Escuela Superior Politécnica del Litoral

Facultad de Ingenieria en Electricidad

"GrafWin"

Graficador de Funciones de dos variables para Windows

Manual de Diseño

Proyecto de Tópico de Grado
Previa a la obtención del titulo de
Ingeniero en Computación

Presentado por:

Barahona Morales, Edison Ricardo
García Heras, Segundo Enrique
Mestanza Yépez, Carlos

Guayaquil - Ecuador
1994

Agradecimiento:

Queremos dar un especial agradecimiento al Ing. Sixto García A., Director del Tópico de Graduación por que gracias a este proyecto pudimos comprender que la preparación académica de nuestra área no termina, y más aún que ahora nos damos cuenta de que la carrera que hemos escogido es tan amplia y tan interesante que no se contenta consigo misma, sino que busca el conocimiento más allá de sus fronteras.

En este sistema, no sólo interactuamos con el área matemática, sino con áreas mucho más abstractas e interesantes como el de la lógica. Comprendimos que el Ingeniero de Sistemas es una esponja que aprende todo acerca del área con la que interactúa y se convierte en un experto en ella, pués se necesita más que el breve conocimiento de ésta para poder crear un sistema a la altura de las necesidades.

Estamos conscientes que aún queda mucho por aprender, pero confiamos en que lo aprendido sea como las bases firmes que ayudarán sirviendo de cimientos para construir un futuro.

Gracias también a nuestras familias que nos apoyaron en todo momento, que aunque no comprendían nada de lo que hablábamos, simulaban que sí para mantenérnos contentos y con ánimos para seguir.

Y principalmente, gracias a Dios que siempre fué la inspiración que necesitábamos.

Dedicatoria :

A nuestros padres que siempre nos apoyaron y sin escatimar sacrificio alguno nos brindaron su ayuda.

Declaración expresa

"La responsabilidad por los hechos, ideas y doctrinas expuestas en este proyecto, nos corresponde exclusivamente; y, el patrimonio intelectual de la misma a la Escuela Superior Politécnica del Litoral".

(Reglamento de Exámenes y Títulos Profesionales)

García Heras Segundo

Barahona Morales Edison

Mestanza Yépez Carlos

Armando Altamirano
Subdecano de la Facultad de
Ingeniería en Electricidad

Ing. Sixto García
Director del Tópico de Graduación

Ing. Guido Caicedo
Miembro del Tribunal

Ing. Mónica Villavicencio
Miembro del Tribunal

Introducción

Su diseño fue hecho Orientado a Objetos, pero cuidando de que sea lo más independiente posible para que sea útil para otras aplicaciones futuras. La plataforma que se escogió para su implementación fué WINDOWS, y se utilizó los recursos que ésta brinda. Se aprovechó de la herramienta utilizada como lenguaje de programación : Borland C++, pués se reusaron clases ya escritas además de las suyas propias.

WINDOWS es un ambiente que permite a GrafWin ser portable e independiente del hardware usado. Pués corre lo mismo en un monitor color como monocromático, o con monitores de diferentes resoluciones, o con distintas impresoras.

Para su implementación, se crearon clases para Validación y Evaluación de la función ingresada, para la Graficación y para la Impresión. El objetivo principal, fué hacer éstas clases lo más independientes posibles, para que sirvan como base a otras aplicaciones mucho más complejas. Otro objetivo fué que se usara en lo más posible los recursos disponibles, lo cual se logró usando las clases suministradas por Borland C++.

Diseño de GrafWin

GrafWin fué diseñado orientado a objetos. Este diseño se guió en los sgtes. requerimientos :

- Debe ser independiente.
- Debe ser portable.
- Debe utilizar las clases predefinidas en la herramienta de programación.

A continuación se presenta el diseño de GrafWin y después se explicará cada una de las partes, además de dar una justificación de porqué se hizo de esa manera.

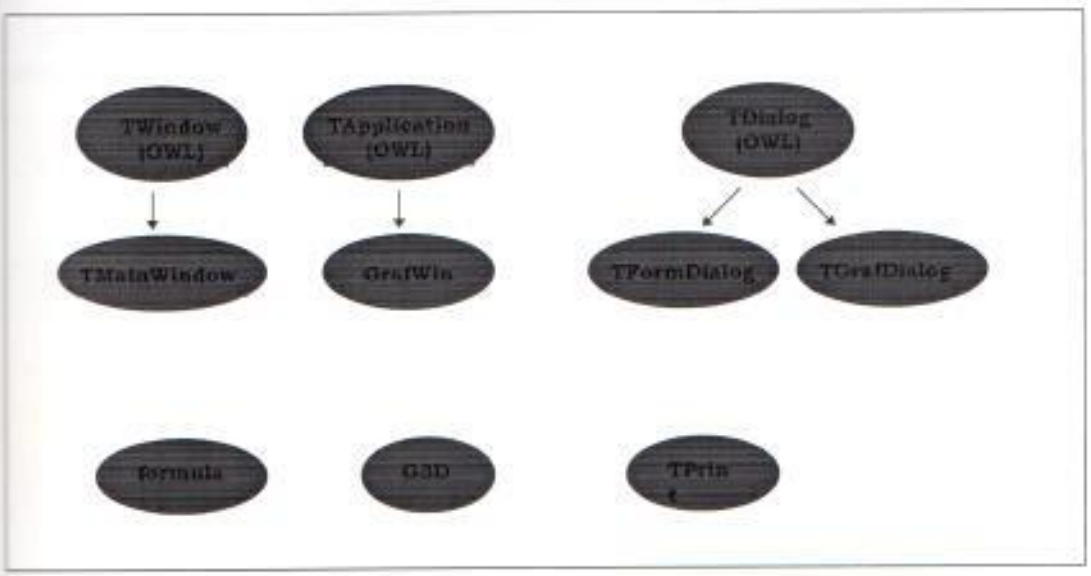

Figura 1. Diseño de GrafWin

Se utilizaron tres clases de predefinidas en la libreria de objetos de Borlandc C++ OWL (Object Windows Lybraries). Estas clase sirvieron de base para derivar las clases que usa GrafWin para interactuar con el ambiente de WINDOWS.

- TWindow es una clase que es usada para el manejo de una ventana de Windows. Como GrafWin sólo usa una ventana, sólo necesitamos crear una clase para esa ventana.
- TApplication sirve para el manejo de la aplicación, mensajes que llegan, interacción con las otras aplicaciones, etc.
- TDialog es una clase que permite la entrada de datos, por medio de una caja de diálogo. GrafWin usa dos de estas clases. Una para la entrada de la fórmula, la otra para la entrada de los parámetros de graficación.

Las otras tres clases independientes son para el manejo interno de la aplicación.

Fórmula es una clase que se encarga de la validación y evaluación de una función. Es independiente tanto así que se puede usar en cualquier otro problema parecido que tenga que validar y evaluar una función de dos variables de la forma z=f(x,y). Si alguien quiere modificar para que acepte funciones de la otra forma, podría hacerlo sin mucho esfuerzo. Sólo tendría que cuidar la misma interfase.

G3D es una clase que se encarga de la graficación de la fórmula. También es independiente y si alguien desea usarla deberá sobrescribir su función Evalúa para que le envíe los valores a graficar. Puede graficar en cualquier HDC, pués es accesible al usuario de la clase, para que él lo setee al HDC que estime conveniente.

Tprint es una clase que se encarga de la impresión. Es independiente y protable. Lee el driver que esta seteado en el archivo de inicialización de Windows WIN.INI y lo carga y usa sus facilidades para imprimir. Consta de funciones de Inicio y fin de trabajo de impresión, inicio y fin de página, etc.

Módulo de Validación y Evaluación de la Función ingresada

Para la validación de la función se usa un juego de reglas de producción que aceptan determinadas operaciones y funciones en notación INFIX. Se ha decidido usar ésta, por que es la notación más usada en nuestro medio. La naturaleza de esta validación es recursiva. La figura 1 muestra las reglas de validación que se usaron.

< *expresion* > = < *termino* > | < *termino* > <*operador aditivo* > <*expresion*>
< *termino* > = < *factor* > | < *factor* > < *operador multiplicativo* > < *termino* >
< *factor* > = (< *expresion* >) | < *constante* > | < *variable* > | < *funcion* >
< *constante* > = *"cualquier número real"*
< *variable* > = " *x* " | " *y* "
< *operador multiplicativo* > = " * " , " / " , " ^ "
< *operador aditivo* > = " + " , " - "
< *funcion* > = <*nombre de funcion*> (< *parametro* >)
< *nombre de funcion* > = *"sen", "cos", "tan", "asen","acos", "atan", "log" , "ln", "exp"*
< *parametro* > = < *expresion* >

Figura 2. Reglas de producción

Para que una expresión sea válida, debe cumplir con los siguientes requisitos :

- Debe estar escrita en notación Infix.
- Las únicas variables aceptadas son X y Y, en cualquier caso.
- Las funciones deben ser : sen, cos, tan, asen, acos, atan, exp, ln, log
- Las constantes pueden ser cualquier numero real. Este número real también puede estar escrito en notación decimal.
- Las operaciones matemáticas aceptadas son : Adición (+), Substracción (-), Multiplicación (*), División (/) y Potenciación (^).

Una vez ya validada la función, se procede a la evaluación. Para ésto se construye un árbol binario. Las hojas del árbol pueden ser constantes o variables, miéntras que las ramas operadores o funciones. La figura 2 muestra cómo quedaría el árbol de una expresión válida :

Sería bueno, en este momento, dar a conocer la estructura de un nodo del árbol, para que quede claro como el árbol puede guardar diferentes tipos de datos en el mismo nodo :

```
struct NODO
{
   OPERATOR op;      // Guarda el Operador,funcion, constante o variable de
                     // acuerdo al caso
   OP_TIPO tipo;     // Guarda el tipo : Operador,funcion,constante o variable
   NODO *hizq,*hder; // Punteros a los hijos izquierdos y derechos. Para una hoja
                     // ambos serian nulos
};
```

Figura 3. Estructura del nodo del árbol de evaluación

La estructura de un Nodo consiste en el dato en si (op), el tipo de dato (tipo), y los punteros a los hijos (hizq,hder). Como el dato puede ser un operador, una constante, una variable o una función, la variable op debe ser también una estructura (OPERATOR), que dependiendo del tipo, guarde el dato en el campo correcto. Como para cada nodo sólo se guarda un tipo de dato especifico, los otros campos de la estructura OPERATOR serían un gasto. Es por eso que OPERATOR ha sido definida como una unión :

```
union OPERATOR
 {
   OPE_MAT operador;                 // Guarda un operador : + , - , * , / , ^

   char funcion[MAXFORMULA+1];       // Guarda una función : sen, cos, etc..

   double constante;                 // Guarda una constante : 112, 24.12,
                                                          2.12e-10, etc...
   VAR variable;                     // Guarda una variable : X ó Y
   };
```

Figura 4. Estructura que guarda el operador

En esta unión, se puede guardar cualquiera de los 4 diferentes tipos de datos. En la variable operador se guarda el signo del operador matemático '+', '-', '*', '/', '^'. OPE_MAT es un tipo de dato enumerado que contiene los codigos ASCII de estos signos; asociados a palabras claves que son mas entendibles en la programación :

```
enum OPE_MAT { OP_ADICION=43, OP_SUBSTRACCION=45, OP_MULTIPLICACION=42,
OP_DIVISION=47, OP_POTENCIACION=94};
```

La variable funcion contiene el nombre de la funcion, tal como fué escrita por el usuario. así como constante, guarda el valor de la constante y variable el de la variable. El campo variable es un campo enumerado que puede tomar como valor cualquiera de las variables aceptadas como válidas en una función.

enum VAR {VARX,VARY};

Para que se pueda accesar al campo correcto, el nodo cuenta con una variable que guarda el tipo del dato. Según este tipo se sabe que es en realidad lo que guarda el nodo. El tipo de dato de este campo (OP_TIPO) se detalla a continuación :

Implementación de fórmula usando Orientación a Objetos OOP

Para implementar la evaluación y validación de la función, se utilizó la clase **fórmula**. Tiene construida dentro de ella, un ***"Árbol de Evaluación"*** que servirá para almacenar la expresión válida. En cada nodo del árbol, como se menciona arriba, se almacena un operador, una constante, una variable o una función dependiendo del caso.

```
class formula
{
  private:
     BOOL Ok;
     ERROR_TIPO error;
     NODO *raiz;
    char expresion[MAXFORMULA+1];
    int OpAditivo(int,int);   // Funciones para Validacion de la expresion
     int OpMultiplicativo(int,int);
     BOOL IsExpresion(NODO **,int,int);
     BOOL IsTermino(NODO **,int,int);
     BOOL IsFactor(NODO **,int,int);
     BOOL IsFuncion(NODO **,int,int);
     BOOL IsParam(NODO **,FUNCIONES,int,int);
     BOOL IsConstante(NODO **,int,int);
     BOOL IsVariable(NODO **,int,int);
     FUNCIONES IsNombreFuncion(char *);
     NODO *CrearNodo(void);          // Funciones que trabajan sobre el árbol
     void InsNodo(NODO** padre, NODO *nodo);
     void FreeTree(NODO **nodo);
     double Evaluar(NODO *,double,double);   // Funciones que evalúan el arbol

  public:
     formula(void);                  // Constructor default
     formula(char *string);          // Constructor desde una funcion almacenada
                                     // en un string
     ~formula();                     // Destructor
     BOOL GetStatus(void);
     ERROR_TIPO GetError(void);
     void Create(char *);
     double EvaluaZ(double,double);
};
```

Figura 5. Clase que implementa la evaluación y validación de una función

La clase posee dos constructores, el primero no recibe parámetros y sirve sólo para inicializar los datos miembros. Si se quiere construir el árbol de evaluación de alguna función, se usará la función miembro *Create* que será detallada posteriormente.
El segundo constructor recibe como parámetro la función en un string y si es válida construye el árbol de evaluación.

Se acordó hacerlo así y no cada vez que se evalúe, debido al retardo que la validación y la creación del árbol ameritan. Es por eso que se crea el árbol primero (en el constructor de la clase) y cada vez que se evalúe, se ahorrará tiempo del procesador, pués el árbol ya está creado y sólo se necesitará recorrerlo.

El destructor de la clase sólo libera el espacio ocupado por el árbol.
Existen 3 Tipos de Funciones :

- ***De Validación***, Las funciones que validan si la expresión esta bien escrita.
- ***De Árbol***, funciones que operan sobre el árbol de evaluación.
- ***De Evaluación***, funciones que reciben los valores de X ,Y y evalúan la expresión recorriendo el árbol.

Las funciones de Validación son una simple transcripción de las reglas de producción. Estas funciones són :

```
int OpAditivo(int,int);
int OpMultiplicativo(int,int);
BOOL IsExpresion(NODO **,int,int);
BOOL IsTermino(NODO **,int,int);
BOOL IsFactor(NODO **,int,int);
BOOL IsFuncion(NODO **,int,int);
BOOL IsParam(NODO **,FUNCIONES,int,int);
BOOL IsConstante(NODO **,int,int);
BOOL IsVariable(NODO **,int,int);
FUNCIONES IsNombreFuncion(char *);
```

Las funciones descritas anteriormente tienen incorporado la formación del árbol. Ésto lo hacen pasándose como parámetro el nodo donde insertarán la información. Si encuentran que algo no es válido, llaman a FreeTree para liberar todos los nodos creados y que son erróneos.FreeTree será hablada más en detalle posteriormente.

Las funciones de Árbol, son una implementación sencilla del TDA Árbol Binario. Las funciones de este grupo són :

```
NODO *CrearNodo(void);
void InsNodo(NODO** padre, NODO *nodo);
void FreeTree(NODO **nodo);
```

CrearNodo reserva un espacio de memoria para almacenar un nodo del árbol y retorna su dirección. InsNodo inserta un nodo en la posición que se le envía. Debido a que ésta, cambia el puntero que se le envía, este valor es pasado por referencia (NODO **). FreeTree es una función recursiva que libera todos los hijos del nodo especificado, así como al nodo mismo.
Las funciones de Evaluación, recorren el árbol en forma INFIX para ir evaluando la expresión. Es de naturaleza recursiva y valida los parámetros de evaluación para evitar los errores de excepción en el procesador. Estas funciones son :

```
double Evaluar(NODO *,double,double);
double EvaluaZ(double,double);
```

Existen otras funciones, que no han sido tomadas en cuenta. La razón es que sólo sirven como interfase entre la clase y el mundo exterior. Estas funciones són:

```
BOOL GetStatus(void);
ERROR_TIPO GetError(void);
void Create(char *);
```

GetStatus simplemente retorna la variable Ok que dice si la función es válida o no. GetError retorna, en cambio la variable error que dice que error ha ocurrido en la validación. Create es una función que se usa cuando se quiere validar otra función, pero el objeto ya se ha creado. Ésta limpia el árbol (llamando a FreeTree) y Valida la función que se le envía en un string (llamando al constructor).

Existen 4 datos privados dentro de la clase, los cuales le dan una mayor funcionalidad.

```
BOOL Ok;
ERROR_TIPO error;
NODO *raiz;
char expresion[MAXFORMULA+1];
```

Ok almacena la validez de la función. Si es FALSA, ha ocurrido un error y este valor se encuentra en la variable **error**. **raiz** guarda el puntero a la raiz del árbol, si es NULL el árbol no se ha podido

crear, ya sea por invalidez de la función o por falta de memoria. La razón en este caso, también se encuentra en la variable **error**. **expresion** almacena la función que dió forma al árbol.

Para dar una mejor idea de la validación y evaluación de función, se mostrará como quedará el árbol de evaluación después de haber de la validación.

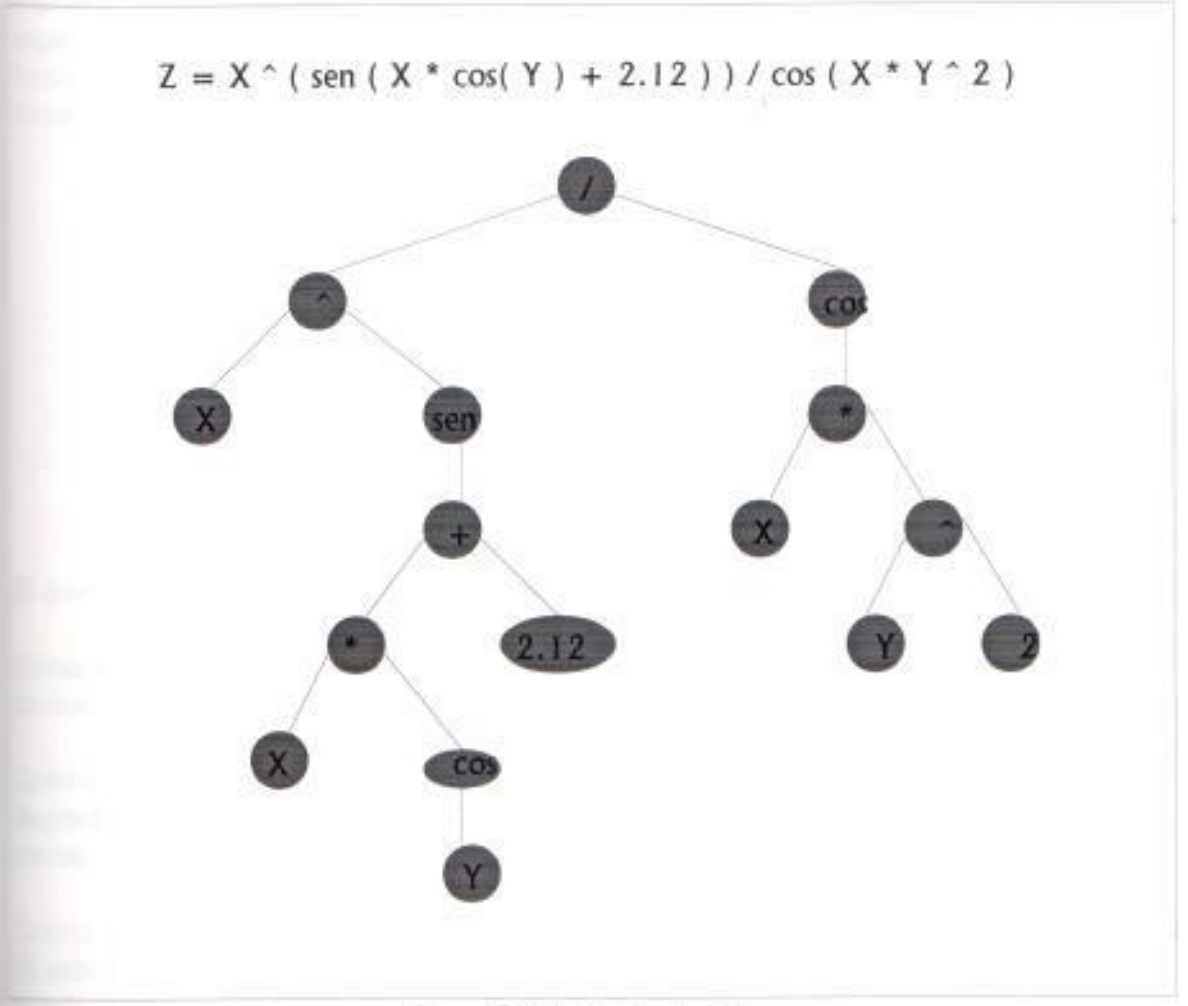

Figura 6. Árbol de Evaluación

Modulo Tprint

Tprint es un módulo que se encarga de obtener el handle de contexto de la impresora. Para aquellas personas que desconozcan el término de dispositivo de contexto, éste es la manera de como windows accesa a los recursos gráficos o d e impresión. Obteniendo el handle de este dispositivo, se puede imprimir en pantalla o e impresora, independiente del hardware instalado.

Para lograr ésto, Tprint lee el archivo de inicialización de Windows para encontrar el driver de impresora instalado, cargarlo y obtener el handle del dispositivo de contexto para la impresora especificada en Windows. Una vez obtenido este handle, se puede utilizar cualquiera de las funciones del GDI (Graphics Device Interfase) de Windows para mostrar texto, lineas, rectángulos, circulos, etc.

Figura 7. Interfase de Tprint con Windows

El driver de la impresora instalada en Windows se encuentra especificado de la siguiente manera :

[Windows]
device=Kyocera F-Series (USA),HPPCL,LPT1:

Obteniendo estas entradas en WIN.INI, cargamos el driver HPPCL y obtenemos el handle del dispositivo de contexto con los otros 2 parámetros. Tprint no sólo cuenta con ésto, sino que da ciertas facilidades usando internamente sentencias de escape que se envian a la impresora.

Cuenta con una cosntructora que encuentra el Handle del dispositivo de contexto de la impresora. Si esteocurrió un error o no se pudo cargar el handle, Tprint setea la variable que contiene este handle (hPr) a NULL. El usuario de la clase, deberá chequear si este valor es válido para poder usarla. Si el usuario de la clase no verifica ésto y usa un handle nulo, puede obtener resultados inesperados.

Cosnta además de una función para el seteo de la impresora PrinterSetup. Esta función carga el driver y obtiene la dirección del puntero a la función de seteo de la impresora default de Windows. Aparece el mismo diálogo que aparece en el la opción Printer Setup del Print Manager.

Posee funciones miembras de inicio y fin de trabajo de impresión, inicio y fin de página, cancelación del trabajo de impresión, etc. Si el usuario quiere mas facilidades, puede derivar la

clase e insertar nuevas funciones asociadas a sentencias de escape. Esto es muy fácil hacerlo. Sólo debe leerse el help de Borlandc C++ o de otro compilador compatible para comprender las diferentes sentencias de escape y entenderlas.

Posee una variable que dice si un error ha ocurrido. Fácilmente se puede preguntar por esa variable y saber si se ha cometido un error. Posee funciones de cálculo del tamaño de la línea para el tipo de letra escogido, útil para hacer una salida bien formateada.

Implementación de Tprint con orientación a objetos

Tprint es implementada en una clase que se detalla a continuación :

```
class TPrint
{
  BOOL Job;
  BOOL Page;
  int Error;
  POINT PageSize;
  char far PrintDriver[MAXPRINTER+1];
  char far PrintPort[MAXPRINTER+1];
  char far PrintType[MAXPRINTER+1];

  void GetPageSize(void);
  void ReleaseDC(void);

public:
  HDC hPr;
  int LineSpace,LinesPerPage;
  float SCALEX,SCALEY,MARGENX,MARGENY;

  TPrint(void);
 ~TPrint(void);
  virtual void AbortDoc(void);
  virtual void StartDoc(void);
  virtual void StartPage(void);
  virtual void EndPage(void);
  virtual void EndDoc(void);
  virtual void PrinterSetup(HWND);
   virtual HDC GetDC(void) { return hPr;};
};
```

Figura 8. Clase Tprint

Tprint ofrece al handle de la impresora como público, para que todos los usuarios de la clase lo puedan usar. Debido a ésto es que el usuario debe tener mucho cuidado en no perder este valor, pués el único que sirve para la labor. Es aconsejable usarlo como de lectura solamente.

Módulo G3D (Gráfico de la función).

General

Los puntos dentro de este algoritmo serán calculados considerando un sistema de ejes cartesianos, esto quiere decir que para definir un punto serán necesarios tres valores, valor para x, valor para y, y valor para z. (x, y, z).

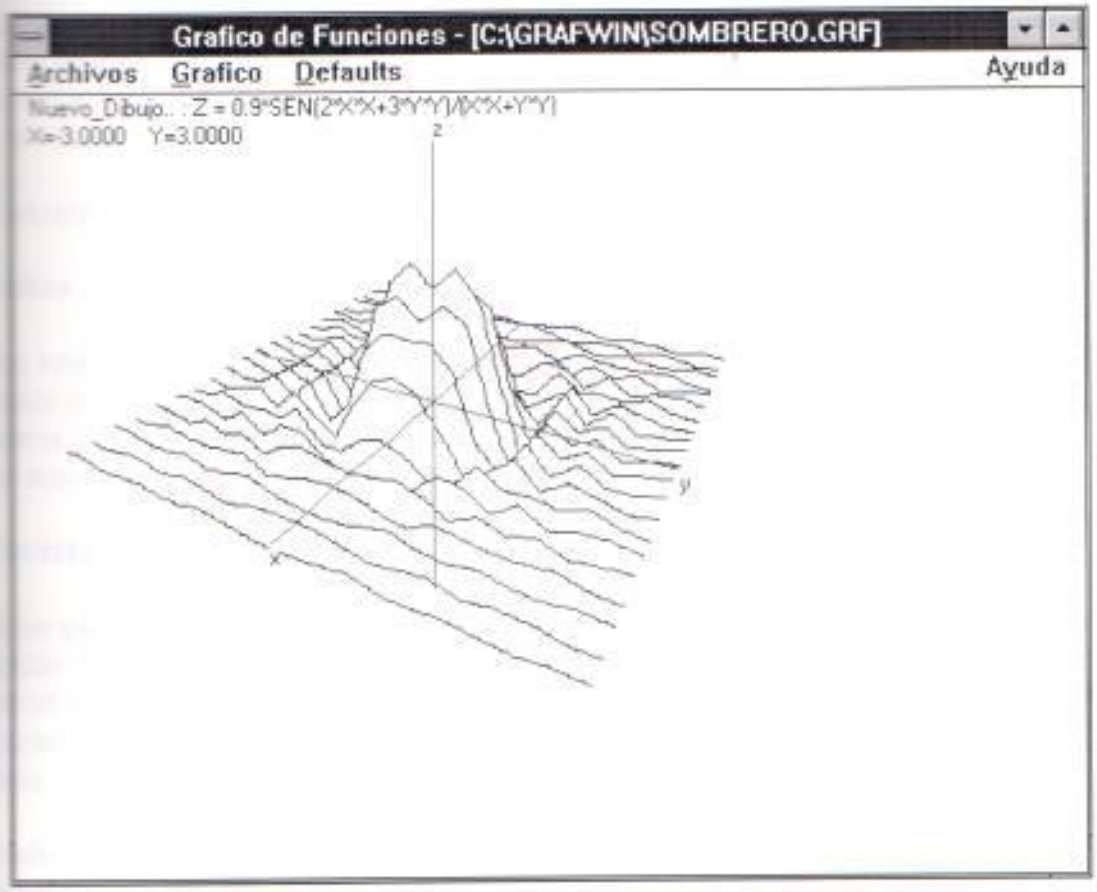

Figura 9.Gráfico obtenido con la clase G3D

Parámetros considerados.

Punto de vista :

El punto de vista o lugar desde donde el usuario desea ver la función, estará dado por un sistema de cooredenadas polares. Para esto es necesario especificar tres valores, ángulo entre el eje X y la proyección del vactor posición sobre el plano XY (theta), ángulo entre el eje Z y el vertor

posición (phi), la magnitud de el vector posición o distancia entre el origen (0,0,0) y el punto de vista (rho).

Se eligió el sistema de coordenadas polares porque nos pareció el más natural para esta función.

Para la interfase con el usuario, theta se leerá como ángulo X, phi como ángulo Z, y rho como distancia.

Dominio :

Los rangos para los valores de X y Y, que serán evaluados, estan especificacos por el usuario a través de los parametros de rango en X, Y y Z. Los valores de xdesde, xhasta, ydesde y yhasta definen el dominio sobre el cual la función será evaluada, los valores de zdesde y zhasta definen el tamaño del eje Z que se graficará.

El usuario verá estos rangos como "RANGO X", "RANGO Y", y "RANGO Z".

Deltas :

Esto indica la calidad o grado de resolución del gráfico, con un delta más pequeño el algoritmo evalua más puntos, los efecto de esto es : se demora más en dibujar la misma función, y se obtiene un gráfico más nítido sobre todo en funciones que presentan cambios muy bruscos (no son suaves).

Escalas :

Sirven para deformar el gráfico a voluntad sin necesidad de cambiar la formula. Por ejemplo la función 'SEN(X+Y)' para X de -5 a 5 y Y de -5 a 5 se ve en la pantalla como algo muy pequeño, que no se aprecia, pero si le ponemos la escala en X igual a 2 y la escala en Y igual a 2, el dibujo aparece tan grande como si elijieramos un dominio de -10 a 10 en X y en Y, y se ve en la pantalla como si la formula fuera 'SEN(2*X+2*Y)'.

Puede ser ademas que las ondulaciones no sean muy notadas o se quiere apreciar mejor las ondulaciones, esto se consigue agregandole un factor a la formula como por ejemplo : 5*SEN(X+Y), este mismo esfecto se consigue con la escala en Z igual a 5, de esta forma el efecto es el mismo y no se requiere cambiar la formula.

Desarrollo matemático :

El punto de vista se representa por las coordenadas : (rho, theta, phi).

Este mismo punto se puede representar en coordenadas cartesianas (x, y, z). Y la relación entre estos puntos es :

x = rho * SEN(theta) * COS(phi)
y = rho * SEN(theta) * SEN(phi)
z = rho * COS(phi)

Además se comprueba que : $rho^2 = x^2 + y^2 + z^2$.

Transformaciones.

Las transformaciones que podemos realizar sobre un espacio tridimensional son : ESCALAMIENTO, ROTACION, TRASLACION y REFLECCION.

Para poder trabajar mas comodamente pasamos los puntos (x, y, z) que corresponden a un espacio vectorial de 3 dimensiones a un espacio vetorial homogeneo de 4 dimensiones, esto se consigue agregando un 1 como cuarto elemento, el punto (x, y, z) será el punto (x, y, z, 1).

Escalamiento.

Esto nos permite hacer más grande o más pequeño la función sin cambiar la formula en sí. La matriz representativa de esta operación es :

$$(x, y, z, 1) \begin{pmatrix} A & 0 & 0 & 0 \\ 0 & B & 0 & 0 \\ 0 & 0 & C & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (Ax, By, Cz, 1)$$

Rotación.

La rotación se realizará en el sentido de las manecillas del reloj. Se requiere de una matriz diferente para la rotación sobre cada uno de los ejes.

La rotación sobre el eje Z tiene la siguiente matriz caracteristica.

$$\begin{pmatrix} \cos f & \text{sen } f & 0 & 0 \\ -\text{sen } f & \cos f & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotación sobre el eje X :

1	0	0	0
0	cos f	sen f	0
0	-sen f	cos f	0
0	0	0	1

Rotación sobre el eje Y :

cos f	0	-sen f	0
0	1	0	0
sen f	0	cos f	0
0	0	0	1

Traslación.

Es el desplazamiento de un punto sobre el espacio :

	1	0	0	0	
(x, y , z, 1)	0	1	0	0	= (x+H, y+K, z+L, 1)
	0	0	1	0	
	H	K	L	1	

Reflección

Es la obtención de una imagen invertida o con efecto de 'espejo', para esto necesitamos tres matrices, una para cada plano.

Para el plano XY.

1	0	0	0
0	1	0	0
0	0	-1	0
0	0	0	1

Para el plano XZ.

1	0	0	0
0	-1	0	0
0	0	1	0
0	0	0	1

Tras el plano YZ

$$\begin{matrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$$

Con el objetivo de obtener una imagen de dos dimensiones (sx, sy) de una imagen de tres dimensiones (x, y, z) es necesario el siguiente proceso :

Si consideramos el punto de vista como el punto P y queremos realizar la transformación de un segmento de recta AB que esta en un espacio de tres dimensiones, la pantalla es un plano sobre el cual la figura es proyectado. El plano de proyección es perpendicular a la linea OP, y a una distancia fija D (distancia entre P y el plano). Todos los puntos del esplacio (x, y, z) son proyectados sobre el plano (sx, sy), esto significa que uno o más puntos (x, y, z) pueden corresponder a un punto (sx, sy).

Los valores de sx y sy se obtienen de la relación entre triangulos semejantes, el desarrollo es puramente geometrica y la esplicación detallada la podemos ver en cualquier texto de geometria.

Implementación de la clase G3D usando OOP.

Descripción de la clase.

La clase tiene de nombre 'g3d' y tiene la siguiente definición :

```
class g3d
{
        float   d,s1,c1,s2,c2,x,y,z;
        float   theta, phi, rho;
        float   sx,sy,ox,oy, ex, ey, ez;
        short   fl,f,mx,my, cx, cy;
        float   MARGENX,MARGENY,SCALEX,SCALEY;
        float   yn[1024],yx[1024];
        void    transforma();
        void    esconde();
        void    evalua();
        float   redondea (float);

        public:

        HDC     hdc;
        void    init_g3d( short tx,short ty,
                          short centx,short centy,
                          float dist,float angx,float angz,
                          float vect,
                          float,float,float,
                          float,float,float,float);
        void    grafica(formula *, float xdesde, float xhasta, float xdelta,
                          float ydesde, float yhasta, float ydelta,
                          float zdesde, float zhasta);
};
```

Figura 10: Descripción de la clase G3D

Privados :

Son variables auxiliares y variables que sirven para almacenar los valores de los parametros.

Publicas :

La función 'init_g3d' que sirve para inicializar los valores de los parametros.
La función 'grafica' que realiza todo el proceso de graficación.

variable HDC, que apunta a la ventana a usar.

Todo el proceso de la tranformación que debe seguir un punto (x, y, z) esta escrita en la funcion 'transforma' dentro de la clase 'g3d'.

El proceso de esconder lineas esta escrito en la funcion 'esconde'.

Diseñado orientado a objetos, es por eso que la herramienta que se debía usar para implementación también debería soportar Orientación a Objetos OOP. BorlandC ++ 3.1 fué la herramienta escogida debido a las facilidades que brinda y a la experiencia anteriormente adquirida en otros compiladores similares de la misma compañía.

Implementación fué modular, con cada módulo del diseño en un módulo de la implementación. También se hizo uso de recursos que Windows necesita para interactuar con su ambiente, cosa que se logró utilizando una herramienta de Borland llamada Resource Workshop, que permite adicionar al programa, recursos como iconos, bitmaps, cajas de diálogos, cursores, etc. Todos estos recursos se encuentran en un sólo módulo de GrafWin.

	Clases contenidas
[illegible].CPP	Aplicación, Ventanas, Cajas de diálogo
[illegible].CPP, GRAFICO.HPP	G3D
[illegible].CPP, PRINTER.HPP	Tprint
[illegible].CPP, FORMULA.HPP	Formula
[illegible].RC	Recursos usados por GrafWin
[illegible].HPJ	Archivo de definición del Help
[illegible].RTF	Archivo fuente del Help
[illegible].HLP	Archivo de Ayuda (HELP)

Figura 11: Módulos de GrafWin

Descripción de Módulos

GRAFWIN.CPP

Este módulo que contiene toda la información que necesita la aplicación para interactuar con el ambiente operativo, es decir con Windows 3.1. Contiene las clases:

- **Aplicación .-** Encargada del manejo de la aplicación y su interacción con Windows.
- **Ventana.-** Encargada del manejo de las ventanas de la aplicación y su coexistencia con las de otras aplicaciones.
- **Cajas de Diálogo.-** Encargada del manejo de todas las cajas de diálogo que la aplicación usa para entrada de datos.

GRAFICO.CPP

Este módulo que contiene la clase G3D, que es la grafica la función, haciendo uso de la as facilidades que brinda el GDI (Graphics Device Interfase) de Windows. Esta clase es independiente a las otras, pués no depende de ninguna, más bien puede usársela en cualquier otro problema similar. Contiene la clase :

- **G3D.-** Encargada de graficar la función que se desea en la pantalla que se le especifique por medio de su dispositivo de contexto, que debe ser seteado antes de la graficación. Debido a que G3D grafica en cualquier dispositivo de contexto, ésta es tan flexible que puede imprimir lo mismo en pantalla, como en una impresora, plotter o cualquier otro dispositivo similar.

PRINT.CPP

Este módulo que contiene la clase de impresión:

- **TPrint.-** Encargada de obtener el dispositivo de contexto de la impresora seteada como default en Windows y de brindar facilidades de Inicio y Fin de trabajo, Inicio y Fin de página, cancelación del trabajo, etc..

FORMULA.CPP

Contiene la clase que se encarga de validar y evaluar la expresión:

- **Formula.-** Encargada de validar y evaluar la expresión, creando internamente el árbol de evaluación que le permite validar y al mismo tiempo evaluar dicha función. Una limitación de la clase es que sólo acepta expresiones en Notación INFIX.

GRAFWIN.RC

Contiene todos los recursos que GrafWin utiliza para su interacción con el usuario. Estos recursos son:

Recurso	Tipo	Descripción
[illegible]Dialog	Caja de Diálogo	Entrada de los parámetros de graficación
[illegible]Dialog	Caja de Diálogo	Entrada de la función
[illegible]	Bitmap	Gráfico de la portada
[illegible]	Icono	Icono de la aplicación
[illegible]MsgBox	Caja de Diálogo	Reseña de la autoría de la aplicación
[illegible]	Menú	Menú Principal de la Aplicación
[illegible]	Caja de Diálogo	Ventana de Portada

Figura 12: Recursos de GrafWin

GRAFWIN.HPJ

Archivo de definición de la Ayuda de GrafWin. Contiene la información necesaria para que el compilador de Help (HELP COMPILER para Windows 3.1) HC31 provisto por Borland compile sin ningún problema el archivo RTF.

GRAFWIN.RTF

Archivo fuente de la ayuda de GrafWin. Es un archivo editado en Microsoft Word 6.0, que contiene toda la ayuda de GrafWin.

GRAFWIN.HLP

Es el archivo RTF compilado y listo para ser usado por la aplicación. Puede también ser leído desde cualquier otra parte, haciendo uso del WINHELP.EXE, que es el programa que provee Windows para poder accesar a los archivos HLP.