UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO.

FACULTAD DE INGENIERÍA.

Diseño e implementación de una interfaz de usuario para un sistema de imagenología ultrasónica.

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

INGENIERO EN COMPUTACIÓN

P R E S E N T A

MARTÍNEZ ROSAS ARTURO.

Director de Tesis: **Dr. Fabián García Nocetti.**

CIUDAD UNIVERSITARIA **2003**

AGRADECIMIENTOS

Esta es una modesta forma de decir gracias a quienes me ayudaron a conseguir un logro tan importante en mi vida:

A la Universidad Nacional Autónoma de México, a la que agradezco mi formación profesional y el orgullo de ser integrante de su grandiosa comunidad.

A la Facultad de Ingeniería por su gentil regazo en el que recibí la formación y valores que han de regir mi vida profesional.

Al Instituto de Investigaciones en Matemáticas Aplicadas y Sistemas al Departamento de Ingeniería en Sistemas de Computo y Automatización por las facilidades prestadas para la realización de este trabajo de tesis.

A CONACYT (Proyectos 37913-A y REDII 7350-858) y DGAPA (Proyecto PAPIIT-117999), por el apoyo en la realización de este proyecto.

Al Dr. Fabián Garcia Nocetti por la confianza brindada al entregarme este proyecto y por el esfuerzo que junto conmigo dedicó a este trabajo.

Al Ing. Luis Alberto Aguilar Beltrán porque siempre creyó que yo podía realizar esta tarea dignamente, y porque nunca me dejó sólo.

Al Ing. Eliseo Díaz por su profesional labor en este proyecto.

Al M en C. Alejandro Sotomayor Ortega y al Ing. Martín Fuentes Cruz por sus consejos y apoyo, además por su trabajo previo que fueron el punto de partida de este proyecto.

Dedico este trabajo a mis padres, quienes han sido mi más grande ejemplo de esfuerzo y perseverancia, quienes con amor y esperanza me dirigieron a perseguir mis sueños.

A mis amados hermanos que son mi orgullo y una de mis más vivas esperanzas (Rafael, Fernando, Jesús, Hugo).

A todos los que soñaron conmigo este momento.

Índice

D

Capitulo IV.- Implementación

Capitulo V.- Resultados

Capitulo VI.- Conclusiones

Capítulo I
Introducción

1.1 –Introducción

1.2 –Objetivos

1.3-Contenido

F

1.1-Introducción

En los últimos años, las técnicas utilizadas para la formación de imágenes, tales como rayos x, resonancia magnética, ultrasonido, entre otros, han influido radicalmente en la práctica de la medicina y se han convertido en una herramienta esencial para el diagnóstico y la práctica de la medicina contemporánea. Desde el punto de vista del diagnóstico, un gran número de enfermedades puede ser detectado eficientemente con los métodos existentes y la información provista por los mismos puede ser de gran utilidad para el tratamiento del padecimiento detectado. Por otra parte las técnicas asociadas con la Imagenología son ya ampliamente utilizadas durante las intervenciones quirúrgicas, ya que permiten a los cirujanos monitorear en tiempo real las condiciones del paciente, mediante imágenes 2-D como 3-D. [1]

Existen diferentes técnicas de tomografía computarizada, algunas usan rayos *x* y gama, otras resonancia magnética y otras ultrasonido. En el caso de los rayos *x* y rayos gama, el manejo inadecuado o un error en el manejo de los mismos representa importantes riesgos para el paciente y para el médico. En el caso de la resonancia magnética, el sistema de adquisición de datos posee un elevado costo. Estas desventajas son superadas por las técnicas basadas en ultrasonido, ya que además de ser técnicas no invasivas, ofrecen una mejor relación costo-beneficio y las imágenes obtenidas ofrecen la resolución suficiente para su utilización en el diagnóstico médico. [4]

El presente trabajo es parte de un proyecto en el área de Imagenología Ultrasónica, que se ha venido desarrollando en el Departamento de Ingeniería en Sistemas Computacionales y Automatización DISCA, perteneciente al Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas IIMAS. El proyecto consiste en el desarrollo de un tomógrafo ultrasónico basado en una arquitectura flexible y abierta. Se ha aprovechado la experiencia del Departamento en las areas de ultrasonido y arquitecturas paralelas, así como los resultados de investigaciones anteriores que representan el punto de partida de este trabajo. [2]

1.2-Objetivos

Este trabajo tiene como objetivo principal el diseño y desarrollo de una interfaz gráfica de usuario (GUI del Inglés Graphical User Interface) para un sistema de imagenología ultrasónica, que permita controlar los procesos de formación y despliegue de imágenes ultrasónicas de dos dimensiones en tiempo real, partiendo de señales generadas por un arreglo lineal de sensores ultrasónicos, las cuales son acondicionadas y procesadas en una plataforma de cómputo paralelo basada en procesadores digitales de señales (DSP's del inglés Digital Signal Processors) antes de ser manejadas por la GUI propuesta.

La GUI ofrece un ambiente amigable que permite definir parámetros de adquisición, procesamiento y despliegue. El diseño y desarrollo de la GUI es compatible con las características de los subsistemas de adquisición y procesamiento de los datos que conformarán las imágenes que se desea desplegar, las cuales deben ser generadas a razón de 25 imágenes por segundo como mínimo.

1.3-Contenido

En el primer capítulo se presenta el origen y contexto de este trabajo, lo mismo que su objetivo central.

El segundo capítulo introduce conceptos generales de ultrasonido, imagenología, interfaces gráficas de usuario y una breve descripción del sistema de desarrollo con el fin de lograr una visión más clara del sistema, la interacción de sus módulos y el rumbo que ha de tomar el diseño de esta interfaz.

El capítulo tres parte de una descripción de todo el sistema mediante diagramas de bloques, hace una descripción general de los módulos que lo integran, para finalmente determinar las consideraciones necesarias para implementar la GUI.

El capítulo cuatro describe el contexto y la estructura de la implementación, después se presenta al núcleo de la aplicación en términos de código.

En el capítulo cinco es una descripción de la interfaz lograda, esta se presenta en dos planos, funcionalidad y desempeño

El capítulo seis resume las conclusiones y se habla de trabajo futuro.

Los apéndices contienen básicamente los listados de todos los programas.

Capítulo II
Generalidades

2.1- Ultrasonido

Una onda de ultrasonido al igual que una onda de sonido son perturbaciones mecánicas de algún medio (gaseoso, liquido o sólido), las cuales pasan a través del medio a una velocidad fija. Las ondas de sonido producidas cuando hablamos consisten de perturbaciones de las moléculas de aire, la vibración pasa de molécula a molécula, desde la persona que esta hablando hasta el oído de quien escucha, no es que las moléculas viajen desde quien habla hasta quien oye, mas bien es un efecto en cadena. Cuando la frecuencia de estas vibraciones alcanza los 20 Khz. deja de ser audible para los seres humanos, es por eso que recibe el nombre de ultrasonido.

En medicina estas perturbaciones son caracterizadas por un cambio de presión local o por la amplitud del desplazamiento de las particulas desde su posición de reposo, este fenómeno mecánico puede ser detectado y convertido en señales eléctricas por un transductor que actúa como transmisor y receptor de las señales ultrasónicas. [15]

La velocidad de propagación de las ondas a través de un medio, depende de la compresibilidad y densidad del mismo. En general el comportamiento de las señales ultrasónicas a través de un material determinado, depende de sus caracteristicas fisicas. Por ejemplo cuando las ondas inciden sobre materiales distintos, en la frontera entres estos dos materiales ocurre un fenómeno importante, algunas ondas pasan y algunas son reflejadas, lo mismo ocurre cuando la luz incide sobre un cristal. El grado de reflexión depende otra vez de las propiedades fisicas de los materiales (densidad y compresibilidad). Cuando los dos materiales que hacen frontera tienen caracteristicas similares (grasa y músculo por ejemplo) el grado de reflexión es pequeño. Esta caracteristica es utilizada por el instrumental ultrasónico, el cual se mantiene periódicamente enviando pulsos de señales ultrasónicas que son recibidos por el mismo transductor después de la reflexión.

El tiempo de llegada del eco (señal reflejada) depende de la profundidad del medio, y nuestro instrumento puede usarlo como un indicador de profundidad. De esta forma el instrumento recibe un eco y es posible determinar la profundidad en función del tiempo de llegada. Por tanto la orientación y el tiempo de arribo de la señal ultrasónica permiten la ubicación y el despliegue del objeto de interés. La amplitud del eco esta determinada por la estructura y composición fisica del reflector, por lo que tiene importancia para el diagnóstico y se usa para determinar el brillo con el que va a desplegarse la imagen creada. [1,15]

Otro fenómeno importantísimo es el que se origina debido al movimiento de los tejidos que reflejan las señales ultrasónicas, el cual modifica la frecuencia de las señales recibidas. Este fenómeno es conocido como efecto **Doppler** y el cambio en la frecuencia (Doppler shift) es proporcional ala velocidad de los tejidos reflectores. [7]

Basados en la información anterior, existen tres modelos para la creación de imágenes a partir de señales ultrasónicas.

2.2- A-SCAN

El instrumento más simple basado en el eco de pulsos es el de tipo A-SCAN, el cual despliega en una gráfica, la amplitud del eco recibido, contra tiempo. [15] La profundidad es determinada por el tiempo de llegada del eco. La figura 2.1 ilustra estas ideas.

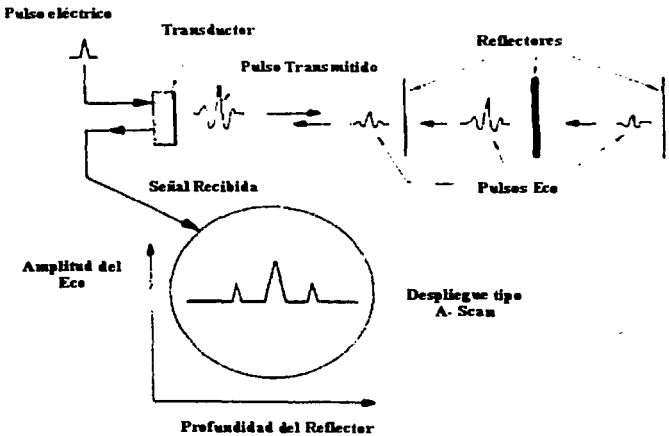

Figura 2.1-Modelo B Scan

2.3- B-SCAN

En este tipo de instrumento es posible generar imágenes bidimensionales. La señal de eco recibida en cada transductor, es desplegada como un conjunto de puntos cuya intensidad depende de la amplitud del pulso, su posición esta determinada por la orientación de la señal ultrasónica y tiempo de arribo de los ecos. [15] La figura 2.2 muestra el tipo de imágenes B-Scan.

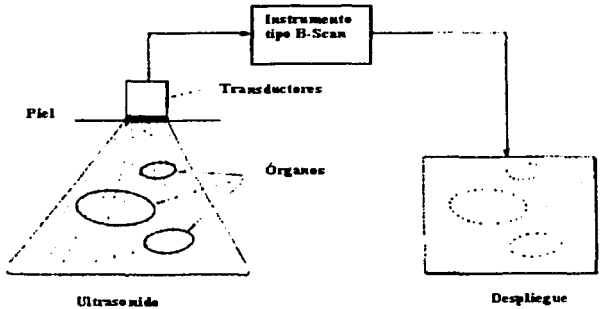

Figura 2.2- Modelo B-Scan

2.4 - M-MODE

El movimiento de los tejidos que generan los ecos de las señales ultrasónicas, pueden desplegarse como función del tiempo, a esto se le conoce como M-MODE. [15] El cambio en la profundidad con relación al tiempo es desplegado como se muestra en la figura 2.3. Esto permite medir la dinámica de las válvulas del corazón.

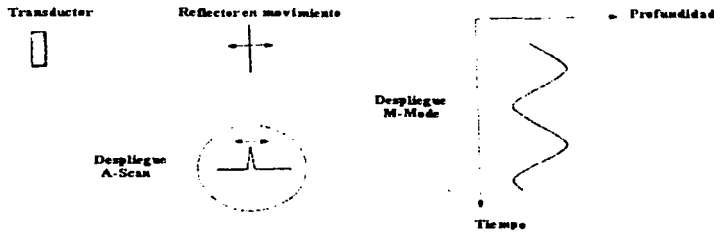

Figura 2.3- M-Mode

2.5-IMAGENOLOGÍA ULTRASÓNICA

El uso del ultrasonido en la formación de imágenes médicas, ha sido utilizado desde principios de los años 50. [5] Desde entonces los avances tecnológicos y las nuevas prácticas clínicas han convertido a la imagenología ultrasónica en una importante modalidad de diagnóstico. Los equipos modernos de ultrasonido tienen un costo relativamente elevado y una arquitectura rígida, tienen una apariencia como la de la figura 2.4.

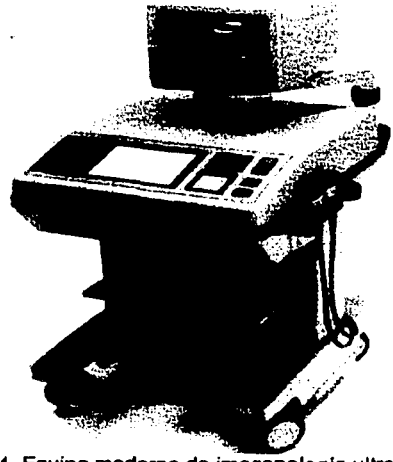

Figura 2.4- Equipo moderno de imagenología ultrasónica

La relación costo/beneficio es particularmente importante en los lugares de escasos recursos, donde se busca la manera de proporcionar sofisticadas imágenes médicas en forma oportuna para el diagnóstico. Actualmente la imagenología ultrasónica se usa en una gran variedad de aplicaciones clínicas como cardiología, obstetricia, ginecología, imagenología abdominal, etc. La figura 2.5 muestra una imagen representativa de lo que podemos ver en los equipos de imagenología ultrasónica. Este tipo de imagen es conocida como "B-mode" del inglés "Brightness mode".

Figura 2.5-.Imagen ultrasónica del corazón en modo B.

El área obscura que se encuentra prácticamente al centro de la parte superior es la sangre contenida en el ventrículo izquierdo.

En la imagen 2.5, el ventrículo izquierdo aparece en la parte superior de la imagen con el atrio izquierdo en la parte inferior derecha. El ventrículo y atrio derecho aparecen en el lado izquierdo de la imagen. En un sistema de tiempo real, veríamos el corazón contrayéndose, las válvulas separando los ventrículos y el atrio abriéndose y cerrándose. [3]

2.5.1- Principios básicos

Prácticamente todos los aparatos de imagenología ultrasónica que se usan en aplicaciones clínicas hoy día, realizan la formación de imágenes basados en el eco. Una onda acústica es emitida sobre el cuerpo usando un arreglo de transductores, la onda interactúa con el tejido y la sangre y parte de la energía transmitida regresa al transductor para ser detectada. Si conocemos la velocidad de propagación en el tejido que estamos analizando, podemos determinar la distancia que existe desde el transductor hasta el lugar donde la interacción ocurrió. Las características de la señal eco (amplitud, fase,etc.) proveen información acerca de la naturaleza de la interacción, por ello nos permite conocer las características del medio en el que ocurrió. [7]

El estudio de ultrasonido en el diagnóstico comienza con el estudio de los principios físicos involucrados. La mayor parte de la imagenología ultrasónica puede describirse a través del análisis de ondas compresionales y su interacción con el tejido y la sangre. Estas ondas pueden producirse fácilmente y emitirse hacia el tejido, excitando un transductor ultrasónico, después la energía de esa onda es reflejada, absorbida y dispersada por el tejido. Parte de la energía que es dispersada o reflejada puede detectarse con un transductor ultrasónico (generalmente el mismo que sirvió para generar la señal inicial). La señal recibida representa la interacción de la onda acústica con el medio biológico. Para entender como se forma una imagen, es necesario conocer la velocidad de propagación en los tejidos, la atenuación de la onda acústica y que es exactamente lo que representa la señal que regresa. [15]

Las ondas compresionales se propagan a través del medio con velocidades especificas. La velocidad de propagación en los tejidos depende del tipo de tejido, la temperatura y la presión. Normalmente se asumen condiciones estándar para la temperatura y presión corporal, asi que sólo se considera variante el tipo de tejido que es lo que nos interesa.
La siguiente tabla muestra las propiedades acústicas de diferentes medios.

Tabla de propiedades acústicas

Medio	Velocidad del sonido [m/seg]	Impedancia 10^6 kg/m*m*s	Atenuación dB/cm a 1MHz
Aire	344	0.004	12.0
Agua	1480	1.48	.0025
Grasa	1410	1.38	0.63
Músculo	1566	1.70	1.3-3.3
Hígado	1540	1.65	0.94
Hueso	4080	7.80	20.0

Los valores de la tabla son representativos y se encuentran en la literatura con pequeñas variaciones. La velocidad en el tejido humano es considerada usualmente como 1540m/s. [5,15]

Debido a que la energía de la señal generada inicialmente es absorbida, dispersada y reflejada mientras viaja a través de los tejidos, la señal se atenúa más y más conforme se adentra más en el cuerpo. La atenuación ocurre por diferentes mecanismos, pero se debe primordialmente a la absorción y dispersión. La atenuación es una función exponencial de la distancia, frecuentemente modelada como $A(x)=Ao\ e^{-\alpha x}$ donde A es la amplitud, Ao es una constante, α es el coeficiente de atenuación(depende de la frecuencia), y x es la distancia. Existe un componente de atenuación que no depende de la frecuencia pero generalmente es despreciable. [15]

Este modelo de atenuación tiene algunas implicaciones para los sistemas de imagenología, por ejemplo la atenuación en el tejido es alrededor de 0.75 dB/cm/Mhz en un solo sentido, así la intensidad disminuye a la mitad cada 0.8 cm a 5Mhz.

La energía acústica que regresa al transductor, lo hace principalmente por dos mecanismos: reflexión especular y dispersión. La reflexión especular se debe al cambio de impedancia acústica en los materiales. De la misma forma que ocurre con las ondas electromagnéticas en una línea de transmisión, cuando una onda acústica se mueve de un medio con impedancia Zo a un medio con impedancia Z1, parte de la onda será reflejada y parte será transmitida.[1]

La dispersión ocurre cuando la onda acústica interactúa con partículas de tamaño comparable con su longitud de onda. Estas partículas o estructuras tienden a generar pequeñas y débiles señales en todas direcciones. Un determinado volumen de dispersores (como células sanguíneas o tejido orgánico) actúa como un reflector difuso.

2.5.2- Sistemas de Imagenología ultrasónica

Los sistemas de imagenología modernos tienen una estructura genérica que contiene elementos básicos que se encuentran presentes en cualquier sistema y que pueden describirse mediante un diagrama de bloques como en la figura 2.6. [5,7]

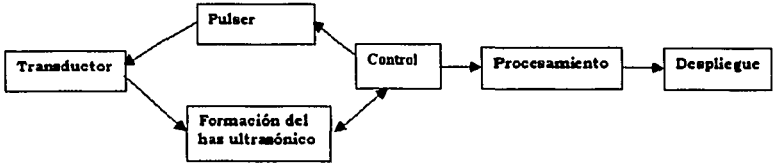

Figura 2.6-Diagrama de bloques de un sistema de imagenología ultrasónica

2.5.3- Pulser

El pulser genera las señales eléctricas que excitan a los elementos del arreglo de transductores. La intensidad y el contenido espectral de la señal acústica emitida desde el transductor puede controlarse variando la frecuencia central, la forma, duración y amplitud de la señal que genera el pulser. Los equipos típicos de

imagenología usan una frecuencia central de entre 2 y 10 Mhz. La amplitud de la señal eléctrica varia frecuentemente entre 2 y 200 volts.

2.5.4- Transductores

La mayoría de los transductores están constituidos por uno o más elementos piezoeléctricos con una capa protectora en la superficie frontal y en la parte posterior. Las cerámicas piezoeléctricas son las más comunes.

El transductor más simple es el que esta formado por un solo elemento, el cual es usualmente circular y tiene curvatura especial para el enfoque. Este elemento se desplaza manualmente para colectar información suficiente para formar una imagen. La principal ventaja es la simplicidad, pero tiene desventajas como el hecho de tener el foco fijo. Esta es una tecnología obsoleta que prácticamente ya no se usa. La tecnología dominante hoy día es la que trabaja con arreglos de muchos elementos (usualmente entre 48 y 200) que transmiten y reciben señales acústicas. [12] La figura 2.7 muestra el aspecto de algunos transductores comerciales.

Figura 2.7. Formas típicas de transductores comerciales

Existen dos tipos importantes dentro de esta variedad: los arreglos lineales y los arreglos de barrido sectorial. Los arreglos lineales disparan grupos de cerámicas directamente sobre la región de interés, el enfoque en transmisión y recepción se hace por medio de retardos, de esta forma se aproxima un frente de onda cóncavo tal como se muestra en la figura 2.8.

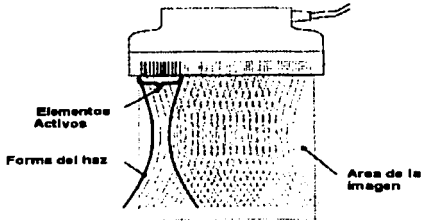

Figura 2.8- Arreglo lineal

La figura 2.9 muestra el arreglo de barrido sectorial, normalmente conocido como *phased array*, tiene un número menor de cerámicas, y se usan todas siempre, tanto en transmisión como en recepción. La forma del área que cubre es distinta y generalmente es más amplia. [18]

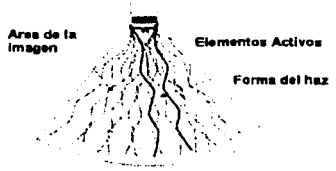

Figura 2.9 – Arreglo de barrido Sectorial

Los arreglos lineales pueden tener suficientes elementos para cubrir la región de interés, pero puede obtenerse una área mayor con un arreglo cuyos elementos se coloquen sobre una superficie convexa. [12,18] La forma de enfoque y barrido es igual que el arreglo lineal, pero usa menos cerámicas. El principio se ilustra en la figura 2.10.

Figura 2.10- Arreglo Convexo

2.5.5-Formación del haz ultrasónico

La formación del haz ultrasónico es una de las partes más importantes de un sistema de imagenología ultrasónica, en ella intervienen múltiples factores que determinan las características del haz ultrasónico emitido desde el arreglo de transductores, el enfoque de las imágenes generadas depende de la forma que se de al haz ultrasónico. [12,18] La ventaja más importante de los arreglos de múltiples elementos es la habilidad de dirigir o enfocar el haz electrónicamente. La figura 2.11 ilustra el enfoque electrónico, para ello se generan retardos en los pulsos que excitan para cada uno de los elementos del arreglo, esto permite que la dirección y profundidad del enfoque pueda cambiar.

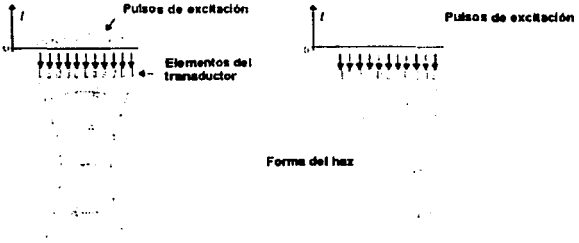

Figura 2.11.Formación del haz

Normalmente se escoge un punto focal para una serie de emisiones y se obtiene un enfoque continuo en la recepción. Una forma de tener más de un punto focal para la transmisión es usar un multiplexor en tiempo o técnica "multizona". Se realizan emisiones para diferentes profundidades, contenidas en la vecindad de la zona focal anterior y se colecta la información entre emisiones.

Las señales eco de cada uno de los elementos normalmente pasa por un amplificador TGC (Time Gain Compensation). El propósito de este amplificados es básicamente compensar las perdidas por atenuación que dependen de la profundidad. Este amplificador es controlado por el usuario a través de una serie de controles manuales. La recepción y el enfoque en los sistemas comerciales es del tipo de retardo y sumas, con algunas variaciones como fase-retardo vs. tiempo de retardo real o algunas otras técnicas en el dominio de la frecuencia que no han sido ampliamente aceptadas. [5]

2.5.6-Control

Este bloque se encarga de coordinar interacción entre transmisión y recepción. Cabe mencionar que puede haber enfoque en la transmisión, recepción o en ambos, lo que muestra la necesidad de coordinar el pulser y la formación del haz ultrasónico.

2.5.7-Procesamiento

Cuando la recepción y el enfoque terminan y se conforma la imagen, esta es sometida a diferentes operaciones de procesamiento antes de presentarla en el subsistema de despliegue con la finalidad de producir una imagen de mayor calidad. La mayoría de los equipos modernos realizan estas funciones digitalmente.

2.5.8-Despliegue

La función del despliegue es principalmente es mostrar las imágenes ultrasónicas construidas. Para ello es necesario utilizar una escala de colores o bien una escala de niveles de gris y darle a la imagen un formato adecuado para despliegue. Los datos acústicos casi nunca son colectados en la misma densidad que el numero de píxeles desplegados, generalmente se utiliza algún tipo de interpolación para escalar las imágenes.

2.6-Interfaces de Usuario

El diseño de interfaces de usuario (IU) es ya considerado toda una disciplina, que se auxilia de diversos campos como las ciencias de la computación, el diseño industrial, la psicología cognoscitiva, el diseño audiovisual etc. Por ello en esta sección nos limitamos a presentar sólo lineamientos generales sobre las interfaces graficas de usuarios (GUI), aquellos que son ampliamente aceptados no sólo por los que han desarrollado teorias sobre IU, si no por todos los que usamos frecuentemente una computadora.

2.6.1-Origen de las GUI

La computadora es un conjunto de circuitos digitales que realizan operaciones aritméticas y lógicas en base dos. Posee la trascendente caracteristica de realizarlas con gran velocidad y poder almacenar datos y resultados, es todo lo que sabe hacer, lo cual no resulta muy significativo para quien la computadora es más un medio que un fin. Las computadoras y su software funcionan basadas en un enorme e invisible sistema que provee pocos elementos que indiquen el estado o la organización del sistema. El propósito de la GUI es proveer en la pantalla de la computadora elementos que creen un ambiente operativo para el usuario, formando un contexto visual y funcional explicito para las acciones del usuario.

Desde sus primeros días resulto evidente la necesidad de crear una interfaz que disminuyera la complejidad de trabajar con la máquina y le permitiera al usuario concentrarse más en aquella tarea especifica que le interesaba lograr con ayuda de la computadora.

A medida que evolucionaron Hardware y Software, las aplicaciones se volvieron cada vez más genéricas; con ello también evolucionaron las interfaces entre la máquina y el usuario. Con la aparición de los sistemas gráficos de ventanas, la interfaz de usuario tomó otro rumbo. Hoy día la mayoría de las aplicaciones cuentan con una interfaz grafica de usuario, compuesta por objetos estándar como botones, iconos, campos de texto, ventanas, menús, etc, mismos que deben presentar una estructura funcional clara y bien definida acorde con el propósito del software. La GUI dirige las experiencias del usuario y hace visible la estructura organizacional del sistema computacional.

2.6.2- Metáforas

Después de algunos años de experiencia con sistemas complejos y abstractos se empezó a construir un modelo conceptual que modelaba a la computadora de la manera que el usuario se la imaginaba, es decir la estructura y organización que el usuario imaginaba. Este modelo permitió a los usuarios entender y predecir el comportamiento del sistema sin tener que memorizar reglas abstractas y arbitrarias. Así pues el objetivo principal del diseño de IU es crear un modelo mental coherente y apropiado para las operaciones y la organización del sistema computacional. Las GUI incorporan metáforas visuales y funcionales del mundo cotidiano para orientar al usuario acerca de las funciones y posibilidades del sistema computacional.

Las funciones del sistema computacional se hacen visibles y se colocan en un contexto lógico y predecible para el usuario, emulando la apariencia y el comportamiento de objetos cotidianos como folders, documentos de papel, herramientas, botes de basura etc. Una de las más famosas e imitadas metáforas es la interfaz de escritorio, creada en Xerox Palo Alto Research Center en 1970. [8] Los diseñadores pensaron que como esas máquinas serían usadas en un ambiente de oficina, la emulación de objetos típicos de una oficina las haría más fáciles de entender. Estas fueron las primeras computadoras que utilizaron iconos gráficos que representaban documentos, directorios, papeleras, buzones de correo etc. [8]

Las metáforas de las interfaces permiten un proceso experimental que algunos especialistas llaman "reactive cognition", lo cual no es más que obtener experiencia sobre la funcionalidad del sistema en la medida que se interactúa con los objetos de la interfaz. Así no es necesario memorizar comandos, más bien se reacciona ante elementos gráficos que tratan de ser más descriptivos y puntuales al mismo tiempo a través de la metáfora, por ejemplo cuando tiramos algo que no nos sirve generalmente lo ponemos en un cesto de basura, para guardar documentos los ponemos en un fólder. Sin embargo agregar elementos gráficos y un mouse a una interfaz, no significa que el sistema sea más fácil de usar o entender. Durante el proceso de evolución, los sistemas computacionales adquieren nuevas capacidades, pero muchos usuarios se quejan por la complejidad funcional y visual de las GUI.

A pesar de que las metáforas son ampliamente aceptadas, a menudo son pobremente implementadas y producen software difícil de entender y usar. Estas dificultades de diseño generalmente se deben a dos problemas: inconsistencia o ambigua relación entre los objetos de la interfaz y un pobre diseño visual. Las metáforas exitosas en el diseño de IU deben ser tan simples que no requieran que el usuario aprenda o recuerde reglas y procedimientos. Si el usuario es forzado a recordar reglas o procedimientos la metáfora se pierde pues la idea es que sea muy evidente, por ejemplo cuando ponemos el icono de un documento en un fólder, al abrir ese fólder esperamos ver el documento ahí, y naturalmente asumimos que permanecerá ahí hasta que decidamos moverlo a cualquier otro

fólder o borrarlo, de la misma forma que hacemos con los folders reales. Si alguna de estas caracteristicas no fuera soportada consistentemente por la GUI el concepto del fólder como una metáfora organizacional carecería de sentido. Las GUI más exitosas se basan en los conocimientos y las experiencias cotidianas de los usuarios y no en convenciones establecidas que permitan al usuario predecir el resultado de sus acciones.

2.6.3-Control

El usuario debe sentir siempre que tiene control directo sobre la aplicación, y no sentir que la computadora realiza acciones por cuenta propia que incluso obliguen al usuario a desperdiciar tiempo. Las buenas interfaces también deben de prever los errores del usuario y tener suficiente estabilidad para no colapsar ante datos inapropiados o intentos de acciones que puedan dañar el sistema y sus datos.

2.7-Consideraciones de diseño

Existen diferentes metodologías y diferentes enfoques para el desarrollo de interfaces gráficas de usuario. A continuación se presentan los lineamientos más generales en los que coinciden la mayoría de estas aproximaciones.

2.7.1-Conceptual

Una imagen mental simple (metáfora), adecuada organización de los datos, funciones, actividades etc, un modelo eficiente de navegación entre datos y funciones , una apariencia de calidad y una adecuada secuencia de interacción con el usuario.

2.7.2-Factores de desarrollo, facilidad de uso, y aceptación

Factores de desarrollo : limitantes asociadas a la plataforma de desarrollo, las herramientas de desarrollo, sus componentes y librerías, soporte para implementar prototipos rápidamente y la flexibilidad de todas estas herramientas.

Factores de uso: las habilidades del usuario, un modelo funcional claro.

Factores de aceptación: políticas corporativas o institucionales, condiciones y amplitud de los mercados o en su caso la comunidad a la que esta dirigida, documentación y capacitación.

2.7.3-Estructura visual

Lo más importante aquí es lograr una presentación consistente, intuitiva, acorde con el concepto funcional, no abrumadora.

El diseño grafico pude ser de gran ayuda para mejorar la capacidad de comunicación de la GUI con el uso adecuado de tipografía, símbolos, color, etc. El color, contraste e iluminación pueden hacer más agradable la interfaz y darle una apariencia más simple. La disposición de los elementos gráficos, formatos, proporciones, organización dimencional es importantísima para no espantar confundir o abrumar al usuario. Son básicamente tres los principios que podemos destacar de todas estas consideraciones y que pueden servir como guía elemental para el diseño de una GUI:

Organizar: proveer al usuario de una estructura conceptual clara y consistente.
Economizar: Maximizar espacio y eficiencia, minimizar controles.
Comunicar: Satisfacer las expectativas y necesidades del usuario.

2.8-Sistema de desarrollo

La plataforma computacional que genera los datos esta basada en la arquitectura DSP-SHARC de Analog Devices, la cual esta compuesta por una tarjeta motherboard-PCI (ASP-15), módulos DSP de procesamiento y módulos de entrada-salida para controlar el arreglo de sensores ultrasónicos y capturar las señales tipo pulso-eco que constituyen la base para generar la imagen 2-D de interés. Esta arquitectura recibe parámetros y entrega los datos procesados a la GUI en la computadora host, la cual se encarga de formar y desplegar la imagen en el monitor de la PC utilizada como host.

Resolver un problema de procesamiento digital de señales implica un minucioso análisis para determinar la capacidad de procesamiento y comunicación del sistema en cada uno de sus módulos y por tanto en todo su conjunto, así como las bondades en términos de hardware de las arquitecturas utilizadas para la implementación de algoritmos paralelos. Los investigadores del DISCA-IIMAS han trabajado en este tipo de actividades durante varios años y han realizado ya algunas implementaciones para este tipo de arquitectura. Así pues, son ellos quienes se han encargado de este minucioso análisis en el que determinaron que la arquitectura SHARC en conjunto con la tarjeta ASP-15, representan una valiosa oportunidad para el adquirir, procesar y desplegar imágenes ultrasónicas en tiempo real. [2]

El ADSP-2106x SHARC—Super Harvard Architecture Computer—es un DSP (Digital Signal Processor)de 32 bits, perteneciente a la familia ADSP-21000 de Analog Devices. Este DSP esta constituido por un sistema completo en el mismo chip, incluye un bloque de memoria SRAM de doble puerto y periféricos de e/s soportados por un bus dedicado de e/s. El chip incluye memoria cache de instrucciones, con la cual puede ejecutar cada instrucción en un solo ciclo de reloj. El ADSP-21062 incluye un DSP de alto desempeño de punto flotante, un procesador que sirve de interface con el host, un controlador de DMA, puertos seriales, links de comunicación entre DSPs y la conectividad para compartir el bus

en un sistema de multiprocesamiento. [20] La figura 2.12 muestra las características de la arquitectura ADSP-21062

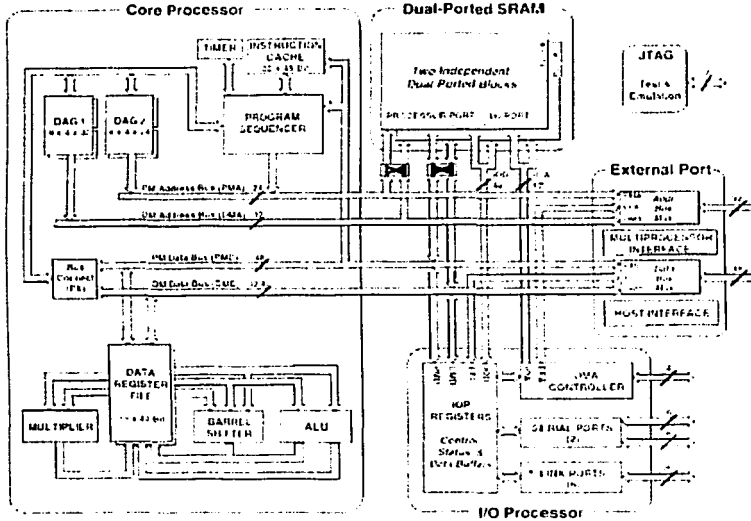

Figura 2.12- Diagrama de bloques del SHARC ADSP-21062

Cuando se trabaja con aplicaciones de tiempo real, es importante analizar varios factores dependientes de la arquitectura del DSP con que vamos a trabajar. Primero el ancho de banda de los buses del procesador, segundo la latencia de distribución de los datos a través del sistema y el desempeño de entrada y salida en los puertos del SHARC

El SHARC esta diseñado para formar clusters en grupos hasta de 6 procesadores, cuando esto ocurre, los procesadores utilizan buses paralelos externos al chip de cada DSP. Este bus es usado para comunicación entre procesos y para el acceso a memoria no incluida en los chips, que en este caso se encuentra alojada en la tarjeta ASP-15. Además tiene un segundo bus paralelo de datos de entrada/salida, es un bus interno al SHARC que envía y recibe datos a través de DMA y posee dos puertos seriales bidireccionales. [19]

Existen diferentes ambientes de desarrollo para construir aplicaciones para los DSP SHARC, se describe a continuación.

2.8.1-Bare Bone

Este tipo de ambientes consta de un compilador, un ensamblador, un emulador y un conjunto de librerías. Este tipo de ambiente, resulta apropiado para aplicaciones de bajo nivel, típicamente embebidas donde esta perfectamente identificada y definida la entrada y salida, así como el procesamiento requerido. [20]

2.8.2-Run Time

Menor que un sistema operativo, pero mas que el Bare Bone , provee un e/s más sofisticada y hacer procesamiento en background o foreground. [21]

2.8.3-Real Time Operating System (RTOS)

Cuando una aplicación se implementa para correr en múltiples procesadores y tiene e/s asincronas, el procesador debe ser manejado como un recurso. Un sistema operativo de tiempo real provee al sistema de herramientas de administración que permiten al desarrollador manejar el procesador definiendo prioridades en los procesos, scheduling y métodos de sincronización de procesos. Los RTOS proporcionan al desarrollador estadísticas para medir desempeño. [21]

2.8.4-Sistema de desarrollo unificados

Cuando una aplicación incluye varios procesadores de arquitecturas diferentes, se necesita un sistema de desarrollo un poco más complejo, que haga convivir DSPs y procesadores de propósito general. [19]

2.9-Ambiente de desarrollo

Escoger el ambiente de desarrollo también es una decisión difícil, fue DISCA quien determino el ambiente de desarrollo, basados en su experiencia y oportunidades.
Las características de nuestro sistema de desarrollo corresponden al tipo "Bare Bone". El desarrollo de software para las tarjetas del SHARC (ASP) implica programar una aplicación para el DSP y opcionalmente una aplicación sobre el host.
El corazón del soporte de programación para el SHARC es el Analog Devices Toolset para la familia ADSP-21000 de DSP's e incluye los siguiente:

Compilador de C
Ensamblador
Ligador
Cargador de red
Utilería para probar la red

Librería de C embebida
Servidor y librería de E/S para el host
Librerías de E/S para las tarjetas
Emulador del SHARC
Simulador del SHARC

El software de Transtech, permite desarrollar aplicaciones para las herramientas de Analog Devices, que corren en múltiples procesadores SHARC, incluyendo clusters de SHARCs. Estos cluster obviamente están implementados en la misma tarjeta ASP, pero también es posible hacer cluster donde cada nodo puede ser una tarjeta ASP con varios SHARCs. La entrada/salida desde las aplicaciones del SHARC, son posibles utilizando el servidor de E/S y las librerías provistas.

2.9.1-Utilerías de Transtech

Las utilerías para el host, permiten detectar hardware disponible y probarlo y permiten a las aplicaciones multiprocesadores, ser cargadas ("live"), desde booteo o, desde una memoria FLASH.

2.9.2-Librerías de soporte para las tarjetas

Las librerías para ASP se dividen en dos, librerías para el SHARC y librerías para el host. Las primeras se ligan con los ejecutables del Sharc y hay dos versiones, una para ADI tools 3.3 y otra para visual DSP. Las librerías para el host se ligan con las aplicaciones del host y se proveen en las formas mas comunes para las distintas plataformas de host. [19]

Sharc	Tops I/O library ASP-xxx Board Support Library
Host	Host Communication Library (HCL) Simple Communications Library (SCL)

2.9.3-TOPS I/O Library

Esta librería permite al Sharc raíz de la red, ejecutar en forma normal operaciones de e/s de C, a través de el TOP server corriendo en el host.

2.9.4-Simple Communications Library (SCL)

Provee servicios de comunicación y booteo para el Sharc, es utilizado por la TOPS library.

2.9.5-ASP Host Communications Library (HCL)

Esta librería permite hacer un DMA, utilizando el máximo el ancho de banda en las transferencias entre el Host y los Sharcs residentes sobre una tarjeta ASP-xxx, para esto es necesario que la tarjeta contenga un PLX9060 o un 9080 PCI bridge,el cual se incluye en todos los productos de Transtech, excepto en los que tienen interface ISA.

2.9.6- ASP-xxx Board Support Library

Su nombre es ASP-xxx BSL porque funciona en todos los miembros de la familia ASP-xxx (ASP-V15, ASP-P14, ASP-P15), que tarejeta usamos!! provee funciones para mapear el espacio de memoria PCI en el host, al espacio de memoria del SHARC, administra la Datatype Conversion Unit (DCU) y maneja las interrupciones de hardware. Al igual que la HCL, esta librería requiere la presencia de un chip PLX 9060 o 9080 PCI bridge.

Los chips PLX proveen dos canales DMA para la transferencia de datos entre el bus PCI y el bus del cluster de Sharcs. La habilidad del SHARC de mapear parte de su espacio de direcciones y después accesar al bus PCI, es llamada **Direct Master Access** . Y la habilidad del PCI para accesar al bus del cluster de Sharcs es llamada **Direct Slave Acces.** Todas las tarjetas extienden la interfaz PLX básica, con dos FIFO's externas y una unidad de conversión de tipos de datos (data type conversion unit DCU). Las FIFO's están situadas entre el PLX y el bus del cluster, conceptualmente en paralelo con los canales Direct Master y Direct Slave. La unidad de conversión (DCU) esta habilitada para accsesarse desde el Sharc y el PCI. Esto le permite al Sharc hacer accesos al bus PCI, de 8 y 16 bits en Direct Master mode y también le permite hacer cambios para el bigendian del host (ordenamiento e interpretación de bits). Las tarjetas ASP-XXX pueden mapear hasta 2Gbytes del espacio de memoria del Sharc al bus PCI, via Master Acces. [19]

Capítulo III
Diseño

3.1 -Diseño del sistema

3.2 -Adquisición

3.3 -Procesamiento

3.4 -Despliegue

3.1 Diseño del sistema

Este capítulo inicia con un diagrama de bloques del sistema y después describe con detalle sus componentes, de esta forma se determinan las consideraciones de diseño. El tomógrafo consta básicamente de tres etapas, adquisición, procesamiento y despliegue, como se muestra en la figura 3.1.

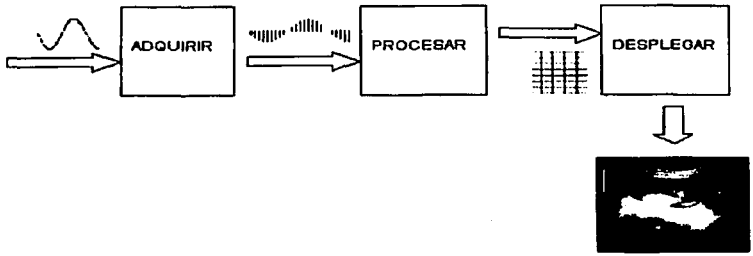

Figura 3.1 Diagrama general del sistema de imagenología

Los dos primeras etapas fueron construidas con base en las investigaciones y trabajos anteriores del DISCA. En la etapa de adquisición se generan, se transmiten y se reciben las señales ultrasónicas con las que trabajamos. La etapa de procesamiento consta básicamente de herramientas de procesamiento digital de señales implementadas para la arquitectura DSP-SHARC. La etapa de despliegue es la encargada de coordinar la interacción con el usuario, la modificación de los parámetros de adquisición y muestreo, así como el despliegue de la matriz de imagen. La secuencia de operación de estas etapas se realiza constantemente. Cada una de las mismas debe mantener un desempeño que evite el retraso de la información desplegada, en cuyo caso no visualizaríamos la imagen en tiempo real.

La figura 3.2 muestra con detalle los componentes del sistema y la interacción entre estos. La parte central del sistema es una tarjeta ASP-15 que contiene un *cluster* formado por cuatro SHARC 21062 en donde se realiza el procesamiento y *slots* de expansión para un módulo de adquisición de datos (A1). Esta tarjeta esta conectada a un sistema host tipo PC via bus PCI.

El A1 se mantiene generando señales de control para los disparos de las cerámicas y recibiendo la información correspondiente a los ecos de las señales ultrasónicas provenientes de la etapa Transmisión/Recepción .

Por su parte la aplicación host utiliza las librerías de soporte de la ASP-15 para comunicarse con la red de DSP's y obtener las imágenes ultrasónicas que serán desplegadas en el monitor de la PC por un módulo de OpenGL incluido en la aplicación host.

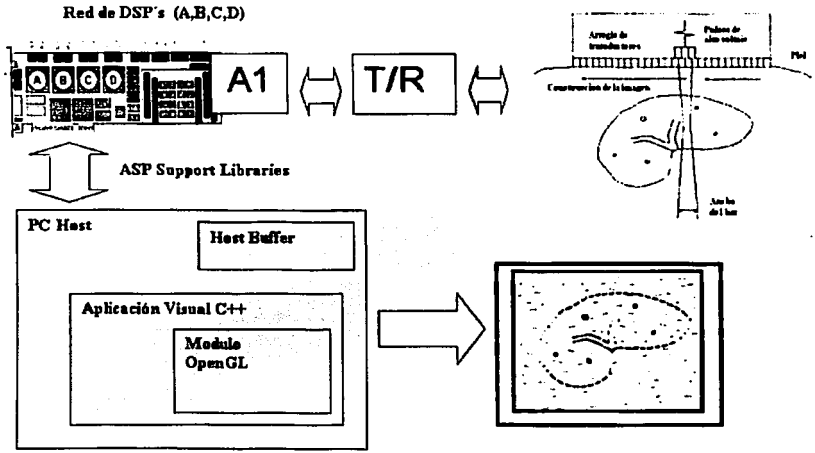

Figura 3.2 Diagrama del sistema

A continuación describiremos cada una de las etapas mostradas en la figura 3.1.

3.2 Adquisición

Las imágenes ultrasónicas que resultan significativas para el diágnostico se generan bajo diferentes condiciones que determinan las características de la imagen, su enfoque, profundidad o contraste por ejemplo. Estas condiciones están definidas por los parámetros de adquisición.

En esta etapa se definen los parámetros con los que se debe generar y transmitir la señal ultrasónica, así como los parámetros bajo los que se llevará a cabo la adquisición de los ecos. Así pues es necesario establecer algunos valores como la frecuencia de muestreo, el número de datos, etc; valores que serán manipulados a través de la interfaz de usuario, pero será aquí donde cumplirán su función.

Esta etapa se constituye por un transductor de arreglo lineal con 64 cerámicas, un dispositivo de electrónica analógica Transmisor/Receptor, una tarjeta con registros de corrimiento que funciona como control de disparo para las señales ultrasónicas y el modulo de adquisición A1, que se encuentra conectado a la tarjeta ASP-15 y envía toda la información adquirida a la red de DSP's (Ver figura 3.3).

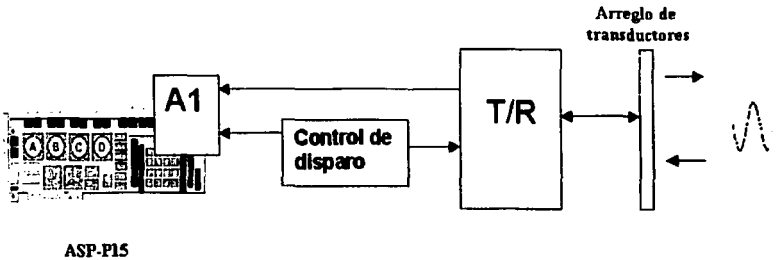

Figura 3.3 Diagrama del módulo de adquisición

El A1 esta basado en una tarjeta que contiene un SHARC DSP-21062, dos convertidores A/D de 8 bits y 4KB de memoria. Puede adquirir datos continuamente a una taza máxima de 60 MHz. El DSP incluido se encarga de manejar la conversión de datos y el manejo de buffers que contienen los datos muestreados, para la transmisión a los otros SHARC via link ports. Aquí se implementa todo el control para la comunicación transmisión y control de este modulo.
La tarjeta con registros de corrimiento es una implementación original del DISCA-IIMAS, se diseñó para controlar los disparos de las cerámicas que deben

realizarse en grupos de ocho cerámicas cada vez, los registros de corrimiento mueven el grupo de las 8 cerámicas que disparan, a través de todo el arreglo de transductores.

El dispositivo de T/R es un diseño analógico elaborado totalmente por el DISCA y el Instituto de Cibernética Matemáticas y Física de Cuba ICIMAF. Se encarga de generar los voltajes que excitán a los transductores para originar la señal ultrasónica, y de recibir los voltajes producto de los ecos recibidos. El arreglo de transductores esta formado por 64 cerámicas que funcionan como transmisor cuando son excitadas con un pulso eléctrico proveniente del T/R y como receptor de la señal eco.

En el A1 se generan las señales de de control y disparo para el arreglo de 64 cerámicas. Estas señales pasan por la tarjeta offset para decidir que cerámicas disparan. Después de esta tarjeta, dichas señales llegan al T/R, el cual genera pulsos de alto voltaje que excitan a las cerámicas del arreglo, lo que finalmente produce una señal ultrasónica. Después de esto el sistema pasa a un estado de recepción, en el que espera recibir el eco de la señal ultrasónica generada. El pulso correspondiente al eco se lleva nuevamente el T/R donde tiene un preprocesamiento analógico (filtrado y amplificación) para finalmente convertirlo en una señal analógica de 0 a 1 Volt, la cual se conecta a uno de los canales del A1 para hacer el muestreo. En el A1 se construye una columna de 250 elementos, con valores que dadas la características del convertidor A/D oscilan entre 0-255. Primero lee 250 datos para hacer la columna, una vez completada la columna la envía el arreglo de DSP en la ASP-P15, esto lo hace 56 (7x8) veces para formar la matriz de 250x56.

3.3 Procesamiento

Como se explicó anteriormente, esta tesis es parte de un proyecto de investigación que realizan en forma conjunta por el IIMAS y algunas otras instancias de investigación nacionales e internacionales. En esta etapa del proyecto uno de los objetivos más importantes es integrar las herramientas disponibles, tal es el caso de la plataforma de procesamiento digital, así como el resultado de otras investigaciones; el dispositivo de Transmisión/Recepción es un buen ejemplo. Así pues los algoritmos de procesamiento digital se encuentran en este momento en su fase de implementación y experimentación a cargo de diferentes investigadores cuya área de experiencia es justamente esa. Para fines de esta tesis un algoritmo sencillo como una interpolación sirve para probar el sistema en su totalidad, y proporciona resultados alentadores ya que las imágenes que se obtienen son de mediana calidad con un modesto procesamiento de por medio.

La funcionalidad del sistema de imaginología dependerá de las implementaciones de los algoritmos mencionados, pero para fines del objetivo central de esta tesis (la implementación de la interfaz de usuario) lo escueto del procesamiento en este momento no representa ningún problema si nuestro diseño e implementación es consistente con toda la plataforma de desarrollo, flexible y modular, eso permitirá en su momento la incorporación de nuevas herramientas de procesamiento digital y como consecuencia nuevas características funcionales para el usuario final. A continuación presentamos de forma breve y en términos muy generales, la estructura que guarda la etapa de procesamiento.

Una de las características más distintivas del DSP es la realización masiva de operaciones aritméticas. La construcción y procesamiento de una imagen ultrasónica requiere una capacidad computacional importante.

Una arquitectura tradicional (Vonn Newman) no satisface dichos requerimientos, por lo tanto es necesario trabajar con arquitecturas optimizadas para el cálculo numérico, estas arquitecturas reciben genéricamente el nombre de DSP's (Digital Signal Processor) los cuales permiten realizar operaciones aritméticas, transferencia de datos y acceso a memoria de forma óptima, además garantizan que el procesamiento de datos se mantendrá aún en las peores condiciones, bajo las condiciones mas intensivas de calculo del algoritmo y la tasa máxima de recepción de datos.

El procesador con el que contamos es un ADSP-21062 SHARC (Super Harvard Architecture Computer) de Analog Devices, que es un procesador digital de señales de alto desempeño, utilizado en el procesamiento de voz, sonido, gráficas e imágenes.

El SHARC es un procesador equipado para construir sistemas de multi-procesamiento. Tiene seis puertos de comunicación que transportan datos a tasas de hasta 40 Mbytes por segundo, que operan en modo half-duplex, esto es, los datos pueden transmitirse en ambas direcciones, pero no simultáneamente. La plataforma computacional que genera y procesa las señales ultrasónicas esta basada en la arquitectura DSP-SHARC, y esta compuesta por una tarjeta motherboard-PCI(ASP-15), módulos DSP de procesamiento y módulos de entrada-salida para controlar el arreglo de sensores ultrasónicos y capturar las señales tipo pulso-eco que constituyen la base para generar la imagen 2-D de interés

A continuación mostramos la estructura que guarda nuestra plataforma, en donde puede observarse claramente la línea de procesamiento.

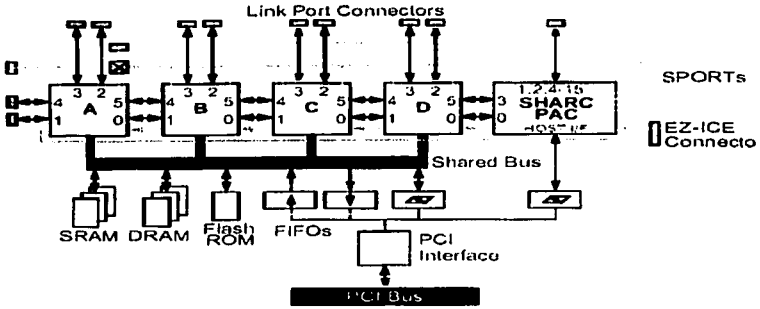

Figura 3.4 Diagrama de la tarjeta ASP-15 (plataforma de procesamiento)

La figura 3.4 muestra la estructura de la tarjeta ASP-15 de Transtech; A, B, C y D son las unidades de procesamiento y el SHARC PAC es el modulo de adquisición de datos.

3.4 Despliegue

La etapa final del tomógrafo, consiste en el despliegue de los datos calculados durante la etapa de procesamiento. Es necesario interpretar todo ese cúmulo de datos y construir la imagen que nos interesa. Esta interfaz debe cumplir básicamente con tres características, eficiencia, funcionalidad y sencillez.

La figura 3.5 ilustra el principo de funcionamiento del transductor que tenemos y la figura 3.6 lo que esperamos ver en el área cliente de una típica aplicación.

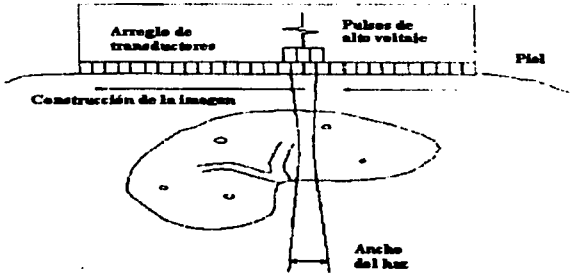

Figura 3.5 Principio de funcionamiento del arreglo de cerámicas

Nuestro transductor es un arreglo de 64 cerámicas que son utilizadas para generar la imagen ultrasónica. La figura 3.6 muestra la imagen que se genera barriendo el arreglo en forma lineal en grupos de 8.

Figura 3.6 Modelo de imagen esperada

Como se explicó antes, la aplicación host obtiene la imagen ultrasónica de la red de DSP's y se encarga de desplegarla en pantalla (ver Figura 3.2), esa aplicación va a ser construida con OpenGL, una API para hacer gráficos que tiene la virtud de ser independiente del sistema de ventanas, lo que le hace muy portable. Cada uno de los componentes de OpenGL ha sido cuidadosamente diseñado y optimizado por los profesionales de la industria de gráficos, SGI, esto implica que no es necesario sacrificar eficiencia por portabilidad, como sucede en otros casos.

Como OpenGL es independiente del sistema de ventanas, es necesario montar el núcleo de nuestra aplicación host, en otra capa de software que proporcione los elementos del ambiente gráfico (ventanas,menus,botones,etc...), lo que hará mas amigable nuestra interfaz. El software elegido para esta parte es Visual C++ ya que es un producto que pertenece a los propietarios del sistema operativo sobre el cual vamos a realizar el proyecto y por tanto proporciona compatibilidad con las herramientas de desarrollo. La figura 3.7 ilustra esta idea.

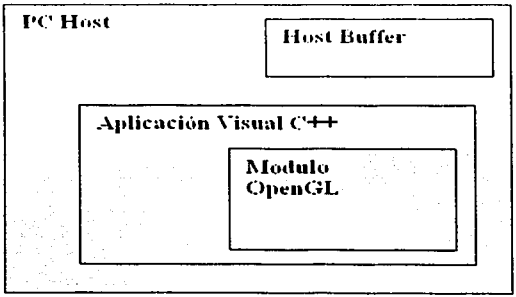

Figura 3.7 Diagrama de la aplicación host

Como en toda GUI, uno de los objetivos importantes es proporcionar un entorno de trabajo amigable y claro, parte de este objetivo se logra gracias al ambiente Windows que es ampliamente conocido.

Los datos que la aplicación host recibe son matrices de 250 x 56 cuyos elementos son números entre 0 y 255.Estos valores representan la amplitud del eco y serán interpretados por el host como niveles de gris en una escala de 256 niveles de gris. La aplicación host recibe un arreglo numérico como el de la figura 3.8.

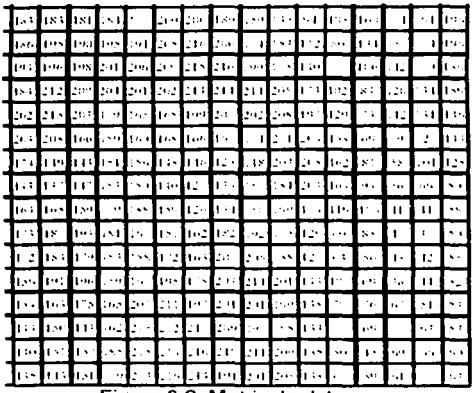

Figura 3.8 Matriz de datos

Esa matriz es convertida a valores RGB comprendidos entre 0 y 1 (son el tipo de valores que maneja glDrawPixels()).Aun que en el futuro se planea trabajar con color, por ahora trabajaremos con los tres valores RGB iguales para trabajar con una escala de grises, así que nuestra aplicación host debe convertir la matriz de datos en una imagen como la figura 3.9 en la pantalla de la PC.

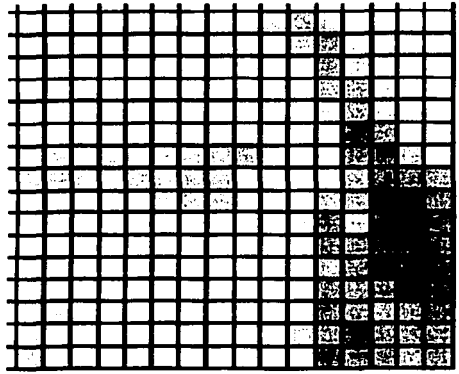

Figura 3.9 Matriz desplegada

Como se mencionó, antes la eficiencia de nuestra GUI es uno de los factores más importantes para el éxito del proyecto y por tanto una de las consideraciones de diseño fue la eficiencia de cada una de las herramientas de desarrollo
OpenGL proporciona uno de los conceptos más importantes en el desarrollo de nuestra GUI y lo hace de manera transparente para el programador lo que resulta aun más atractivo, este concepto es conocido como "doble buffer" (nos referiremos a estos buffers como son llamados en la literatura inglesa "front buffer" y "back buffer"). Este concepto es comúnmente utilizado en animaciones para evitar la sensación de parpadeo cuando los cambios de imagen no son suficientemente rápidos.

La figura 3.10 muestra el principio de funcionamiento del doble buffer, OpenGL opera en alguno de los dos buffers mientras el otro esta siendo desplegado, así no hay que esperar si es que el barrido vertical en la pantalla esta ocurriendo en ese momento para modificar la memoria de video. Puede parecer que con esto se tratará de mover enormes cantidades de datos, pero en realidad lo único que se hace es cambiar un puntero, por eso resulta eficiente. [13]

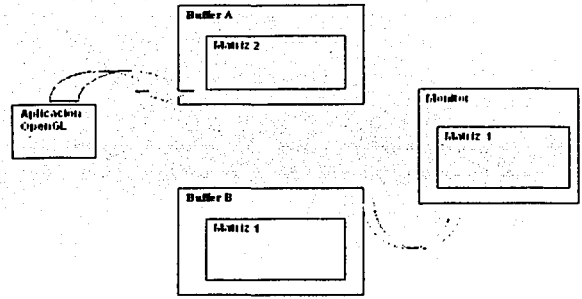

Figura 3.10 Doble buffer

Esto nos ayudara a eliminar la posibilidad de percibir la molesta sensación de parpadeo que se genera cuando los cuadros de una animación no se generan lo suficientemente rápido para que el ojo humano no perciba los cambios.

Capítulo IV
Implementación

4.1 –Descripción general

4.2 –Contexto de implementación

4.2.1 –Modelo de programación Windows

4.2.2 -Microsoft Foundation Classes

4.2.3 –OpenGL y las MFC

4.3 –Implementación de la GUI

4.1 Descripción general

El objetivo principal de toda interfase de usuario es coordinar la interacción entre el sistema y el usuario. Para ello la interfaz necesita controlar el funcionamiento de todos los programas que participan en el sistema de imagenología.

Nuestro sistema esta constituido por diferentes programas asociados a un elemento de hardware, por lo que ha sido necesario comunicar y coordinar los diferentes programas que se encuentran en cada modulo. En el diagrama de la figura 4.1 se pueden apreciar tanto los módulos de hardware del sistema como los elementos de software mapeados en cada uno de ellos.

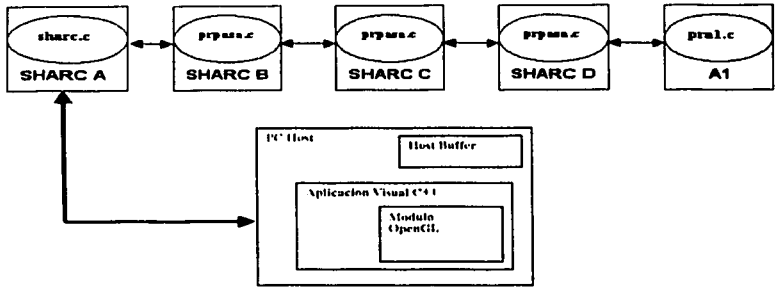

Figura 4.1 Programas en el sistema de imagenología ultrasónica

Los programas que se ejecutan en los procesadores B, C y D realizan la parte de procesamiento y el que se ejecuta en A1 controla la adquisición.

El SHARC A es el punto de comunicación entre la aplicación corriendo en la PC y la red de DSP's. El programa que corre en este procesador "sharc.c", recibe los datos provenientes del la línea de procesamiento y forma la matriz de imagen que deberá transmitirse al sistema host. Sin embargo como su programación es procedural, fue necesario encontrar un mecanismo de comunicación que nos permitiera actuar en función de las peticiones del host y permanecer en espera o apagado según el host lo requiriera. Este problema fue resuelto con un modelo de software conocido como *token passing*. *Token passing* es un esquema de control de software diseñado para coordinar la comunicación entre dos o mas entidades, permite a un programa establecerse como transmisor o receptor en un sistema dado.

En la figura 4.1 puede verse la representación de la PC y dentro de ella una aplicación visual C++ (GUI) que envuelve un pequeño modulo de OpenGL.

Cabe señalar que un objetivo de esta tesis es el despliegue de imágenes ultrasónicas en una aplicación Windows típica. Por lo tanto necesitamos crear una serie de programas con las herramientas disponibles, que puedan interactuar y desplegar imágenes en el monitor de la PC. Esto es un reto interesante por los diferentes paradigmas del software utilizado para la implementación, la API de Transtech, VC++ y OpenGL.

Inicialmente se necesita comunicar a la red de procesadores SHARC con la PC (Host), esto se logra utilizando librerías y utilerías que provee Transtech, específicamente la "*Host Communication Libray*" (HCL). Esta librería provee funciones de comunicación de alto nivel para hacer más fácil la codificación, y funciones de bajo nivel para optimizar la comunicación. Las rutinas de alto nivel, siguen un modelo de comunicación conocido como "*channel communications model*", en donde uno de los procesadores llama una rutina de envío (SendW) y el otro una de recepción (RecvW). [19] OpenGL se utiliza como núcleo de nuestra GUI para el despliegue y VC++ se encarga de todo el ambiente Windows, menús controles, mouse etc.

Por tanto el esquema general de implementación es: la API de Transtech resuelve la comunicación, OpenGL nos da el despliegue y VC++ envuelve toda la aplicación y se encarga de la interacción con el sistema operativo.

4.2 Contexto de Implementación

Con objeto de que se entienda mejor el núcleo de nuestra aplicación y su contexto, además que se justifique porque solo presentamos en este capitulo fragmentos de código, haremos un pequeño paréntesis para hablar de algunos aspectos importantes de la programación Windows, que son relevantes para esta implementación.

4.2.1 Modelo de programación Windows

La programación Windows tiene un modelo bastante peculiar, la naturaleza de las interfaces Windows hacen posible que los programas puedan tener diferentes entradas provenientes de diferentes destinos en diferentes momentos. Un usuario puede escribir datos, seleccionar menús o mover el mouse en cualquier momento. Estas acciones son conocidas por Windows como eventos, y típicamente resultan en la ejecución de alguna pieza de código de la aplicación. Así la secuencia del programa esta determinada por las acciones del usuario. Por ello las aplicaciones Windows consisten básicamente de diferentes piezas de código que responden a los diferentes eventos de Windows.

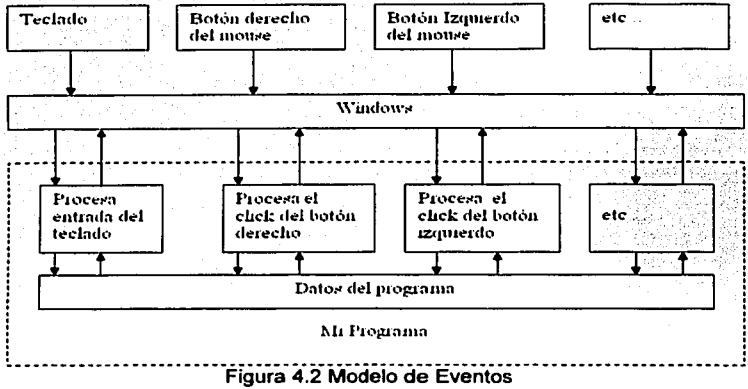

Figura 4.2 Modelo de Eventos

En la figura 4.2, la región punteada representa nuestro programa y cada bloque dentro de el representa una pieza de código escrita especialmente para atender un evento en particular.

El esquema típico para las aplicaciones Windows involucra básicamente dos funciones, WinMain(), que inicia la ejecución del programa, y WindowProc() que se encarga de procesar los mensajes de Windows. [10]

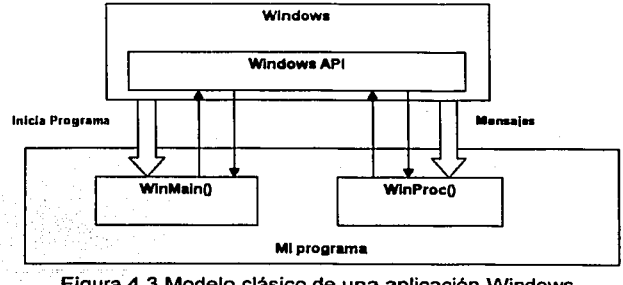

Figura 4.3 Modelo clásico de una aplicación Windows

La figura 4.3 muestra el estilo típico de programación Windows, WinMain() y WindowProc() interactuaban directamente con la API de Windows, esto en realidad sigue ocurriendo sin embargo Microsoft creó las Microsoft Foundation Classes (MFC), un conjunto de clases que encapsulan prácticamente todas las funciones de la API de Windows y ofrecen el paradigma de objetos a la programación Windows.

4.2.2 Microsoft Foundation Classes

La biblioteca MFC contiene clases para la gestión de objetos Windows y ofrece una serie de clases de propósito general que se pueden utilizar tanto en aplicaciones MS-DOS, como Windows. Por ejemplo existen clases para crear y gestionar archivos, cadenas, tiempo, almacenamiento permanente y manejo de excepciones. La biblioteca MFC representa casi todas la prestaciones de la API de Windows e incluye un código sofisticado que simplifica el procesamiento de mensajes, diagnósticos y otros detalles que son partes normales de todas las aplicaciones Windows. Sus clases prestan apoyo a todas las funciones API de Windows utilizadas con frecuencia, incluyendo funciones de ventanas; mensajes, controles, menús, cuadros de dialogo, objetos GDI (Graphic Device Interface) tales como fuentes, brochas, lápices y mapas de bits, enlace de objetos y la interfaz de documento múltiple. Además ofrece un manejo automático de mensajes. Incorpora una arquitectura robusta. Anticipando la gran necesidad del estándar ANSI de C, la biblioteca MFC incorpora una amplia arquitectura para el manejo de excepciones . Esto permite que un objeto MFC se recupere elocuentemente a partir de condiciones de errores estándar tales como "sin memoria", opción no valida y problemas cargando algún recurso o archivo.

4.2.3 OpenGL y las MFC

La parte OpenGL en nuestra interfaz es verdaderamente pequeña, almenos en este momento, pero no por ello carece de importancia, al contrario, es una de las consideraciones medulares de esta implementación.

Como se menciono antes OpenGL es independiente del sistema de ventanas y funciona de diferente forma en diferentes plataformas, en Windows puede utilizarse con librerías auxiliares como "glAux" o "glut", pero en nuestro caso utilizamos el soporte de las MFC.

Las MFC tiene un paradigma estándar cuando es el propio Windows el que dibuja en el área cliente, pero nuestra aplicación suprime a Windows en esa tarea, y es OpenGL el que se encarga de dibujar, para ello la aplicación host requiere algunas consideraciones adicionales mas o menos elaboradas, cuya descripción invitamos al lector a consultar en las referencias, a fin de no perder

el foco de nuestra atención, sin embargo dada su importancia, hacemos mención de ellas rápidamente. [9,21]

La especificación de OpenGL soporta conceptos como el doble buffer y planos adicionales de video que no son soportados por Windows, por eso se necesitan las siguientes inicializaciones:

- Modificar el estilo de ventana
- Definir el formato de pixel
- Crear el *Device Context*
- Crear el *Rendering Context*
- Sobrescribir la función que atiende el mensaje WM_SIZE
- Sobrescribir la función que atiende el mensaje WM_PAINT
- Sobrescribir la función que atiende el mensaje WM_ERASEBACKGROUND
- Eliminar el *Rendring Context* y el *Device Context* al finalizar la aplicación

Estas consideraciones son esenciales e indispensables para que funcione el código OpenGL que utilizamos en nuestra implementación. [16]

4.3 Implementación

Par la implementación de esta interfaz de usuario utilizamos el Ambiente Integrado de Desarrollo (AID) de visual C++ (VC++). Las herramientas de automatización de Visual C++ permiten desarrollar aplicaciones Windows rápidamente; contestando solo algunas preguntas VC crea automáticamente todas las clases necesarias, que constituyen el esqueleto funcional de toda aplicación MFC, el cual incluye elementos gráficos típicos de una aplicación Windows como menús y controles. Después de eso el desarrollo se vuelve interesante, hay que agregar el código que atenderá las acciones del usuario y los datos de la aplicación. Podemos agregar y quitar controles en el editor grafico de VC++ a discreción y solo debemos preocuparnos por agregar código consistente con las MFC que realice las acciones que cumplen los objetivos de la GUI.

Partamos del esqueleto funcional que nos ofrecen los asistentes de VC++, así describimos el contexto general de la aplicación y gradualmente nos enfocamos al código que representa la funcionalidad de nuestra interfaz. El AID de VC organiza sus aplicaciones en espacios de trabajo llamados proyectos, que agrupan todos los recursos de una aplicación (controles, mapas de bits, fuentes, headers, etc), a nuestro proyecto decidimos llamarlo "Tomografo".

Los asistentes de VC++ siempre generan 4 clases además de algunas clases adicionales propias de cada tipo de proyecto (dll, exe, consola, etc.). La figura 4.4 muestra las clases del proyecto *Tomografo* .

Figura 4.4 Clases del proyecto tomógrafo

Las clases básicas son CTomografoApp que representa la aplicación en si, por ello solo existe un objeto de esta clase, ese objeto es la aplicación. La clase CMainFrame se encarga de todo lo que tiene que ver con el marco principal de la GUI. CTomografoDoc es una clase que se utiliza para crear objetos que contienen los datos de la aplicación. La clase CTomografoView, se encarga de las vistas (despliegue de datos en el área cliente).

Una de las ventajas en el desarrollo en VC es que no es necesario conocer todas las clases y por supuesto tampoco es necesario conocer a detalle toda su funcionalidad, pero la idea general del diagrama de clases ayuda a entender de donde vienen las clases que se generan con las herramientas de automatización y por ello nos permite entender mejor la aplicación.

A continuación presentamos la figura 4.5 que muestra un diagrama de las clases más importantes y la forma en que se derivan para dar origen a una aplicación determinada, en este caso nuestra interfaz a la que llamamos tomógrafo.

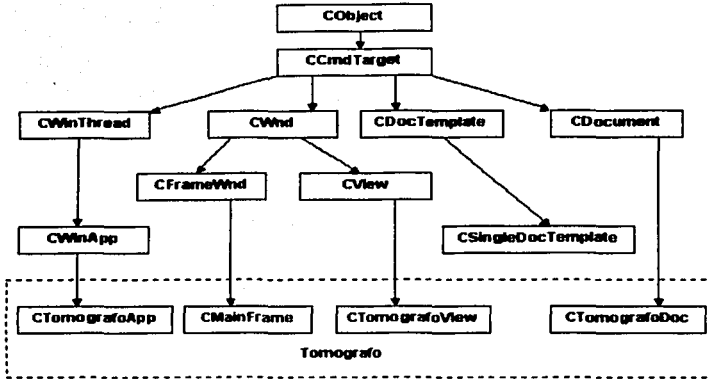

Figura 4.5 Derivación de clases para crear la aplicación tomógrafo

En el diagrama pueden verse las clases más importantes y sus derivaciones. Para nuestros fines conocer las clases de nuestra aplicación, (aquí se muestran enmarcadas por una línea punteada) es suficiente.

Las clases se declaran en archivos .h, por ejemplo la clase CTomografoView esta declarada en el archivo Tomografoview.h, la clase CTomografoDoc esta declarada en Tomografodoc.h, etc. La implementación de estas clases esta contenida en archivos .cpp, la clase CTomografoView esta implementada en Tomografoview.cpp, la clase CTomografoDoc en Tomografodoc.cpp, etc.

La figura 4.6 fue tomada del IDE de VC++ y muestra los archivos de declaración e implementación para las clases de nuestra aplicación.

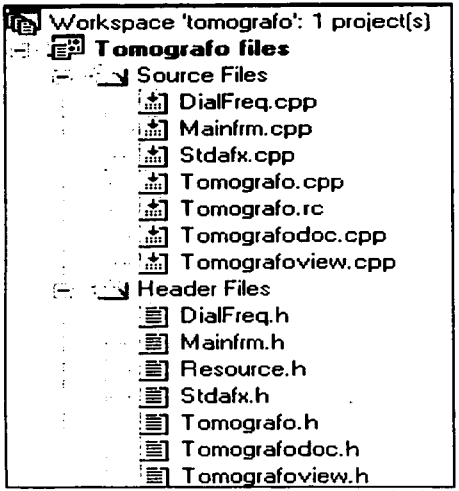

Figura 4.6 Archivos de las clases de la aplicación Tomógrafo

Así pues buena parte del código de nuestra aplicación host, es código que se encuentra en cualquier aplicación MFC y puede construirse rápidamente con los asistentes del AID. Por ello vamos a permitirnos ignorar algunas líneas de código y presentar solo aquellas que son relevantes. De cualquier forma el código completo se anexa en los apéndices y esta suficientemente comentado para poder ser leído con facilidad.

La clase más importante es CTomografoView declarada en Tomografoview.cpp, es en esta clase donde se encuentran las instrucciones del despliegue y el código que interactúa con el SHARC. La figura 4.7 muestra la estructura de esta clase, de la forma en que se ve en el AID de VC++, aquí se pueden ver todas las funciones y variables miembro de esta clase.

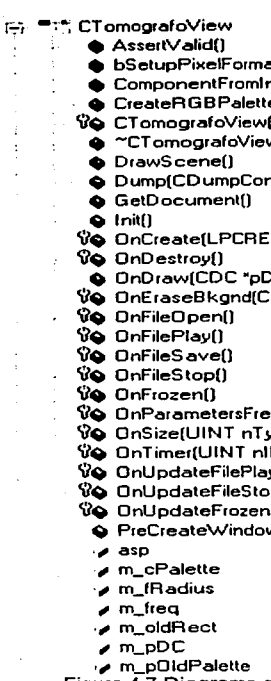

Figura 4.7 Diagrama de la clase CTomografoView

La mayoría de las funciones miembro son funciones estándar que aparecen en cualquier clase de vista(CView) generada por los asistentes de VC++; las funciones bSetupPixelFormat, ComponentFromIndex, CreateRGBPalette e Init se encarga especificamente de las inicializaciones para OpenGL exactamente de la misma manera que se explico anteriormente. Las funciones que requieren ser analizadas a detalle para fines de este proyecto son OnFilePlay, OnTimer y DrawScene. A continuación describiremos el núcleo funcional de nuestra GUI y nos referiremos en detalle a cada una de estas funciones

Todo empieza cuando al usuario hace clic en un botón en la barra principal titulado "Play", el cual tiene código asociado para llamar a la función OnFilePlay() que es una función miembro de la clase CTomografoView, el listado 4.1 es el código de la funcion OnFilePlay.

```
void CTomografoView::OnFilePlay()
{
    ErrCode res;
    m_play = m_play ? FALSE : TRUE; // m_play = !m_play
    if (m_play)
    {

    DWORD host_flags = PCI_ITER << 16; // corre iter 16 bits a la izquierda
    DWORD_Ptr buf = 0; // corre the library to allocate a locked DMA buffer for us

    asp=asp::Initialise(0,ASP_P15_DEVID);
    ASP14_BoardIF_Common_Ptr uif = asp->GetUtilityIF(); //uif instancia de la clase ASP_14_BoardIF..
    #ifdef PCIBUS_SWAPPED
        host_flags |= 0x2;
    #endif

    uif->ASP_POKE_REG_PCTL( PCTL_MBOX6, host_flags ); //escribe a los registros del xharc
    res=asp->BootSharc( "prvdsp.bin" ) ; //bootea los xharc con este binario
    res=asp->OpenComms( buf, N_ELEMENTOS ) ; //el tamaño del buffer son N_ELEMENTOS intel words(32b)
    SetTimer(1, 10, NULL); //timerID=1, genera WM_TIMER cada 10mSec

    }

    else
    {
    KillTimer(1);
    var='e';
    asp->SendW( buf2, sizeof(unsigned) ) ;
    asp->CloseComms() ;
    delete asp;
    asp::Cleanup();

    }


}
```

Listado 4.1 Función OnFilePlay

Lo primero que hace esta función es negar una variable booleana llamada "m_play" que se utiliza globalmente para saber si la animación esta corriendo o esta detenida, su valor inicial es "False". Si la animación esta detenida no existe ningún canal de comunicación con los DSP's y la memoria esta limpia(no hay buffers relacionados con la comunicación SHARC-Host). La parte host de la "Host Communication Library" provee un clase para comunicación llamada asp. La función miembro asp::Initialise(host) crea e inicializa una instancia de la clase asp, para comunicarnos con la tarjeta ASP-15 en al sistema host. La aplicación host utiliza la función asp::BootSharc(host) para cargar una aplicación en la red de SHARC's. En este caso carga "prvdsp.bin", un archivo binario que contiene instrucciones para cargar los diferentes ejecutables en los respectivos DPS's.

Una vez booteada la aplicación en la red de SHARC's puede iniciarse una sesión de comunicación con una llamada a la función asp::OpenComms(host), que termina con la función asp::CloseComms(host). Cuando la sesión de comunicación esta abierta se usan las funciones asp::SendW(host), asp::Recv(host). Después dispara un timer que genera mensajes cada 10 mili segundos. Este timer es el que vamos a utilizar para la animación, la función que responda a estos mensajes es la que se encargara del despliegue de los nuevos datos, su nombre es OnTimer y también es una función miembro de la clase de vista.

Si "m_play" es igual a verdadero significa que la animación esta corriendo, lo que implica la existencia de una instancia asp y por tanto de un canal de comunicación implementado vía buffers, así que lo que hace es destruir el objeto "asp" y limpiar la memoria. Con "m_play" igual a verdadero, la ejecución del programa entrara en el bloque else del condicional if, aquí termina el timer que genera los mensajes para realizar la animación, envía el token de terminar a los DPS's, cierra el canal de comunicaciones, elimina la instancia asp y libera de memoria todos los objetos creados con asp::Initialise(host), lo que se logra con asp::Cleanup(host).

Asumiendo que la animación esta detenida cuando se presionó *Play* lo siguiente que ocurre es el inicio de la función OnTimer, cuyo código se muestra en el listado 4.2.

```
void CTomografoView::OnTimer(UINT nIDEvent)
{

    ErrCode res;
    var='b';
    buf2=&var;
    DWORD_Ptr buf = asp->GetDMABufferMemory();
    if( Failed( res = asp->SendW( &var, sizeof(unsigned) ) ) ) //envia token
        goto Done;
    asp->RecvW( buf, N_ELEMENTOS*sizeof(unsigned) ) ; // lee la matriz
    imagen=buf;
    DrawScene();
    Done:;
    CView::OnTimer(nIDEvent);
    MSG msg;
    while(::PeekMessage(&msg, m_hWnd, WM_TIMER, WM_TIMER, PM_REMOVE));

}
```

Listado 4.2 Código de la función OnTimer

La función OnTimer lo primero que hace es enviarle un token al programa sharc.c, hemos convenido que los tokens sean letras, el token que envía es una "b" cuyo significado para el programa sharc.c es "listo para recibir la nueva matriz, entonces el SHARC sabe que debe enviar la matriz porque el host ya esta esperando. En la siguiente línea lee la matriz enviada por el SHARC y después llama a la función DrawScene() que es la que hace el despliegue propiamente, código que se presenta en el listado 4.3.

```
void CTomografoView::DrawScene(void)
{
    int *ptr=NULL;
    float *ptr2=NULL;

    if(m_play)
    {
    ptr=(int *)imagen;
    ptr=ptr+N_ELEMENTOS;
    ptr2=(float*)matrizrgb;
    for(int d=0;d<N_ELEMENTOS;d++)  // llena la matriz RGB  **********
    {
        for(int h=0;h<3;h++)
        {

            *ptr2=((float)(*ptr)/255);
           ptr2++;
        }


        ptr--;
    }                              //              ****************************

    }

    glClearColor(0.75,0.75,0.75,1.0);
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
    glColor3f(.75,.5,.4);
    glRasterPos2i(xpos,ypos);
    glPixelZoom( 4, 2 );
    glDrawPixels(COLUMNAS,RENGLONES,GL_RGB,GL_FLOAT,matrizrgb);
    SwapBuffers(wglGetCurrentDC());

}
```

Listado 4.3 Código de la función DrawScene

La función DrawScene es también una función miembro de la clase CTomografoView, es llamada una vez por matriz recibida y lo que hace es generar a partir de la matriz recibida (una matriz bidimensional de enteros) una matriz de tres dimensiones con componentes RGB (tres valores entre cero y uno), escoge el color con el que va a dibujar, la posición en donde va a poner la imagen y llama a la función glDrawPixels que lee la matriz RGB y la despliega en pantalla (esta matriz es desplegada en el back buffer), finalmente hace el cambio de buffers SwapBuffers() .

Este proceso se repite por cada mensaje que recibe de la función OnTimer. Cuando el usuario decide detener la animación (presiona uno de los controles) se mata el timer y se elimina la instancia de la clase asp con lo que se limpia la memoria como se ve en el Listado 4.1.

La contraparte de nuestra aplicación es el programa "sharc.c" que se ejecuta en el SHARC raíz (A), como se mencionó antes este es el punto de comunicación entre la GUI y la red de DSP´s, aquí se recibe la información proveniente de la línea de procesamiento, se forma la matriz de imagen y se envía al sistema host cuando es solicitada. Para ello el programa se mantiene en un while infinito en el que sólo esta leyendo los mensajes del host y ejecuta diferentes acciones en función de las peticiones del host. El código de este programa se muestra en el listado 4.4.

```
#include <asp_lib.h>
#include "eg.h"
#include <stdio.h>
#include <tdsp.h>
#include "deftomo.h"
#include "protomo.h"

void main(int argc, char *argv[])
{
    unsigned c[1];
    int a=0;
    unsigned *img;
    unsigned *buffer;
    int fm, nd, linkin,j, m;
    int linkout, cuenta;
    int t;
    int imagen [RENGLONES] [COLUMNAS];
    int wait=0;
    int p;
    ASSERT_FUNC_OK( pmx_init( TRUE, TRUE ) );
    ASSERT_FUNC_OK( asp_Initialize( 0, 0, 0, host_flags & 0x1 ==) );
    set_flag_nybble( 0 );

    /* Frecuencia de muestreo a 2 MHz*/
      fm = 0x15ef4e;

    /*Numero de datos en la cuenta, con 16bit */
      nd = 250;

    /*Verifica el link preprogramado para la salida*/
    linkout = atoi (argv[1]);

    /*Verifica el link preprogramado para la entrada*/
    linkin = atoi (argv[2]);
```

```
    /*Programa el link de salida*/
    programa_link_salida(linkout);

    /*Programa el link de entrada*/
    programa_link_entrada(linkin);

    /*Ubica el buffer en la memoria para recibir los datos*/
    buffer = (unsigned *)malloc(RENGLONES);

    /*Habilita el link de salida*/
     IOP(LCTL) |= LEN(linkout);

    /*Envia parametros de operacion*/
    envia_dato(linkout, fm);
    envia_dato(linkout, RENGLONES);

     /*Habilita el buffer de entrada*/
     IOP(LCTL) |= LEN(linkin);


     img=(unsigned *)imagen;
     z = nd + 1;

     while (1)
     {
            for (j = 1; j <= COLUMNAS; j++)   /*Recibe y forma la matriz de imagen
        {
          link_dma_start_dma(linkin, buffer, RENGLONES+1);
          link_dma_wait_dma(linkin);
          for (cuenta =250; cuenta>0 ; cuenta--)
          {
              t = buffer[cuenta];
              imagen [cuenta] [j] = 255 - t;
          }
        }

     ASSERT_FUNC_OK( asp_RecvW( c, 1, DMODE_3232 ) );  /* Lee el token proveniente del host
     switch(c[0]) {

     case 'b':
                  asp_SendW(img,N_ELEMENTOS,DMODE_3232);
                  break;
      case 'e':  exit(0);
           }
     }
}
```

Listado 4.4. Código del programa sharc.c

Actualmente lo que hace *sharc.c* es formar la matriz de imagen (primer *for* dentro del *while* infinito), después lee un buffer de un carácter en donde lee el *token* enviado por el host, este *token* nos sirve para entrar en un case que depende de su valor, se ejecuta uno u otro fragmento de código en función del mismo. Por ahora sólo están implementados dos casos, enviar la imagen o terminar la aplicación. Naturalmente aquí podemos agregar tantos casos como queramos y podemos agregar fragmentos de código que ordenen diferentes tareas a la línea de procesamiento, que transmitan o reciban parámetros, etc.

Capítulo V
Resultados

5.1 –Funcionalidad

5.2 –Desempeño

Vamos a presentar los resultados en dos planos, funcionalidad y desempeño. En la parte de funcionalidad hacemos un pequeño sumario de las características con las que el usuario interactúa directamente. En la parte de desempeño básicamente medimos el número de imágenes por segundo y comparamos las imágenes desplegadas por la GUI con una aplicación Matlab que construimos previamente para tener un patrón de comparación de las imágenes ultrasónicas que esperábamos ver.

5.1 Funcionalidad

El resultado desde el punto de vista de usuario, es una aplicación Windows que funciona con los elementos usuales, menús y controles. Las aplicaciones Windows son extraordinariamente populares, prácticamente cualquiera que ha usado una computadora conoce el modo de operación de Windows, de ahí que el uso de nuestra interfaz sea prácticamente intuitivo, además tiene pocos controles lo que hace mas simple su operación y evita la abrumadora sensación de complejidad que provocan las interfaces con controles excesivos.

En la figura 5.1 aparece nuestra GUI, es una aplicación de una sola ventana que despliega las imágenes ultrasónicas en el área cliente. Esta ventana tiene todo el comportamiento de las aplicaciones Windows, puede moverse, cambiar de tamaño, maximizarse, minimizarse, etc.

Figura 5.1- Pantalla de inicio.

El usuario encontrará los menús clásicos como File, Save, View y Help. La figura 5.2 muestra el clásico diálogo Windows que se despliega cuando vamos a abrir un archivo.

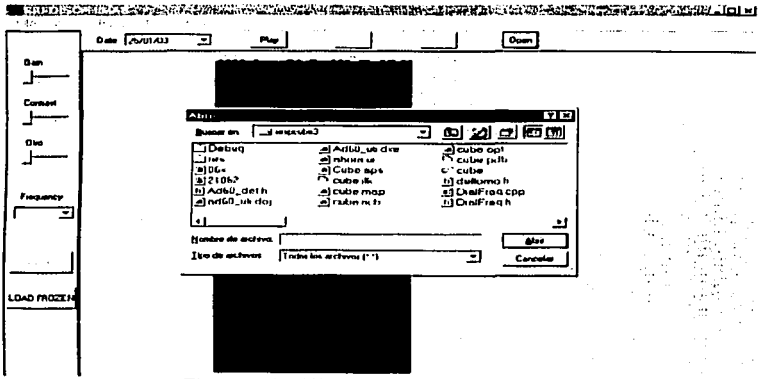

Figura 5.2 Diálogo para abrir un archivo.

La animación se inicia con opciones incluidas en el menú *File*, o mejor aun usando los botones ubicados en la barra superior claramente visibles y accesibles, tal como puede verse en la figura 5.3.

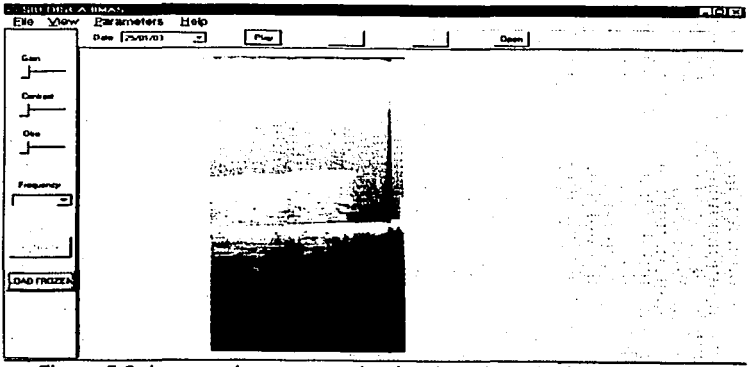

Figura 5.3- Imagen de una esponja, desplegada en la GUI construida.

El usuario puede iniciar una animación y tiene dos opciones para guardar imágenes que resulten de su interés. La primera es usar el botón "FROZEN" ubicado en la barra lateral izquierda, este botón congela la animación y guarda la imagen actual en el formato ".dat". Este formato puede ser leído por matlab o bien por nuestra GUI, para ello solo necesitamos presionar el botón "LOAD FROZEN", esto provoca que nuestra aplicación despliegue la ultima imagen congelada. Si ninguna imagen había sido congelada previamente, "LOAD FROZEN" envía un mensaje de error como el que aparece en la figura 5.4.

Figura 5.4- LOAD FROZEN

La otra alternativa es usar la opción *Save* del menú *File*, o bien el botón *Save* en la barra superior, ello despliega una ventana de dialogo en la que el usuario puede elegir el nombre de la imagen y el directorio donde la quiere guardar. Naturalmente existe su contraparte es decir la opción y control *Open* que funciona exactamente de la misma forma pero obviamente para abrir archivos creados con la GUI.

La figura 5.5 muestra el diálogo que aparece cuando el usuario decide guardar una imagen usando la opción *Save*.

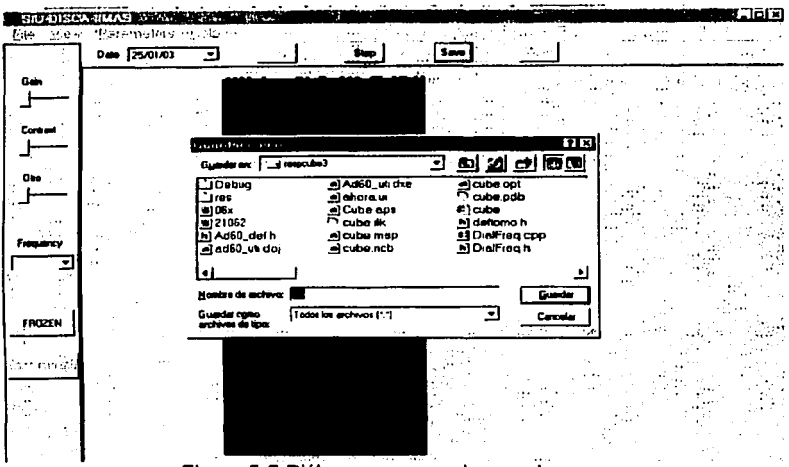

Figura 5.5 Diálogo para guardar una imagen

Por otra parte nuestra implementación contiene los principales mecanismos de comunicación y control para interactuar con los SHARC, lo que implica que futuras implementaciones pueden extrapolarse para utilizar este modelo y solo tendrán que hacer pequeñas adaptaciones
La parte mas importante que son los esquemas de comunicación y el contexto de la aplicación del lado de los DSP's y del lado del host están ya implementados, lo que permite extender la funcionalidad del sistema sin problemas.

5. 2 Desempeño

Aquí presentamos una comparación entre las imágenes desplegadas por nuestra GUI y esas mismas imágenes desplegadas usando Matlab, además hablamos de la taza de despliegue.

Con el fin de tener una buena idea de lo que esperábamos ver en nuestra GUI, construimos un programa que utiliza funciones de Matlab para desplegar las matrices transmitidas de la red de SHARC's al sistema host. Estas matrices son capturadas en el formato que tenían en los DSP's (matrices bidimensionales de enteros) y se despliegan en Matlab para ver las imágenes que esperamos en nuestro sistema; las imágenes son prácticamente iguales. Como puede verse, la imagen de la figura 5.6 es prácticamente igual a la imagen de la figura 5.3. Desde luego es la misma matriz esta siendo desplegada de forma distinta por diferentes programas, la imagen 5.3 es la que podemos ver en nuestra GUI y la imagen 5.6 es desplegada por el programa en Matlab cuyo código ha sido anexado en el apéndice D.

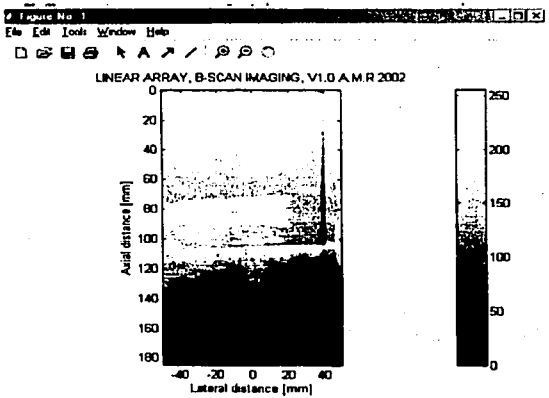

Figura 5.6 Imagen de una esponja desplegada en el programa matlab.

A continuación presentamos otra imagen con objeto de mostrar la gran similitud en el despliegue, lo que confirma que la GUI esta funcionando correctamente.

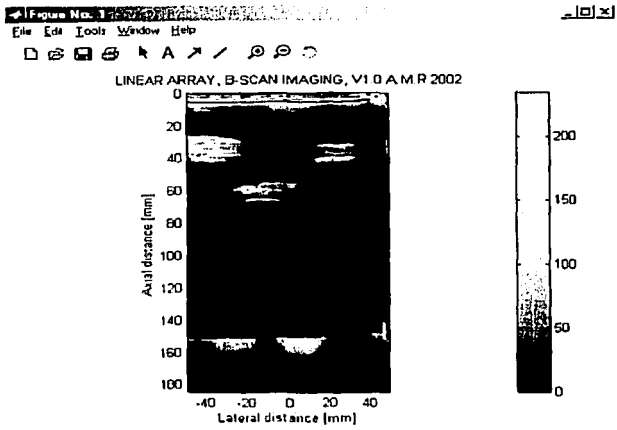

Figura 5.7 Imagen de tres dedos humanos

La figura 5.4 es la imagen de los dedos índice, medio, y anular desplegada en Matlab, nótese que es idéntica a la que despliega nuestra GUI, figura 5.8.

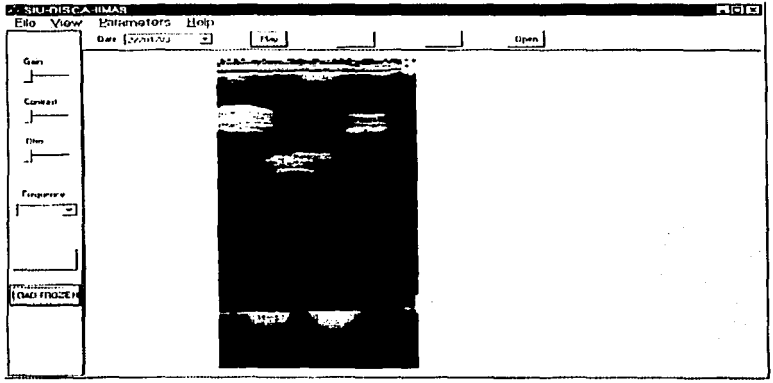

Figura 5.8 Imagen de tres dedos humanos

Lamentablemente en el trabajo escrito no podemos mostrar la imagen en movimiento, en la que puede apreciarse como cambia la imagen en el momento que los dedos se mueven.

Para conocer el número de imágenes desplegadas por segundo, lo calculamos de dos formas. En la primera agregamos un contador que se incrementa cada vez que se llama a la función DrawScene, la cual despliega solamente una imagen cada vez que es llamada, medimos el tiempo que pasa entre el inicio de la animación y el final de la misma, después dividimos el número de imágenes desplegadas entre el tiempo transcurrido. El pseudo código de las siguientes funciones muestra como hicimos esta medición.

```
CTomografoView::OnFilePlay()
{
        Si (animación)
        {
           Hace inicializaciones
           Inicia timer
           Toma tiempo inicio
        }

       Si no

        {
          Toma tiempo final
          Termina timer
          Limpia memoria
        }
}


CTomografoView::OnTimer()
{
        Hace inicializaciones
        Llama DrawScene()
        nimagenes=nimagenes++

        Más instrucciones
}
```

El número de imágenes por segundo es igual al tiempo final menos el tiempo de inicio entre el número de imágenes desplegadas.

Para nuestro segundo cálculo medimos cada cuando cambia la imagen desplegada en el área cliente de nuestra aplicación, para ello tomamos el tiempo que pasa entre llamadas a la función de OpenGL que despliega la matriz de imagen, tal como se muestra en el siguiente pseudo código.

```
CTomografoView::DrawScene()
{
        Hace inicializaciones
        Llama glDrawPixels();
        Llama glSwapBuffers();
        Tomar tiempo
}
```

Con ayuda del IDE de VC ++ obtuvimos el tiempo que pasa entre llamadas a la función glSwapBuffers y calculamos cuantos cambios de imagen se hacen en un segundo, lo cual es equivalente al número de imágenes que se despliegan en un segundo.

Ambas mediciones mostraron que estamos desplegando 10 imágenes por segundo. A pesar de ser un número de imágenes menor al que esperábamos la animación es bastante buena, no se aprecia ningún efecto molesto a simple vista. Por otra parte la capacidad del subsistema de adquisición es de 33 a 40 matrices de imagen por segundo. Actualmente se esta realizando trabajo en esta dirección con la finalidad de aprovechar la capacidad del subsistema, para poder desplegar imágenes con una taza arriba de 25 imágenes por segundo.

Capítulo VI
Conclusiones

6.1 –Conclusiones

6.2 –Trabajo Futuro

6.1-Conclusiones

Este trabajo ha presentado el diseño y desarrollo de una GUI para un sistema de imagenología ultrasónica disponible, que permite controlar los procesos de formación y despliegue de imágenes ultrasónicas en dos dimensiones en tiempo real.

La GUI ofrece un ambiente amigable, que permite definir diferentes tipos de parámetros asociados a los procesos de adquisición, procesamiento y despliegue del sistema de imagenología ultrasónica. Es una aplicación MFC que integra tres diferentes piezas de software, la API de Transtech (ASP-P15), OpenGL y las mismas MFC.

El número de imágenes desplegadas por segundo esta por debajo de lo que esperábamos, sin embargo la animación es bastante buena, no parece lenta a simple vista, el movimiento de los objetos en observación se aprecia con sensibilidad natural. Es importante destacar que el subsistema de adquisición puede entregarnos entre 33 y 40 matrices por segundo, de acuerdo a mediciones realizadas en el mismo, por lo que tiene sentido la optimización del proceso de despliegue.

La GUI permite obtener imágenes ultrasónicas en diferentes formatos que pueden ser leídas, almacenadas o transmitidas a cualquier computadora, lo cual genera oportunidades de intercambio y cooperación con otros proyectos de investigación.

6.2-Trabajo Futuro

Uno de los ejes principales del trabajo futuro es la optimización del proceso de despliegue. Durante este trabajo, realizamos mediciones que demuestran que el subsistema de adquisición puede entregar a la aplicación host entre 33 y 40 imágenes por segundo, suficiente para un sistema de tiempo real.

Por otra parte el presente trabajo constituye una plataforma abierta que servirá de base para el trabajo de investigación que se desarrolla actualmente en al DISCA-IIMAS-UNAM y que esta orientado al procesamiento de imágenes ultrasónicas, así las implementaciones futuras más inmediatas estarán enfocadas a herramientas de procesamiento digital como filtros, supresores de ruido, etc. Estas futuras implementaciones requerirán la adición de algunos menús y controles para coordinar la nueva funcionalidad del sistema de imagenología, para lo cual solo será necesario extrapolar el modelo de implementación actual.

Una de las características más atractivas para desarrollo futuro es la posibilidad de guardar animaciones de unos cuantos segundos en el formato nativo de Windows AVI, o bien implementar un formato propio. Esta nueva funcionalidad podría implementarse directamente en el código actual utilizando las MFC o bien puede auxiliarse de los famosos "Active X Controls" que son piezas funcionales de software que pueden agregarse fácilmente a una aplicación. En caso de implementar un formato propio, puede reutilizarse el código que estamos usando para guardar y desplegar, con lo que podría cubrirse un excelente porcentaje del código necesario para almacenar y reproducir animaciones.

Finalmente Transtech también provee los drivers de sus tarjetas y sus librerías para Linux, y como la parte central de esta aplicación esta construida con el software de Transtech y OpenGL, se puede migrar todo nuestro sistema a Linux, plataforma que ofrece dos ventajas importantes sobre nuestra implementación actual, una es al tipo de licencias (totalmente libre y abierta), la otra es el apoyo de la comunidad Linux.

Referencias

Articulos:

[1] Frederic L. Lizzi, "Ultrasound Imaging", IEEE Technology Requirements for Biomedical Imaging, 1991 Proceedings, 132-139, 1991.

[2] García Nocetti D.F, Solano J, Moreno E, Sotomayor A, "An Open High Performance System for Real Time Ultrasonic Imaging", Microprocessors and Microsystems (Elsevier) Vol 23, Number 6, 357-363,1999.

[3] G.E. Mailloux, A. Bleau, M Bertrand, and R. Petitclerc "Computer Analysis of Heart Motion from Two-Dimensional Echocardiograms", IEEE Trans. Biomedical Engineering, Vol. 34, 356-364, 1987.

[4] Jens U. Quistgaard, "Ultrasonic Image Formation: Implications for the Image Processing Practitioner", Proc. IEEE International Conference on Image Processing, Vol. 3, 533-537, 1994.

[5] Jens U. Quistgaard, "Signal acquisition and Processing in Medical Diagnostic Ultrasound", IEEE SIGNAL PROCESSING MAGAZINE, Vol 15, 67-74, 1997.

Libros:

[6] Chris H. Pappas, William H. Murray, Visual C++ 6.0 McGraw-Hill, 1999.

[7] David H Evans,W. Norman McDicken, Doppler Ultrasound: Physics Instrumentation and Signal Processing 2 edition, Wiley,2000

[8] Dan R. Olsen, Jr, Developing User Interfaces, Morgan Kaufmann Publishers, Inc. 1998

[9] Dennis M. Ritchie, El lenguaje de programación C, Prentice Hall, 1992

[10] Ivor Horton, Beginning Visual C++ 6, Wrox Press, 2002

[11] Jackie Neider, Tom Davis, Mason woo, OpenGL Programming Guide Addison-Wesley, 1993.

[12] Jorgen Arendt Jensen, Estimation of Blood Velocities Using Ultrasound Cambridge University Press, 1996.

[13] Mark J. Kilgard, OpenGL programming for the X windows system Addison –Wesley,1996

[14] Norman, D. A, Why interfaces don't work. In The art of human-computer interface design, B. Laurel. Reading, MA: Addison-Wesley

[15] Peter Fish, Physics and Instrumentation of Diagnostic Medical Ultrasound John Wiley & Sons, 1990.

[16] Richard S. Wright, Jr, Michael Sweet, OpenGL Superbible, Waite Group Press, 1996

[17] Schneiderman. B, Designing the user interface: Effective strategies for effective human-computer interaction. 2nd Ed, Reading, Mass. Addison-Wesley.1992.

Cursos:

[18] Jorgen Arendt Jensen,Notes for the Short Course *Ultrasound Systems for Blood Velocity Estimation* México, 2001.

Manuales:

[19] ASP TOOLSET MANUAL
1999 Transtech DSP Corp.

[20] Sharc User's Manual
1999 Transtech DSP Corp.

Referencias electrónicas :

[21]

http://www.analog.com/

[22]

http://www.codeguru.com/forum/forumdisplay.php?s=b594398b44e54b5fac39396bc926caed&forumid=7

[23]

http://ieeexplore.ieee.org/

[24]

http://msdn.microsoft.com/library/

[25]

http://www.opengl.org

[26]

http://www.transtech.com

Apéndices

Apéndices

Como se explicó en el capítulo de implementación la aplicación host es una aplicación MFC construida con el App Wizard de Visual C ++, este tipo de aplicaciones están constituidas por diferentes clases que se implementan en diferentes archivos. Aquí presentamos el código de esos archivos para la aplicación "Tomografo" (GUI), que es el nombre del proyecto en al ambiente de desarrollo de Visual C++.

El listado completo de todos los programas de este proyecto supera las 40 páginas, por lo que decidimos incluir un disquete que contiene los apéndices de esta tesis. Los archivos en el disco están distribuidos de la forma que marca el índice, para el apéndice A existe un directorio llamado Apéndice-A que contiene los listados de la aplicación host. Para el apéndice B, el directorio es Apéndice-B, este contiene el programa "sharc.c". El apéndice C contiene los programas que son usados en los distintos nodos de la red de DSP's. Finalmente el directorio Apéndice-D incluye el listado del programa Matlab que se uso para tener idea de lo que esperábamos ver en nuestra GUI.