanto a cálculos se refiere, se recomienda hacer primero una selección del área útil de la imagen y luego recortarla para luego realizar el enderezamiento.

6.2.4.5. Análisis de la señal

Esta herramienta analiza la señal pluviométrica que se encuentra en la forma de un trazo, curva Spline o Curva Bezier y obtiene a partir de este análisis la correspondiente Curva Masa, Histograma de precipitación e histograma de intensidad.

Al escoger esta herramienta se muestra un diálogo donde se puede especificar la unidad de muestreo para el análisis de cada una de las gráficas y ejecutar el análisis, luego se puede ver cada gráfica haciendo clic en el botón correspondiente en este mismo diálogo.

6.3. BARRAS DE HERRAMIENTAS

Dentro de la aplicación disponemos de 4 barras de herramientas, cada una de ellas ejecuta un conjunto de tareas en común.

- Barra de herramientas principal.
- Barra de utilidades.
- Barra de formato y canales
- Barra de diálogos
- Barra de curvas e imágenes

Cada una de las herramientas mencionadas en cada barra fue explicada anteriormente en la sección que trata acerca del menú principal.

6.3.1. Barra de herramientas principal

Figura 70. *Barra de herramientas principal*

En esta barra de herramientas se encuentran las operaciones básicas de la aplicación referentes al manejo de archivos, las tareas que puede ejecutar son las siguientes:

- Crear nuevo documento.
- Abrir archivo.
- Guardar archivo.
- Imprimir

6.3.2. Barra de utilidades.

Figura 71. *Barra de utilidades.*

En la barra de utilidades disponemos de las siguientes opciones.

- Herramienta selección.
- Herramienta selección rectangular.
- Herramienta cortar subsección.
- Herramienta enderezar imagen.

6.3.3. Barra de formato y canales

Figura 72. *Barra de formato y canales*

Esta barra de herramientas contiene opciones referentes a la visualización de mapas de bits en la vista de la aplicación. Las opciones son las siguientes.

- Convertir a RGB.
- Convertir a HSI.
- Convertir a HSV.
- Ver tricanal.
- Ver Canal A.
- Ver Canal B.
- Ver Canal C.

6.3.4. Barra de diálogos

Figura 73. *Barra de diálogos*

Mediante esta barra de herramientas se tiene un rápido acceso a los diálogos que se utilizan para aplicar ciertos tratamientos especiales a la imagen. Los siguientes son los diálogos que se pueden desplegar mediante la utilización de esta barra de herramientas.

- Unidades, medidas y escalas
- Filtro espacial
- Umbralización
- Morfología binaria
- Componentes conexas

6.3.5. Barra de curvas e imágenes

Figura 74. *Barra de curvas e imágenes*

En esta barra de herramientas se disponen opciones para la manipulaciones de trazos y curvas Spline o Bezier. Sus miembros son los siguientes.

- Ver Trazos.
- Ver Spline.
- Ver Bezier.
- Ver Imagen/Canal.

6.4. LOS DIALOGOS DE LA APLICACIÓN

Algunos de los algoritmos más importantes utilizados en el procesamiento de pluviogramas propuesto en este proyecto requieren del ajuste de ciertos parámetros con el fin de adaptar su funcionamiento a un pluviograma particular. Con este fin se han creado un conjunto de diálogos que definen las interfaces en las cuales estos algoritmos serán parametrizados por el usuario. Cada uno de estos diálogos contiene una o más funcionalidades agrupadas según la labor que desempeñan. Los diálogos son los siguientes

- Unidades, medidas y escalas
- Filtro espacial
- Umbralización
- Morfología binaria
- Componentes conexas
- Análisis

6.4.1. Diálogo filtro espacial

Este diálogo permite manipular un algoritmo que aplica un filtrado espacial al canal activo (paso alto, paso bajo, etc.) para lo cuál dispone de una matriz donde podemos definir el filtro que deseamos aplicar así como las dimensiones de este.

Este diálogo permite previsualizar el resultado del filtro oprimiendo el botón correspondiente, una vez obtenidos los resultados deseados se puede hacer clic en el botón Aplicar que se encuentra bajo la matriz.

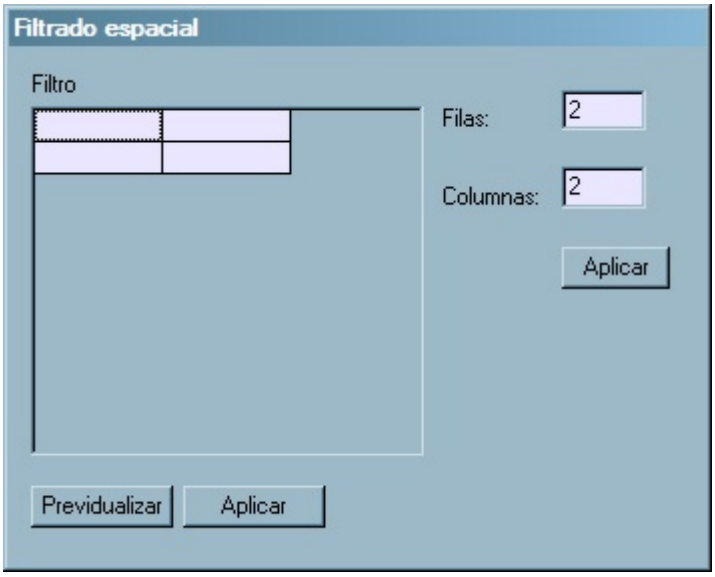


Figura 75. *Diálogo Filtro espacial*

De esta manera se pueden aplicar filtros con el objetivo de reafirmar o disminuir ciertas características u objetos dentro de la imagen.

6.4.2. Diálogo de unidades y medidas.

Este es uno sencillo pero importante diálogo que permite especificar a la aplicación la escala en que se encuentra el pluviograma original y los límites que tendrá esta dentro de la vista de la aplicación.

Generalmente se trabaja con pluviogramas de 24 horas y cuya escala de precipitación varia de o a 10 cm. Con miras a extraer la señal de la imagen, es necesario suplir al software de información que le permita calcular proporciones entre la escala de la imagen y la posición que la señal ocupa en la pantalla, es decir, la hora 0 podría estar en el pixel 100,x y la hora 23 en el pixel 1000,x de esta manera 24 horas de registro pluviométrico estarían distribuidos uniformemente en

900 pixeles horizontales, de lamisca manera 0 centímetros de precipitación podrían estar en el pixel x,500 y 10 centímetros de precipitación en el pixel x,50, de esta manera 10 cm. De precipitación estarían distribuidos uniformemente en 450 pixeles dentro de la vista de la aplicación.

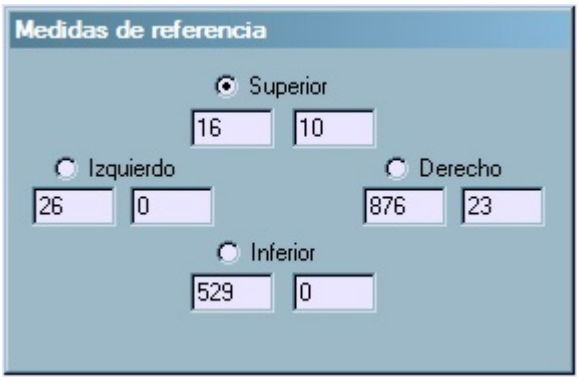


Figura 76. *Dialogo unidades y medidas*

El diálogo de unidades y medidas ayuda precisamente a definir estos valores, los cuales pueden ser digitados directamente en las casillas de texto dispuestas para este fin. Con la intención de hacer más cómoda esta tarea, la vista de la aplicación soporta la selección interactiva de los puntos en pantalla que definen la escala del pluviograma, de esta manera dichos puntos pueden ser seleccionados haciendo clic en el lugar de la pantalla que define cada límite.

6.4.3. Dialogo umbralización

Este diálogo permite ejecutar una umbralización sobre el canal activo de la aplicación. El diálogo muestra en todo momento el histograma del canal seleccionado y al pasar el Mouse por encima de está gráfica disponemos de información referente a la marca de clase correspondiente del histograma como cantidad de apariciones y frecuencia estadística, también se muestra información general como el total de puntos de la imagen.

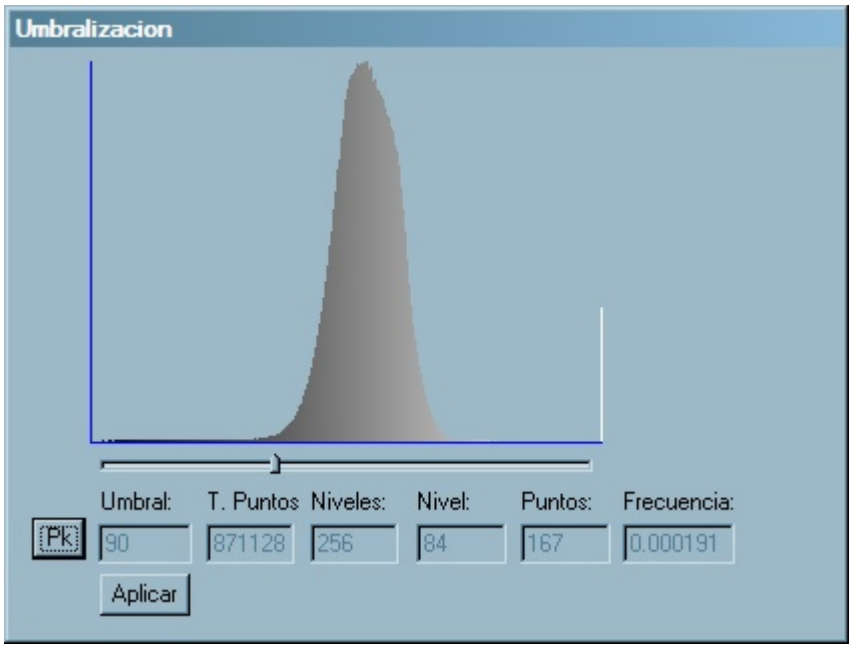


Figura 77. *Diálogo umbralización*

Este importante diálogo dispone de dos herramientas interesantes, la primera es una barra de deslizamiento que permite elegir un umbral de manera interactiva al tiempo que el resultado de la umbralización se puede visualizar en pantalla, la segunda es un botón de comando que ejecuta sobre el histograma el algoritmo de "Punk" para calcular un umbral sugerido automático, el cual puede ser también modificado por la barra descrita anteriormente.

Es importante recalcar que la umbralización no se lleva es definitiva hasta que no se haga clic sobre el botón "Aplicar" de este cuadro de diálogo.

6.4.4. Diálogo morfología binaria

Este diálogo pone a disposición del usuario las herramientas necesarias para aplicar sobre el canal activo algunos algoritmos básicos de morfología matemática. Este diálogo dispone de una matriz que permite definir el elemento estructurante que se aplicará sobre el canal así como sus dimensiones. Dos aspectos importantes se deben mencionar acerca de este procedimiento.

- Los algoritmos morfológicos implementados solo se pueden aplicar sobre imágenes binarias (blanco puro y negro puro), si el canal activo no cumple con esta condición debe ser primero sometido al procedimiento de umbralización.
- Algunos algoritmos morfológicos no requieren de elemento estructurante, por ejemplo el algoritmo de esqueletización por transformación del eje medio.

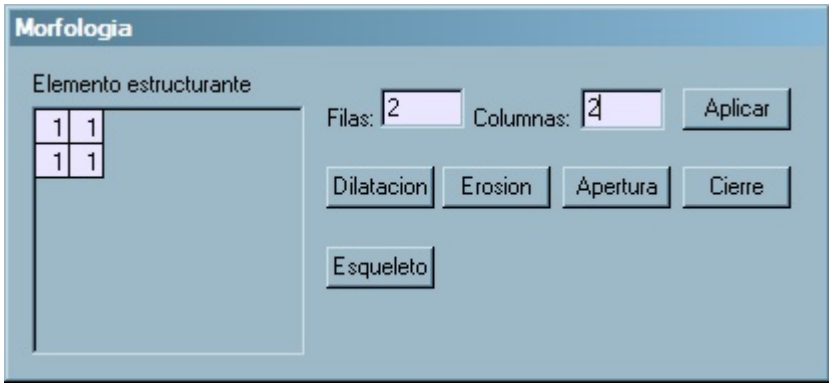


Figura 78. *Diálogo Morfología matemática.*

La aplicación de morfología matemática sobre imágenes binarias es importante en la reconstrucción de segmentos de líneas conexos y la eliminación de elementos no pertenecientes a la señal final.

6.4.5. Diálogo componentes conexas

El diálogo de componentes conexas permite realizar la extracción de la señal pluviométrica a un conjunto de vectores que definan los segmentos en que se descompone, para luego ser sometido a una selección, inicialmente automática, de los segmentos que harán parte de la señal final. Este diálogo permite definir el valor umbral de las componentes que harán parte de la señal y eliminar componentes conexas. Haciendo clic en el botón Etiquetar, las componentes conexas son calculadas mediante el etiquetado de estas en el canal activo.

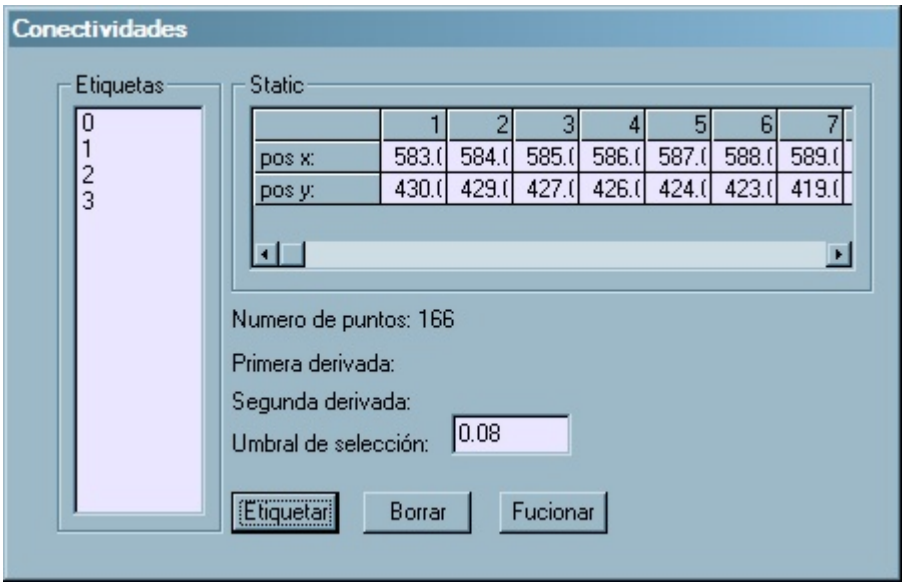


Figura 79. *Dialogo de etiquetado y componentes conexas.*

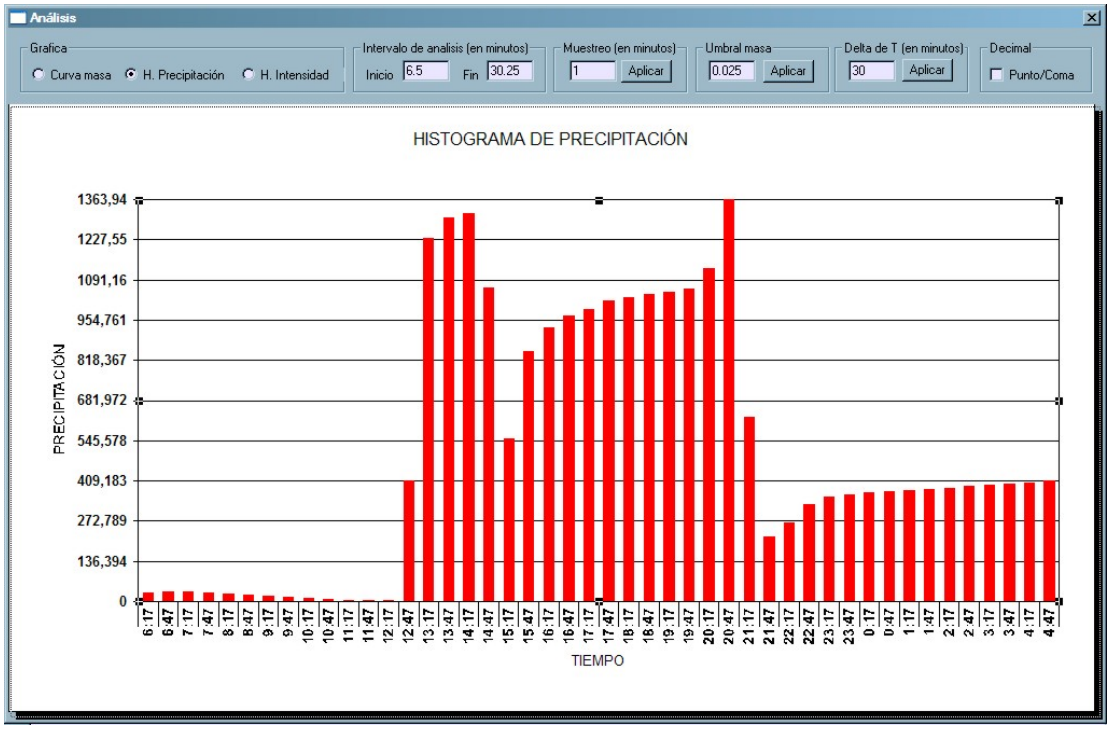
En todo momento este diálogo presenta una lista de las componentes conexas numeradas donde se puede elegir una de ellas, la cual aparecerá en pantalla de una manera resaltada. También se presenta en todo momento una lista de los puntos que hacen parte de la componente seleccionada y el número de puntos de esta componente. Una vez elegida una componentes esta puede ser eliminada haciendo clic en el botón Borrar. Una vez elegidas las componentes que harán parte de la señal final, estas deben se unidas en un solo vector haciendo clic en el botón Fundir.

El umbral es un valor de 0 a 1 que funciona como un filtro que hace que solo se tomen en cuenta las componentes que tienen más de cierta proporción de punto con respecto a la componente más extensa.

El etiquetado de componentes conexas y por tanto todos los demás procedimientos solo puede ser aplicado a imágenes binarias.

6.4.6. Diálogo de análisis

El diálogo de análisis tiene como objeto ejecutar sobre la señal extraída algunos algoritmos que calculan información derivada de los valores contenidos en la señal. En este diálogo se muestra la Curva Masa, el Histograma de precipitación y el Histograma de intensidad.



En este diálogo podemos encontrar un conjunto de controles que parametrizan la ejecución del análisis y el comportamiento de la gráfica y están divididos así:

- Un conjunto de controles que permiten escoger el tipo de gráfica a mostrar en pantalla (curva masa, histograma de precipitación o histograma de intensidad).
- Dos casillas de texto que permite especificar el intervalo de tiempo a graficar, por ejemplo, si se tiene un Pluviograma que va de 6 de la mañana a las 8 de la mañana siguiente y se quiere graficar solo de 9:30 de la mañana a 4:15 de la mañana siguiente se deben digitar los siguientes valores: 9.5 y 28.25
- Una casilla de texto que permite especificar cada cuanto tiempo se acumularan valores para construir la curva masa, es decir, si se digita un valor de 1 un nuevo valor para la curva masa será calculado en intervalos de

1 minuto en el pluviograma. Si se coloca un valor de cero la acumulación se hará con la máxima resolución de tiempo posible.

- Una casilla de texto que permite especificar el umbral que se utilizará para considerar cuando no hay cambio en la precipitación registrada, caso en el cual no se acumulará el valor en la curva masa. Funciona de la siguiente manera, si se establece que un nuevo valor de curva masa será calculado cada minuto y el umbral es de 0.025 mm quiere decir que si la diferencia entre el valor de precipitación en un instante de tiempo dado y el de hace un minuto es menor a 0.025 se interpreta como si no hubiera cambio en la precipitación y por tanto no se acumula en la curva masa, luego el nuevo valor para la curva masa es igual al anterior.
- Una casilla de texto que permite especificar el Delta de T para calcular los histogramas de Precipitación e Intensidad.

6.5. MANIPULACIÓN DE LA SEÑAL AJUSTADA A CURVA SPLINE O BEZIER.

Como se ha mencionado anteriormente es posible ajustar la señal pluviométrica contenida en un trazo a una curva Spline o una Curva Bezier con el fin de manipularla de una manera coherente e interactiva.

Las curvas Spline y Bezier estar definidas por un conjunto de nodos unidos entre sí pero a diferencia de Spline, cada nodo Bezier posee además de dos puntos de control, los cuales hemos denominado “Aspas”. La interfaz gráfica implementada facilita la edición de las curvas Spline y Bezier mediante la manipulación de estos elementos dentro de la propia vista de la aplicación y con la ayuda del Ratón “Cono si se tratase” de un software de edición de imágenes vectoriales, de esta manera es posible mover, borrar o agregar nodos pertenecientes a cualquiera de las dos curvas o mover las Aspas de un nodo Bezier en particular de una manera libre pudiendo nodos esquinas o conservando la suavidad del nodo manteniendo un ángulo de 180 grados entre ambos puntos de control.

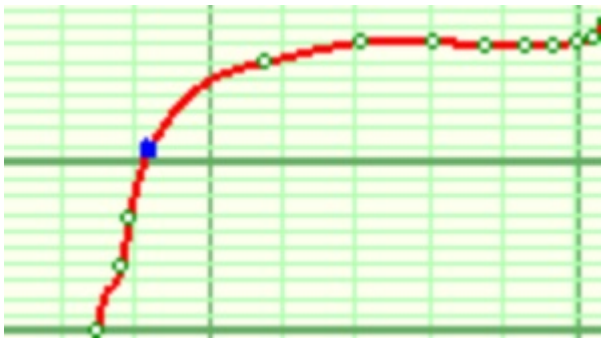


Figura 80. *Segmento de ajuste a curva Spline, el nodo azul se encuentra seleccionado.*

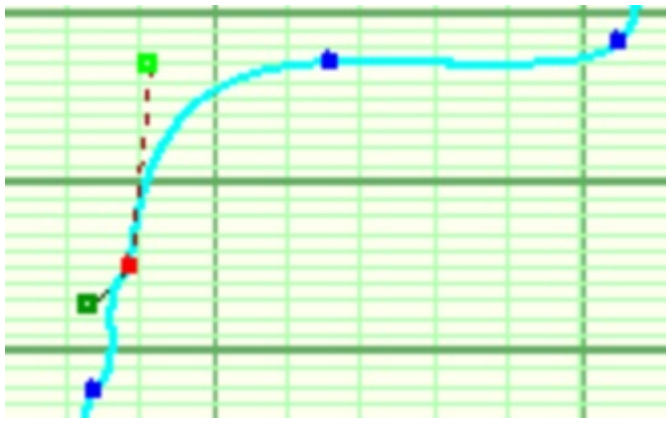


Figura 81. *Segmento de ajuste a curva Bezier, el nodo rojo se encuentra seleccionado al igual que el Aspa que se encuentra arriba de este.*

Las operaciones mencionadas anteriormente se pueden realizar de la siguiente manera:

- Para mover un nodo selecciónelo con el botón izquierdo del Mouse y arrástrelo hasta su nueva posición.
- Para mover el Aspa de un nodo Bezier seleccione primero el nodo en cuestión, esta hará que se visualicen sus Aspas, luego selecciones de la misma manera una de sus Aspas y arrástrela hasta su nueva posición. Si Hace esta operación de manera libre podrá crear nodos de esquina, si desea conservar la suavidad del nodo, presione la tecla “Shift” al tiempo que mueve el Aspa, esto hará que se conserve un ángulo de 180 grados entre los puntos de control definidos por cada Aspa.

- Para borrar un nodo, selecciónelo y presione la tecla “Delete”.
- Para crear un nuevo nodo, mantenga presionada la tecla “Control” y haga clic en algún lugar sobre la curva Spline o Bezier donde no se halle ubicado ningún nodo.

7. CONCLUCIONES

- Durante la implementación del proyecto se constató como la programación orientada a objetos constituye un paradigma enfocado a insertar al software en un marco de pensamiento muy propio del ser humano facilitando la comprensión de este a medida que se hace más complejo independientemente. También se observó como la POO facilita la detección y el manejo de de errores evitando su propagación hacia módulos externos.

- Con herramientas poco complejas en el tratamiento de imágenes digitales es posible obtener resultados promisorios en cuanto al objetivo de extraer señales provenientes de hojas de papel, los resultados finales dependen directamente del estado de la imagen analizada y del estado de la señal. La utilización de herramientas más complejas en el tratamiento de imágenes digitales promete no solo arrojar mejores resultados finales si no también dar a los algoritmos implementados más flexibilidad en cuanto a la calidad de las imágenes que están en condiciones de procesar.

- El lenguaje de programación visual C++ demostró ser un excelente lenguaje de alto nivel enfocado a brindar al programador de la flexibilidad que se requiere cuando no se desean límites a la hora de desarrollar, aunque esto suponga un mayor cuidado en la utilización de las herramientas de las que dispone y responsabilidad por parte del programador a la hora de administrar recursos. Los punteros y el acceso directo a la memoria se convirtieron en el pie de apoyo del proyecto al momento de implementar estructuras que brindaran a los algoritmos de buenas velocidades de ejecución. Las herramientas de optimización ofrecen una versatilidad única haciendo que las aplicaciones aprovechen al máximo los recursos de procesamiento disponibles en el sistema de cómputo.

- Los métodos numéricos en general se establecen como una herramienta de suma importancia en la resolución de problemas donde las matemáticas

tradicionales como el cálculo no pueden brindar soluciones analíticas. Por otra parte, la versatilidad de esta metodología y su naturaleza procedimental la hace mas que apta para la utilización en computadoras.

- Las condiciones de captura de la imagen digital constituyen un elemento esencial para el procesamiento, no importa que tan buen estado tenga la imagen física original, si las condiciones de captura como iluminación, resolución del dispositivo de captura, profundidad de color o formato de compresión no son las adecuadas, la imagen capturada puede ser no apta para su procesamiento.

BIBLIOGRAFIA

- *Berger, Mark*. **Computer Graphics with Pascal**. *Benjamín Cummings*
- *Ceballos, Fco. Javier.* **Microsoft Visual C++ 6.0, aplicaciones para Win 32.** Segunda edición. *Alfaomega y ra-ma editores*.
- *Ceballos, Fco. Javier*. **Microsoft Visual Programación avanzada en Win32.** *Alfaomega y ra-ma editores.*
- *Ceballos, Fco. Javier*. **Programación Orientada a Objetos con C++.** Segunda edición. *Alfaomega y ra-ma editores.*
- Gonzáles, Rafael C. **Tratamiento Digital de imágenes.** ADDISON-WESLEY
- Jacobson, Ivar. Booch, Grady. Rumbaugh, James. **El Lenguaje Unificado de Modelado.** Primera edición*. Addison Wesley. España, 1999.*
- Jacobson, Ivar. Booch, Grady. Rumbaugh, James. **El Proceso Unificado de Desarrollo de Software.** Primera edición*. Addison Wesley. España, 2000.*
- *Jain, Anil K.* Fundamentals of Digital Image Processing. *Prentice Hall. 1989.*
- *Kruglinski, David J*. Programación **Avanzada con Visual C++ versión 6.0**. *Mac Graw Hill.*
- McConnell, Steve. **Desarrollo y Gestión de Proyectos Informáticos.** Primera Edición. *McGraw-Hill. España, 1997.*
- Pressman, Roger. **Ingeniería del software. Un enfoque práctico.** Cuarta edición. *McGraw Hill. España, 1998.*
- *Maidment, David R*. **Hidrología aplicada**. *Mc Graw Hill*
- Microsoft. MSDN library – january 2001

http://www.profc.udec.cl/~gabriel/tutoriales/rsnote/contents.htm
http://www.netnam.vn/unescocourse/computervision/computer.htm
http://ciips.ee.uwa.edu.au/~williams/research/presentations/colours/
http://www.dai.ed.ac.uk/HIPR2/wksheets.htm
http://www.khoral.com/contrib/contrib/dip2001/index.html

TABLA DE CONTENIDO

TABLA DE ECUACIONES

TABLA DE FIGURAS