Instituto de Computación - Facultad de Ingeniería
Universidad de la República
Año 2004

Tipografía Digital en C5

Un generador constructivo de familias de tipos de letras

Proyecto de Grado

Zelmar Echegoyen Ferreira

Tutor: Juan José Cabezas

Resumen

El diseño de tipos de letras es una tarea mucho más compleja de lo que uno podría suponer. En él intervienen muchos conceptos matemáticos y de computación gráfica que deben ser utilizados como herramientas para describir una idea artística.

En este documento se presenta el análisis, diseño y construcción de un generador constructivo de fuentes de letras (fonts) que se enmarca dentro del Proyecto de Librería Gráfica para el lenguaje de programación C5.

Inicialmente se presenta una acercamiento al problema, donde se explican los principales elementos que lo componen y se mencionan las características más importante que debería cubrir la solución.

Luego, se presentan el diseño y la construcción de un prototipo de Generador de Fonts, donde se utiliza un diseño modular, que divide en 3 capas la generación de las fuentes de letra, de forma tal que en una primera capa se manejen los conceptos de Computación Gráfica referentes a la resolución y la representación de fronteras, en la segunda la definición de cada parte de la letra y en la tercera la composición de la letra final, integrando cada una de sus partes. Aquí se muestran las ventajas del constructivismo en la creación de fuentes de letras.

Luego se experimenta el prototipo de generador de fonts con la creación de una fuente de letras romanas, donde se comprueba la facilidad que el generador le brinda al diseñador. Y Finalmente se analizan los resultados obtenidos, comparando la relación entre precisión, costo de ejecución y tamaño (en bytes) de la letra final.

Asimismo, se muestra además la importancia que tiene que las letras generadas continúen siendo objetos del lenguaje sobre los cuales se puedan realizar operaciones. Por ejemplo, para poder generar efectos sobre el color y tamaño de las letras.

Organización del Documento

El informe se encuentra estructurado en 6 capítulos. En los mismos, se pretende dar al lector una visión global de los aspectos más relevantes del proyecto, profundizándose algunos temas en los documentos que se ofrecen en el Anexo. A continuación se resume el contenido de cada uno de los capítulos.

- Capítulo Nº 1.- Se presenta una introducción al problema y se describen las motivaciones y objetivos del Proyecto, así como el alcance que éste tendrá y la metodología empleada para el desarrollo del mismo.

- Capítulo Nº 2.- Se presenta el problema que dio lugar al Proyecto, mostrando una visión general del estado del arte e introduciendo el contexto y los conceptos principales necesarios para la comprensión del mismo. También se describen aquellos conceptos que deberían tenerse en cuenta en la solución.

- Capítulo Nº 3.- Se da una visión general de la solución, y se explican los conceptos previos necesarios para entender el posterior desarrollo de cada uno de los componentes del diseño.

- Capítulo Nº 4.- Se da una descripción general del diseño global, se presenta la arquitectura del Prototipo y se realiza un estudio de la implementación.

- Capítulo Nº 5.- Se experimenta el generador diseñado con la creación de una fuente de letras reducida (en este caso romanas), explicando las principales decisiones tomadas y los resultados obtenidos.

- Capítulo Nº 6.- Finalmente, se describen las conclusiones obtenidas como resultado del trabajo realizado, y el trabajo futuro.

Al final de este documento se encuentran la Bibliografía y los Anexos citados a lo largo del mismo.

Índice

Capítulo Nº 1
Introducción

La generación de fuentes de letras mediante una descripción matemática fue tratada por primera vez en el siglo XV; luego se volvió más popular en los siglos XVI y XVII. Más tarde, en el siglo XVIII, fue dejada de lado debido a la poca practicidad que tenía para la época. Durante el siglo XX se volvió a utilizar, ya que los adelantos en la computación y la matemática, permitían realizar una gran cantidad de cálculos en poco tiempo.

Inicialmente estas descripciones matemáticas estaban orientadas a la utilización de la pantalla como salida para el usuario. Más tarde, sin embargo, la matemática y la ciencia de la computación comenzaron a ser relevantes para la tipografía al ir surgiendo equipos de impresión modernos, que mediante líneas de rastreo especifican la posición de la tinta utilizando patrones de ceros y unos de una forma discreta con gran precisión. Ahora es posible dar una descripción precisa de las formas de las letras que producen resultados equivalentes en todas las máquinas basadas en líneas de rastreo. Más aún, las formas pueden ser definidas en términos de parámetros, permitiendo así que se puedan "dibujar" nuevos tipos de caracteres en segundos; lo cual brinda la posibilidad a los diseñadores de realizar experimentos que antes eran impensados.

El presente proyecto "Tipografía Digital en C5" se enmarca dentro de la materia Proyecto de Grado del Instituto de Computación de la Facultad de Ingeniería de la Universidad de la República Oriental del Uruguay.

C5 [JJC02] es un súper conjunto del lenguaje de programación C [D&D94] [TIC00]. En este proyecto nos centraremos principalmente en su biblioteca gráfica, la cual define un lenguaje constructivo para la descripción de páginas, llamado Lenguaje de Ventanas Tipadas. Los lenguajes que definen la descripción de una página son conocidos como Lenguajes Descriptivos de Página (PDL).

La utilización de fuentes de letras es imprescindible en cualquier lenguaje descriptivo de página y en ellos las fuentes de letras son objetos especiales y su utilización suele ser a través de librerías, con el objetivo de facilitar su manejo.

Este Proyecto consiste en la investigación de los factores involucrados en el diseño de fuentes de letras; y este análisis debe materializarse con la creación de un prototipo de generador de fuentes de letras para la biblioteca gráfica del lenguaje de programación C5.

1.1 Motivación y objetivos

La utilización de fuentes de letras es imprescindible en cualquier lenguaje descriptivo de página y actualmente el generador de fuentes de C5 no es capaz de representar las letras minúsculas. Esto sucede debido a que el generador no posee un manejo adecuado de curvas.

Tampoco resulta sencillo diseñar las letras mayúsculas y se debe tener mucha precisión en el momento de definirlas, puesto que estas se forman como la composición de curvas, entre las cuales tiene que haber una relación muy precisa entre los puntos de contacto y la continuidad.

Si bien se podría realizar un diseño de muchas de las letras minúsculas u otros caracteres especiales, lograr parametrizar sus propiedades sería algo extremadamente complejo.

Por lo tanto, sería deseable lograr construir un generador de fuentes de letras que sea lo bastante flexible como para poder representar una gran cantidad de formas, pero que además esto no resultara en un esfuerzo importante por parte del diseñador.

Es entonces que el presente proyecto tiene como objetivo principal el estudio de la viabilidad de la construcción de un generador de fuentes de letras para C5, que permita además de lograr una parametrización razonable, realizar una definición lo más natural posible por parte del diseñador, que permita describir el conjunto de caracteres.

Para evaluar los resultados de dicha investigación es que se deberá diseñar y construir un prototipo del generador de fonts, el cual deberá generar los distintos caracteres a partir de las definiciones con las que luego se experimentará.

1.2 Alcance del Proyecto

El objetivo de este proyecto, como se mencionó en el punto anterior, es investigar el problema de la generación de fonts para la biblioteca gráfica del lenguaje de programación C5. Es por esto que el presente trabajo tiene un gran componente de investigación del dominio del problema. De esta forma los resultados de esta investigación constituirán las bases para el posterior diseño un generador de fonts que cumpla con los objetivos planteados.

Entonces, con la construcción de un prototipo experimental que será evaluado a lo largo del proyecto podremos experimentar el diseño de un subconjunto de una fuente de letras, y con los resultados obtenidos y el análisis previo se obtendrán las conclusiones generales del proyecto.

La experimentación deberá cubrir un subconjunto representativo de las letras del alfabeto romano. La fuente de letras que se utilizará para esta prueba es el de las letras romanas, y deberá parametrizarse de tal forma que se pueda distinguir razonablemente entre los tipos de letra: Arial, Courier, Roman y Gothic.

1.3 Metodología de trabajo

El trabajo para este proyecto se dividió en tres etapas bien diferenciadas:

- **Investigación y análisis.** – Se basa principalmente en la evaluación de los requerimientos del proyecto y en el estudio del estado del arte de la tipografía digital, así como también en el lenguaje de programación C5 y en el movimiento artístico "Universalismo constructivo", desarrollado por el pintor uruguayo Joaquín Torres García.

- **Diseño y construcción.** – En esta etapa se diseñó el generador de fuentes de letras y se construyó un prototipo experimental, con la finalidad de evaluar los resultados del diseño.

- **Evaluación y documentación de los resultados.** – Finalmente, se evaluaron los resultados obtenidos del prototipo y se elaboró el informe final.

Cada una de estas etapas contenía un conjunto de actividades de las cuales se mencionan, a continuación, las principales involucradas en el desarrollo del Proyecto:

- Estudio del estado del arte.
- Aprendizaje de C5.
- Definición de la arquitectura general.
- Implementación del prototipo.
- Evaluación primaria de los resultados.
- Experimentación del prototipo.
- Evaluación de los resultados.
- Elaboración del informe final.

En el siguiente diagrama de Gantt se muestra la distribución, a lo largo del proyecto, de las actividades recién mencionadas.

Id.	Nombre de tarea	Inicio	Fin	Duración
1	Estudio del estado del arte	03/11/2003	31/12/2003	8.60s
2	Aprendizaje de C5	01/12/2003	30/01/2004	9s
3	Definición de la arquitectura general	15/01/2004	13/02/2004	4.40s
4	Implementación del prototipo	16/02/2004	30/04/2004	11s
5	Evaluación primaria de los resultados	12/04/2004	30/04/2004	3s
6	Experimentación del prototipo	03/05/2004	31/05/2004	4.20s
7	Evaluación de los resultados	17/05/2004	15/06/2004	4.40s
8	Elaboración del informe final	17/05/2004	12/07/2004	8.20s

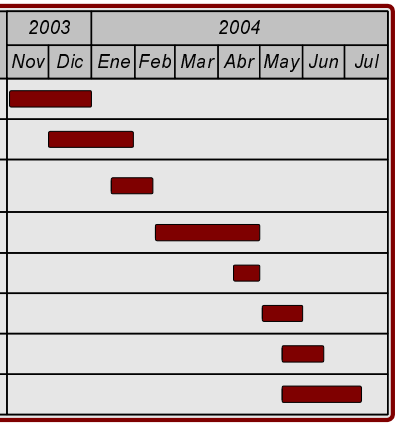


Figura N° 1 – Distribución de actividades

Capítulo Nº 2
Presentación del Problema

Como ya se mencionó, la utilización de fonts en los Lenguajes Descriptivos de Página es fundamental. El concepto de PDL fue introducido por la empresa Adobe con el fin de diferenciar su lenguaje PostScript [PSRM90] del resto de lenguajes, que hasta ese momento basaban su soporte gráfico en el uso de bibliotecas, ya que éste utiliza los comandos gráficos como propios del lenguaje.

En este Proyecto se debe utilizar C5, que es una variante de los Lenguajes Descriptivos de Página que utiliza la idea del Constructivismo Universal, desarrollada por el pintor uruguayo Joaquín Torres García; el cual se ha denominado Lenguaje de Ventanas Tipadas.

En estos lenguajes las fuentes de letras son objetos especiales y suelen utilizarse a través de librerías, con el objetivo de facilitar su manejo.

Muchas técnicas se han utilizado en la representación de letras, desde un almacenamiento matricial hasta una descripción de las curvas que definen cada una de sus partes. Sin embargo, estas no han sido capaces de utilizar una parametrización, que permita a los usuarios seleccionar un tipo de la fuente a través de valores para el conjunto de parámetros de la misma; si no que para cada tipo de letra se tiene que definir una nueva fuente, aunque la diferencia entre ellos se mínima.

Esto llevó a que se haya desarrollado hace algunos años, en la materia Proyecto de Grado, un prototipo de generador de fonts, del cual C5 basa actualmente la representación de las fuentes de letras. Si bien se pudo solucionar el problema de la parametrización para las letras mayúsculas, esto se hizo a un costo en el diseño tal que la representación de las letras minúsculas resulta excesivamente compleja, dado que tampoco ofrece un manejo aceptable para las curvas.

En este proyecto se busca investigar el ambiente del diseño de fonts y lograr definir un prototipo de generador de fuentes de letras que resuelva los problemas que en la actualidad tiene C5 para el manejo de las mismas.

Se pretende obtener un generador de fonts que sea lo suficientemente parametrizable como para que la persona que utiliza C5 en el diseño gráfico, pueda elegir por sí misma el tipo de letra que prefiere, al escoger los valores que crea convenientes para los parámetros.

2.1 Estado del arte

En esta sección se presenta la investigación del estado del arte de la tipografía digital, describiendo las características de las principales herramientas para el diseño de fonts.

El primer lenguaje que se presenta es METAFONT [TMFB], el cual representa el primer estudio serio de la tipografía digital moderna, dejando de lado la vieja representación vectorial. Luego se mencionan la principales características del lenguaje gráfico PostScript [PSRM90], el cual define la representación de fuentes de letras como si se tratase de cualquier otro elemento gráfico. Seguidamente, se presenta una breve introducción a las fuentes de tipo TrueType [TTF], las cuales son ampliamente difundidas en los productos de la empresa Microsoft. Y finalmente se describe la forma en que se maneja el actual generador de fonts de C5.

2.1.1 MetaFont

METAFONT [TMFB], creado por Donald E. Knuth [DTDKn], es un lenguaje de computación que permite producir fuentes de símbolos utilizando declaraciones matemáticas de tipos.

En este lenguaje, el diseñador debe escribir un programa para cada letra o símbolo; estos programas varían sensiblemente de los programas comunes de computación, puesto que son esencialmente declarativos en lugar de imperativos.

METAFONT brinda las herramientas necesarias para la descripción de símbolos siguiendo el modelo del pintor, donde se utiliza la idea tradicional de dibujar utilizando un lápiz. Para describir una forma se define el formato del lápiz, y un conjunto de líneas que éste deberá seguir, las cuales deben definirse considerando que se mantengan o no las continuidades geométricas, G^0 y G^1, entre ellas.

Si dos segmentos de curva se unen en uno de sus extremos, la curva tiene continuidad geométrica G^0. Si además, las direcciones de los vectores tangente son iguales en el punto de unión, la curva tiene continuidad geométrica G^1.

Para describir estas líneas es que se utiliza un tipo de curvas cúbicas paramétricas llamadas curvas de Bézier.

Curva de Bézier

La idea básica de una curva de Bézier es la de definir una curva a través de cuatro puntos (z_1, z_2, z_3, z_4). Estos puntos determinan la forma y posición de la curva de la siguiente manera: los puntos z_1 y z_4 son los puntos de contacto, empezando la curva en el punto z_1 y finalizando en z_4; los puntos z_2 y z_3 son los puntos de control, los cuales definen las direcciones de la curva en los puntos z_1 y z_4 respectivamente, como se muestra en la figura N° 2.1.

7

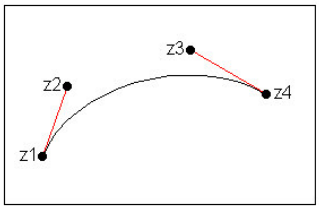


Figura 2.1 – Curva de Bézier

Las curvas de Bézier son también llamadas curvas de punto medio, debido a que su forma puede ser obtenida a través del siguiente procedimiento:

Inicialmente se tienen los cuatro puntos que definen la curva (z_1, z_2, z_3, z_4), y se obtienen los tres puntos medios (z_{12}, z_{23}, z_{34}), como $z_{12} = ½ [z_1, z_2]$, $z_{23} = ½[z_2, z_3]$, $z_{34} = ½[z_3, z_4]$ (figura 2.2 (a)).

Luego, se obtienen los puntos medios de segundo orden $z_{123} = ½[z_{12}, z_{23}]$ y $z_{234} = ½[z_{23}, z_{34}]$. Finalmente se construye el tercer punto medio $z_{1234} = ½[z_{123}, z_{234}]$ (figura 2.2 (b)).

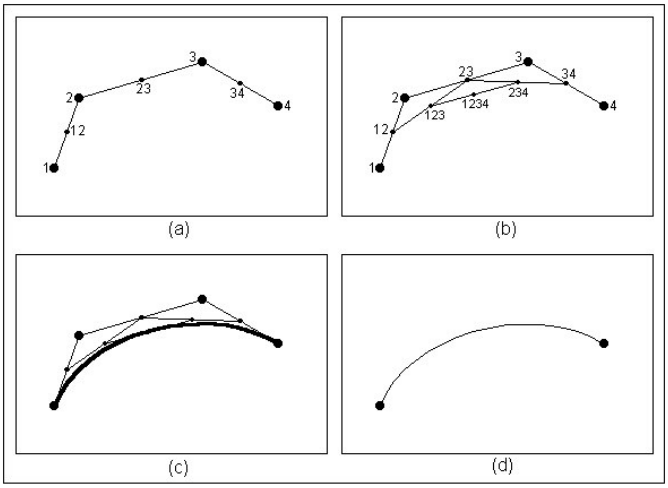


Figura 2.2 – Curva de Bézier como curva de punto medio

Este punto, z_{1234}, es uno de los puntos de la curva determinada por (z_1, z_2, z_3, z_4). Para obtener el resto de los puntos se debe repetir el mismo procedimiento para las curvas definidas por los puntos (z_1, z_{12}, z_{123}, z_{1234}) y (z_{123}, z_{234}, z_{34}, z_4), y así infinitamente. En la figura 2.2 (c) se puede ver como el punto finalmente obtenido pertenece a la curva y en la figura 2.2 (d) la curva final.

Este método converge rápidamente a la curva definida por estos puntos y aunque no es utilizado para desplegar la curva, brinda una idea intuitiva que permite entender su comportamiento. Sin embargo, la idea principal que se debe tener de las curvas de Bézier es que interpolan dos puntos, pudiendo determinar la direcciones y magnitudes de las pendientes en estos de una forma natural. Si el lector quiere tener un conocimiento mayor de las curvas de Bézier, puede ver una descripción de su definición matemática en el anexo [curvas cúbicas paramétricas].

Metafont realiza la definición de símbolos, como una secuencia de curvas de Bézier y rectas. Las curvas son definidas mediante cuatro puntos entre paréntesis rectos [z_1, z_2, z_3, z_4], mientras que las rectas están definidas por dos puntos entre paréntesis rectos [z_1, z_2].

El diseñador tiene la posibilidad de describir la forma del pincel que imaginariamente dibuja los trazos que él indica en sus declaraciones. En la figura N° 2.3 se muestra la definición de una curva cerrada, realizada utilizando la forma de pincel que se muestra en la figura y haciéndola pasar por el contorno de una elipse, produciendo la elipse distorsionada en función de la forma del pincel. Este tipo de representación, que se obtiene al barrer un objeto a lo largo de una trayectoria, recibe el nombre de *Representación de barrido* y es ampliamente utilizada en el modelado de sólidos, estando íntimamente relacionado con el modelo del pintor.

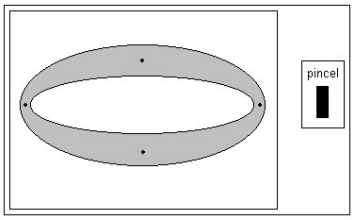


Figura N° 2.3 – Modelo del Pintor

Para lograr representar con fidelidad un diseño artístico con METAFONT, se deben poseer dos cualidades: 1) una gran capacidad de razonamiento abstracto y 2) una gran capacidad para expresar ideas intuitivas en términos formales.

En este lenguaje, uno explica dónde estarán posicionados los principales componentes de una forma y cómo se deben relacionar unos con otros, pero uno no debe manejar los detalles exactos, si no que la computadora realiza el trabajo de resolver las ecuaciones que se deducen de las especificaciones realizadas por el diseñador.

Las fuentes definidas en Metafont deben ser compiladas para luego poder ser utilizadas en algún programa de representación gráfica. Es por este motivo es que su utilización se realiza a través de librerías de tipos de letras. Uno de los lenguajes de representación gráfica más importantes que utilizan Metafont es LaTex, el cual es ampliamente utilizado en la notación matemática debido al gran poder de definición de letras que brinda Metafont.

2.1.2 PostScript

Este lenguaje de Programación fue el que introdujo el concepto de Lenguaje Descriptor de Página (PDL), con el objetivo de la empresa Adobe (su creadora) de diferenciar su nuevo lenguaje del resto. Los lenguajes de programación existentes hasta ese momento basaban su soporte gráfico en el uso de bibliotecas, mientras que PostScript tuvo desde su comienzo la expresividad gráfica como comandos propios del lenguaje.

De forma parecida a la que MetaFont sigue el modelo del pintor, PostScript permite dibujar fácilmente composiciones de líneas, aunque sin resolver directamente el problema de la continuidad G^1 entre ellas. Existen dos versiones para cada primitiva, una que utiliza coordenadas absolutas, definiendo partes independientes dentro de un símbolo, y otra que utiliza coordenadas relativas, permitiendo definir fácilmente curvas con continuidad G^0.

Entre }las herramientas gráficas que brinda PostScript, se cuenta con funciones para definir segmentos de recta, elipses y curvas de Bézier. Estas funciones permiten definir líneas o curvas cerradas, pudiéndose estas “pintar” o rellenar respectivamente, escogiendo los colores que se usarán en cada caso.

Utilizando las herramientas gráficas generales es que se define un font; cada font es un programa ejecutable que “dibuja” las formas de un caracter utilizando las mismas herramientas que para cualquier otro elemento gráfico. Por lo tanto, el texto es pensado como un gráfico más, y puede ser transformado de la misma forma.

Las fonts pueden ser definidos a partir de las primitivas gráficas disponibles. Una vez que un programa que “dibuja” una forma particular es creado, su código puede ser agregado en el diccionario de fonts y luego ubicado en la página con los mecanismos de manejo de texto.

Con la utilización de las fuentes de letras como descripciones gráficas es que se puede realizar una integración completa del texto y los gráficos en una página. Esto significa que una página puede ser escalada o rotada por cualquier ángulo, ya que la página está completamente integrada como una única entidad gráfica.

Con PostScript no es posible que el usuario escoja un tipo de la fuente a través de un conjunto de parámetros, sino que debe seleccionar un tipo de los ya definidos. Por lo tanto, cada fuente de letras define un único tipo, debiendo ser compilada cada una de ellas antes de ser utilizada junto con el programa.

2.1.3 TrueType

En los últimos años de la década de los 80, la mayoría de las personas del mundo de las computadoras personales estaban seguras que la tecnología del manejo de fonts sería una parte importante de los sistemas operativos futuros. Por este motivo es que Adobe trataba de venderle la licencia de su programa *PostScript* a Apple y Microsoft, con el propósito de mantener su hegemonía sobre el diseño de fonts; sin embargo, estas dos empresas pretendían, naturalmente, tener un control total sobre las partes claves de sus sistemas operativos, y no desprenderse de la gran cantidad de dinero que deberían pagarle a Adobe por sus servicios. Además, Apple se había irritado cuando Adobe licenció PostScript para el manejo de impresoras, ya que esta lo utilizaba en su propia impresora LaserWriter. Todo esto llevó a que Apple y Micrsoft se pusieran de acuerdo en desarrollar un sistema de generación de fuentes de letras que finalmente estuviese disponible para las dos compañías, pudiendo Microsoft utilizar un estilo gráfico similar a PostScript en su tabla de caracteres (TrueImage), mientras que Apple podía utilizar un sistema de fuentes de letras, tal vez mejor que el de Adobe.

Así es que surgió el lenguaje de definición de fuentes de letras TrueType, que si bien utiliza el modelo del pintor, al igual que Metafont y PostScript, presenta además ciertas complejidades de diseño gráfico.

En TrueType, un font se define a través de los contornos internos y externos, los cuales identifican de que lado está la letra. Estos contornos reciben el nombre glyph. Cada glyph es representado por una curva cerrada definida por una secuencia de curvas de Bezier de 2, 3 o 4 puntos. Las curvas de 3 puntos son idénticas a las de 4, ya que su punto de control se comporta de la misma forma que lo hacen los dos puntos de control de la curva de 4 puntos y finalmente la curva de dos puntos simplemente interpola los dos puntos de contacto con el segmento de recta que los une. En la siguiente figura se puede apreciar una definición de la letra “B”, utilizando tres glyphs, donde los puntos de contacto se representan con círculos rellenos, mientras que los de control se muestran vacíos.

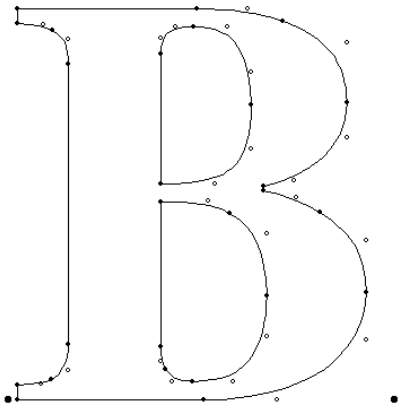


Figura 2.4 – Glyphs para la letra B

Si bien TrueType presenta varias ventajas con la utilización de glyphs, como la de poder utilizar patrones sobre las letras, no puede parametrizarse cada fuente de letras y el usuario simplemente tiene la opción de elegir el tipo que prefiere de un conjunto de tipos de letras preestablecido.

Al igual que en el caso de Metafont y PostSript, con TrueType no se puede elegir} un tipo de letra de una fuente, si no que directamente cada fuente de letras define un único tipo.

2.1.4 GenFont, el generador de fuentes de C5

GenFont es el generador de fuentes de letras de C5 y un primer prototipo de éste fue generado por Daniel López y Fabricio Pirez, como parte de la materia Taller V del año 1999. Este generador fue construido utilizando una aproximación por niveles, tratando de reconocer en un primer nivel las estructuras básicas de cada letra y dividiéndola a ésta en los sectores en los que se representará cada parte. En un segundo nivel se toman los sectores generados en el primero, y dependiendo de la letra y del sector de esta al que pertenecen, se aplican las funciones apropiadas para representarla. Finalmente, en el tercer nivel es donde se realiza la representación gráfica de los detalles, refiriéndose el término detalle a aquellas partes de la letra que no son esenciales en la forma de la misma, si no que la adornan.

Con este generador, la diferencia entre tipos de letras para una misma fuente se basa en el tipo de detalle utilizado para cada uno de ellos y en los anchos de cada parte de la letra.

Este generador cumplió con el objetivo de poder parametrizar un conjunto representativo de letras mayúsculas para la fuente romana, el cual luego fue extendido para cubrir completamente las mayúsculas de la fuente. Si bien se logró este objetivo, el costo de realizar cada diseño fue muy grande. A modo de ejemplo mencionamos que la definición de la letra “S” llevó no menos de dos semanas. Esto se debe a que este generador presenta una gran limitación en el manejo de curvas. Además, al separar el tercer nivel para los detalles, hace que agregar un detalle a una letra sea extremadamente complicado, ya que éstos deben adaptarse a cada parte donde son definidos.

El principal motivo por el cual se hace prácticamente imposible definir las letras minúsculas es que no se proveen las herramientas necesarias y suficientemente flexibles como para poder definir curvas. Si bien se brindan funciones de representación de curvas, éstas no solucionan problemas como la continuidad G^0 o G^1, si no que es el propio diseñador de fonts quien debe utilizar las funciones de tal forma que estas produzcan los resultados deseados. Por lo tanto, el diseñador de fonts debe tener un conocimiento matemático apreciable para poder definirlas correctamente.

Cada parte de la letra es representada por la zona que se encuentra entre los valores de dos funciones que utilizan ejes paralelos a la pantalla. Una de estas funciones define el valor máximo que tienen todos los puntos de cada recta vertical y la otra los valores mínimos, como se muestra en la figura N° 2.5. El puerto o rectángulo en el que se está definiendo una parte de la letra, puede ser rotado 90° de forma tal de poder representar funciones cuyo dominio esté en un eje vertical y no horizontal. En esta figura puede verse que en el caso de querer agregar continuidad entre segmentos de curvas, se debe resolver la complicada tarea de igualar los puntos de contacto y calcular las pendientes de cada parte, pudiéndose dar casos de pendientes infinitas.

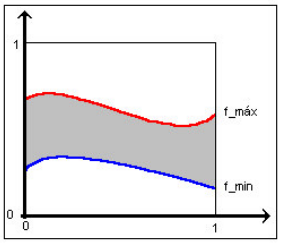


Figura N° 2.5 – zona entre funciones

En la siguiente figura se muestra un ejemplo de cómo funciona el generador para la letra A. Inicialmente se divide el puerto en el que será representada la letra, en este caso en tres zonas, definidas cada una por un rectángulo distinto. El rectángulo rojo representa la zona en la cual deberá estar el segmento de recta que comienza en el extremo inferior izquierdo, mientras que el rectángulo azul representa el port del segmento de recta que se inicia en el extremo inferior derecho; finalmente, el rectángulo gris representa la zona en la cual se dibujará el segmento de recta horizontal. Las flechas que se muestran en cada zona, representan la orientación del eje x para las funciones que determinan el área que debe representarse en cada una.

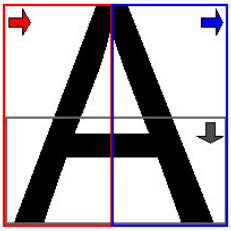


Figura N° 2.6 – Definición de la letra A utilizando GenFont

Luego, en el segundo nivel de depuración, se toman los ports generados en el primero y se realiza la representación de cada parte de la letra utilizando las funciones máximo y mínimo, tomando en cuenta desde este momento los criterios de parametrización de la fuente. Finalmente, en el tercer nivel se realiza la representación gráfica de los detalles, en la cual hay que tener mucho cuidado en la continuidad y en los tipos de detalle; ésta es la tarea más complicada del diseño, dado que deben utilizarse muchos conceptos matemáticos con gran precisión.

Entre los problemas que se tienen con este método tenemos que en cada zona sólo pueden definirse figuras que puedan ser representadas por funciones, resultando la definición de fonts en una tarea bastante compleja y debiendo tener, el diseñador, un conocimiento matemático tal que le permita entender los conceptos de continuidad y poder utilizarlos adecuadamente.

La gran ventaja que ofrece GenFont frente al resto de programas de generación de fuentes de letras, es que en éste cada fuente de letras define una infinidad de tipos, de los cuales el usuario selecciona uno de ellos a través de la utilización de parámetros que modelan el comportamiento de la letra.

2.2 Herramienta de Desarrollo

En esta sección se describen las principales características del lenguaje de programación C5.

C5 es un súper conjunto del lenguaje de programación C. Las extensiones introducidas en C5 son la noción de Par de Tipo Dependiente (DPT) y la Expresión de Inicialización de Tipo (TIE).

Un DPT es un caso particular de un registro de tipos dependientes de la forma genérica:

$\{ T_1() L_1;\ \ T_2(L_1)\ L_2; \ldots ; T_n(L_1, L_2, \ldots ; L_{n-1})\ L_n; \}$

donde Li es la etiqueta y $Ti(L_1, L_2, \ldots, L_{i-1})$ el tipo del i-ésimo campo del registro.

Un DPT es un registro de dos campos, de la forma:

$\{Ctype\ L_1; L_1\ L_2; \},$

dónde Ctype es el tipo de todos los tipos de C y L_1 es entendido como un valor para el primer miembro del par y como un tipo para el segundo.

En C5, un DPT es implementado como un tipo de datos abstracto con dos funciones constructoras llamadas DT_pair y C5_gos y un conjunto de funcionas selectoras. Estas funciones conforman la librería DPT de C5. El compilador de C5 asegura que los DPT´s están bien formados chequeando la construcción de los tipos DPT.

A continuación se muestra un ejemplo de la utilización de DPT´s, donde se utiliza la redefinición de la función de C printf a su par de C5 (C5_printf) para la cual se realiza un chequeo de tipos en tiempo de compilación:

```
typedef char String[5];
typedef float Fnr;

main() {
        String st = "coef";
        Fnr n = 42.56;
        C5_printf(DT_pair(String,st));
        C5_printf(DT_pair(Fnr, n));
}
```

La Expresión de Inicialización de Tipo (TIE) puede ser agregada a un tipo. Una TIE es una expresión constante. Por ejemplo, esta no puede ser modificada en tiempo de ejecución.

La sintaxis de una TIE es una secuencia, separada por comas, de expresiones constantes encerradas entre corchetes. Una expresión constante de C es una expresión aritmética construida por enteros, números de punto flotante y caracteres.

Existe una regla sintáctica para insertar TIEs en una declaración de tipo: una TIE es posicionada a la derecha del tipo relacionado. El siguiente ejemplo muestra dos definiciones de tipo con TIEs:

```
DT_typedef int {1} Numbers [10] {2}  [20] {3};
DT_typedef struct [
        Numbers [4] nrs;
        char{5} * {6} String-ptr;
        } {7} Rcrd;
```

En la primera definición de tipo, la TIE {1} es adjuntada a un tipo entero y las TIEs {2} y {3} son adjuntadas a un arreglo doble. En la segunda definición, las TIEs {4}, {5}, {6} y {7} son adjuntadas a los tipos Numbers, char, puntero a caracter y struct respectivamente.

Con los DPTs y las TIEs se puede agregar información en la definición de tipos y luego obtenerla del DPT mediante un conjunto de funciones proveídas por C5. Por ejemplo, podría utilizarse como primer miembro de un DPT el tipo "*fuente romana de caracteres*", habiendo utilizado una TIE en su definición, mediante la cual se ingresan los parámetros que esta tendrá, determinando, de esta forma, un tipo dentro de la fuente. La instancia de este tipo está determinada por el segundo miembro del DPT, pudiéndose obtener de esta forma el caracter y su tipo.

2.2.1 Biblioteca gráfica

La biblioteca gráfica de C5 se basa en las ideas del Constructivismo Universal, desarrollado por Joaquín Torres García. De esta forma es que se interpreta una imagen como un conjunto de planos de colores, seleccionando de cada uno de estos los sectores que se precisan y descartando el resto. Cada región seleccionada es definida como un Port. Un Port es un rectángulo con lados horizontales y verticales, con una dimensión determinada y con una lista de colores especificada.

La biblioteca gráfica de C5 utiliza, para brindar sus funcionalidades, la Oriented Port Machine, la cual consiste en un conjunto de operaciones para la manipulación de listas de Ports orientados.

Un port orientado es más que una región rectangular de la página; los atributos de un port orientado son las coordenadas, la lista de colores y la orientación de la región rectangular. Existen cuatro orientaciones posibles, Right, Down, Left y Up. El color actual de un port es el primer elemento de la lista de colores. Si la lista de colores está vacía, entonces el color por defecto es Blanco.

La Oriented Port Machine es una librería de C5 basada en el tipo de datos abstractos Port_List (Lista de puertos), que tiene las siguientes funciones:

- Port_List opm_null()
  Esta función retorna una lista de ports vacía.

- Port_List opm_page(Color_List cl)
  Esta función devuelve una lista de ports de orientación right, que tiene el tamaño de la página y la lista de colores cl.

- Port_List opm_inters(Port_List pl1, Port_List pl2)
  Esta función devuelve una lista de ports construida a partir del producto cruz de las intersecciones de las listas pl1 y pl2.

- Port_List opm_rot(int rot_nr, Port_List pl)
  Esta función aplica la rotación a cada puerto de la lista pl.

  La orientación de los puertos es rotada nr veces, de acuerdo a la siguiente regla: rotate(Right) = Down, rotate(Down) = Left, rotate(Left) = Up, rotate(Up) = Right.

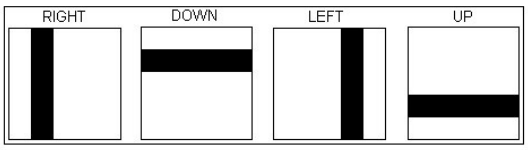


Figura N° 2.7 – Función sel_split para n=5 e i=2

- Port_List opm_selsplit(int n, int i, Port_List pl)
  Esta función aplica la función sel_split a cada uno de los ports de la lista pl.

  Si n>0 e i varía en el rango {1, n} entonces la función

  Port sel_split(int n, int i, Port p),

  parte el port en n sub-rectángulos y devuelve un port con las coordenadas del i-ésimo sub-rectángulo. La orientación y el color son tomados del puerto p. En la figura N° 2.7 se muestran los resultados de esta función, al ser aplicada a cuatro ports con las mismas dimensiones y de distintas orientaciones. Si el valor de *i* está fuera del rango [0, 1], entonces la función devuelve un port vacía.

  Finalmente, en el caso que n sea no positiva (n<=0), la función devolverá un port igual a p para cada valor de i.

- Port_List opm_partition(float f, Port_List pl)
  Esta función aplica la función partition a cada puerto de la lista pl.

  Si 0<f<1 entonces la función divide el port en dos rectángulos proporcionales a f y devuelve un port que representa el primer sub-rectángulo.

  La orientación y el color son tomados del port p. La figura N° 2.8 muestra los cuatro resultados diferentes dependiendo de las cuatro orientaciones posibles de p, identificando en negro cada uno de los ports finales.
  .

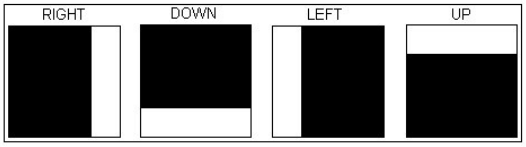


Figura N° 2.8 – Ejemplo para la función opm_partition, con f = 0.75

- Port_List opm_set_color(int n, Port_List pl)
  Esta función aplica la función drop_color a cada una de los ports de la lista pl.
  La función Port drop_color(int n, Port p) devuelve un port con la lista de colores p, descartando los primeros n elementos si n>0. Los otros atributos del port retornado son iguales a los de p.

Con estas funciones básicas se construyen las funciones gráficas para la Orient Port Machine, conformando la librería gráfica de C5.

2.3 Torres García y el Universalismo Constructivo

Joaquín Torres García [GPL92], nacido en Montevideo el 28 de julio de 1874, fue uno de los pintores más destacados del Uruguay y padre del Universalismo Constructivo. El Universalismo Constructivo es un arte de gran contenido ideológico, el cual aspira a dar una visión unitaria del Mundo por medio de una estructura rígida y de un esquematismo formal y colorístico, sin que esto incida en la abstracción total.

Este movimiento artístico representa una obra como un conjunto de estructuras geométricas y colores que unidos forman "ideogramas" [TG69]. En este contexto es que una obra está compuesta por ideogramas, donde cada uno define una parte específica del todo que representa la obra.

Como puede verse en la figura N° 2.9, cada ideograma define una unidad lógica dentro de la obra y la ubicación de cada uno y sus relaciones con los demás dan una visión unitaria de la misma.

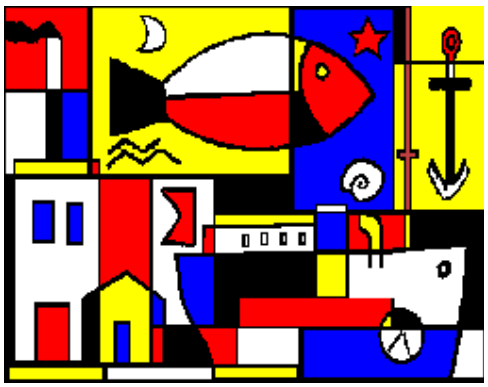


Figura N° 2.9 – Torres García

El nombre "Universalismo Constructivo" pretende dar una visión constructiva de una obra, viéndola como un universo en el que intervienen varios elementos que deben ser ubicados en base a una estructura rígida. Sin embargo el movimiento del universalismo constructivo va más allá de una simple pintura y pretende dar una visión de estructura y de universalidad como los dos pilares sobre lo que todo debe apoyarse, tanto se hable de un edificio, como de un partitura o de una pintura, y lo mismo si se tratase de cualquier otro problema; ya que a través de todas esas cosas sólo se pretende llegar a la esencia universal en la cual se identifican la Regla Constructiva y el Universo.

2.4 Visión Global del Problema

Dada una idea artística por parte de un diseñador de fonts, el problema que se intenta resolver es el de brindarle la mayor flexibilidad y control posible, para que éste pueda especificar con gran naturalidad la forma y comportamiento de su diseño.

Por lo tanto éste debería poder definir las propiedades y el comportamiento de cada parte de la letra, y la forma en que cada una de estas se relacionan.

En una primera etapa debería poder identificar cada parte de la letra teniendo en cuenta las formas de representación que ofrece el generador. Luego, en una segunda etapa, determinar aquellas partes que parametrizará y finalmente definir, en el lenguaje del generador, cada una de estas partes y la forma en que se relacionan, como se muestra en la figura N° 2.10.

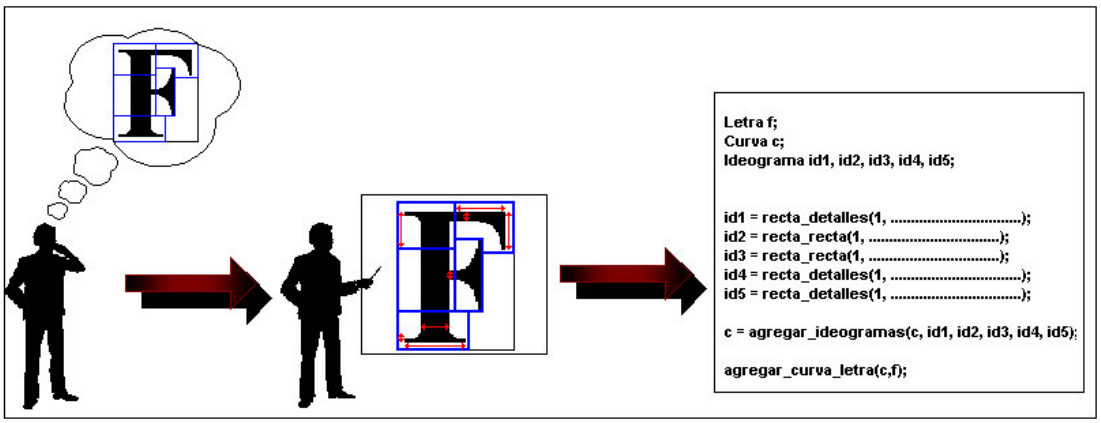


Figura N° 2.10 – Proceso de definición de fuentes

Todo esto plantea una serie de problemas que deberán resolverse:

- ***Determinar un conjunto representativo de formas.***
  Debe brindársele al diseñador un conjunto de representaciones que le permita describir cada parte de la letra.

- ***Especificar las relaciones entre las distintas partes de la letra.***
  El diseñador debe poder establecer las relaciones que tienen los distintos componentes de la letra entre sí.

- ***Permitir continuidad.***
  Se debe proveer la facilidad de establecer continuidad entre los distintos componentes de la letra sin que esto sea una tarea difícil para el diseñador.

- ***Permitir una fácil parametrización.***
  Las herramientas que se le provean al diseñador deben permitirle una fácil parametrización de los elementos que define.

- ***Abstraerle las nociones matemáticas al diseñador.***
  No debe asumirse que el diseñador tenga un gran conocimiento matemático en el momento de definir un font. Éste debe poder realizar sus diseños de la forma más natural posible.

Capítulo Nº 3
Descripción de la Solución

En este capítulo se presentan los principales aspectos del diseño de un generador de letras para el presente Proyecto.

Dicha construcción se basa en las ideas fundamentales detectadas en la etapa de análisis, las cuales se especifican al final del capítulo anterior.

3.1 Idea general de la Solución

Con la idea de facilitar el trabajo de la especificación de fonts, es que la presente solución se basa en las ideas constructivas que permiten una descripción más natural de la letra. De esta forma, se busca que la definición de fuentes de letras sea vista como la composición de figuras prediseñadas que puedan ser modificadas al gusto del diseñador. Para realizar la composición de estas figuras se necesitan operadores que permitan mantener relaciones entre ellas. Por ejemplo, en la figura N° 3.1 se puede ver la letra "a", compuesta por curvas (en color gris) y rectas con salidas curvas (en color rojo y naranja), las cuales definen relaciones de continuidad entre ellas.

Figura N° 3.1 – Descomposición en ideogramas

A las figuras prediseñadas, que son los componentes constructivos, se les puso el nombre de "*Ideogramas*", ya que su idea original se inspira en el movimiento artístico "Universalismo Constructivo", desarrollado por Joaquín Torres García. Estos ideogramas representan figuras sobre rectángulos paralelos a la pantalla, llamados puertos, y sólo puede haber una relación entre aquellos ideogramas contenidos en puertos adyacentes, ya que las relaciones son de continuidad.

Puerto

Un puerto es un área rectangular orientada de lados horizontales y verticales, donde se representa una parte de la letra. Cada puerto está determinado por las coordenadas de su vértice inferior izquierdo, sus dimensiones y su orientación, en función del rectángulo global en el que debe representarse la letra. En figura N° 3.2 se puede apreciar el área donde debe describirse la letra, en color negro, y la definición de un puerto perteneciente a éste.

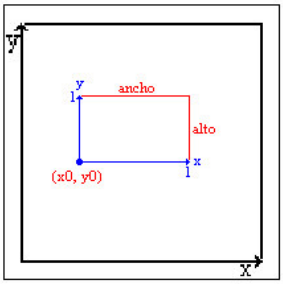


Figura N° 3.2 – Definición de un puerto

La definición de un nuevo puerto tiene sentido, sólo si un ideograma será definido sobre él y las relaciones entre ideogramas se dan sobre los segmentos de recta compartidos entre los puertos que los contienen.

Ideograma

El término Ideograma, en el contexto del diseño de fonts, refiere a los elementos constructivos para la composición de letras. Cada uno debe tener un puerto asociado sobre el cual debe representarse, y un conjunto de parámetros con los cuales el diseñador pueda controlar su comportamiento. Además, cada ideograma debe definir con precisión las posiciones sobre las cuales puede relacionarse con otros y cuáles deben ser las características de estos relacionamientos.

En la figura N° 3.3 se pueden observar los dos puertos superiores de la letra “a” y los ideogramas que sobre cada uno de ellos se representan. Además puede observarse que la relación se presenta sobre el segmento de recta común que existe entre los bordes adyacentes de cada puerto, y esta relación define una continuidad geométrica G^1.

Figura N° 3.3 – Continuidad entre ideogramas

La forma en que los ideogramas se relacionan es unidireccional, determinando uno de ellos los puntos de contacto y las direcciones en estos, y el otro simplemente adaptándose a ellas. En este caso, el ideograma que pertenece al *Puerto 1* especifica su “salida y el que pertenece al Puerto 2 simplemente se adapta a ella, teniendo en cuenta además los parámetros que utilice el diseñador en su definición.

Esta relación se define por medio de entradas y salidas. Cada ideograma recibe una entrada a la cual adaptarse, que especifica los puntos desde donde tiene que comenzar su diseño y las direcciones iniciales. Además cada ideograma define una cantidad arbitraria de salidas, de las cuales otro ideograma puede seleccionar una como su entrada.

Entrada

Una Entrada es una descripción que se da sobre un puerto y puede o no ser utilizada por el ideograma que se define sobre éste para comenzar su diseño. Esta descripción determina en qué puntos debe comenzar un ideograma y cuáles son las direcciones iniciales que se deben seguir.

Salida

Una *Salida* define una terminal de relación para un ideograma y esta puede o no ser utilizada para construir una relación con otro. Las salidas tienen relevancia en la medida que se establezcan las relaciones de entrada-salida entre dos ideogramas.

Para que una relación definida de esta forma pueda especificarse, los puertos de los dos ideogramas deben ser adyacentes y además la salida del primer ideograma debe estar ubicada en el segmento de recta compartido entre los dos puertos.

Es así que se puede obtener un árbol de ideogramas, donde cada nodo representa un ideograma y cada rama una relación entrada-salida. Al realizar un recorrido en preorden y en cada paso obtener el ideograma determinado y sus salidas, es que se obtiene la figura final. En la figura N° 3.4 se puede observar el árbol de ideogramas que se obtiene en la definición del letra “a”. En este caso, las salidas de los puertos 6 y 8 están definidas perpendiculares a sus respectivos puertos sobre los mismos puntos de contacto, debido a que la representación de árbol no permite definir ciclos. Sin embargo, establecer estas salidas con la misma dirección y en la misma posición puede hacerse con facilidad.

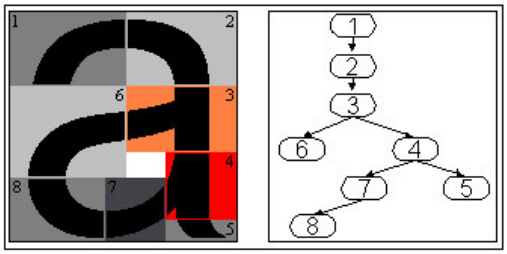


Figura N° 3.4 – Árbol_curva

La entrada del ideograma raíz no proviene de ningún otro ideograma, por lo que debe definirse por separado.

Pudo haberse escogido una representación de grafo en lugar de un árbol; sin embargo, de esta manera no sólo se logra eliminar el problema de ciclos, si no que también se facilita el control del comportamiento del ideograma, puesto que éste sólo debe adaptarse a una única entrada. Más allá de esta entrada, cada ideograma puede utilizar parámetros particulares que permitan controlar su forma.

A estos árboles de ideogramas se les puso el nombre de *árbol_curva*, debido a que las figuras que se obtienen como resultado de procesar el árbol, conforman una curva cerrada.

De esta forma, una letra está definida por un conjunto de *árboles_curva*, donde cada uno determina una curva cerrada específica de la letra. Por ejemplo, en la figura N° 3.5 podemos observar la representación de la letra “i”, en la que se utilizan dos *árboles_curva*.

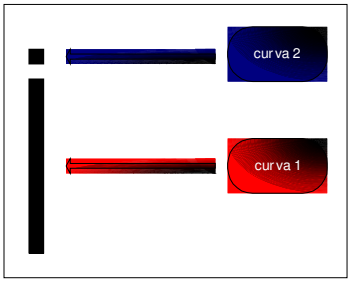


Figura N° 3.5 – Representación de la letra “i” con dos Árboles_curva

En las siguientes secciones se muestran los distintos componentes del diseño y las resoluciones de los problemas que se presentaron durante el mismo.

3.2 Puertos y Sistemas de Coordenadas

La letra es definida sobre un área rectangular y para poder determinar un punto sobre la misma se utiliza un sistema de coordenadas cartesianas, con su origen en el vértice inferior izquierdo y un dominio acotado, dado que las coordenadas x e y varían en el rango [0, 1], de forma tal que el vértice inferior izquierdo tiene las coordenadas (0,0), el vértice inferior derecho (1, 0), el vértice superior derecho (1, 1) y el vértice superior izquierdo (0, 1), tal como se muestra en la figura N° 3.6.

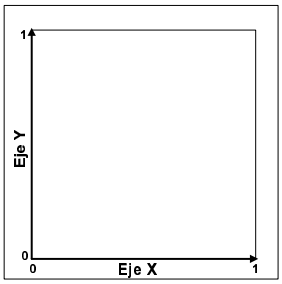


Figura N° 3.6 – Sistema de Coordenadas de la Letra

Asimismo, cada *Puerto* define su propio sistema de coordenadas en función del sistema de coordenadas de la letra. El sistema de coordenadas de un puerto está determinado por el vértice inferior izquierdo, sus dimensiones y su orientación. En la siguiente figura se puede observar el sistema de coordenadas de la letra, en color negro, y el sistema de coordenadas de un puerto, en color azul, mientras que en rojo se muestran las coordenadas del vértice inferior izquierdo del puerto y sus dimensiones en función del sistema de coordenadas de la letra.

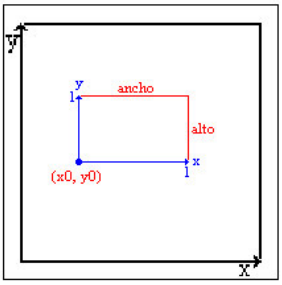


Figura N° 3.7 – Sistema de coordenadas de Puerto

Los ejes de un sistema de coordenadas de puerto están determinados por su sentido, el cual puede ser 0, 1, 2 o 3, correspondiéndose con las figuras N° 3.8 (a), N° 3.8 (b), N° 3.8 (c) y N° 3.8 (d) respectivamente.

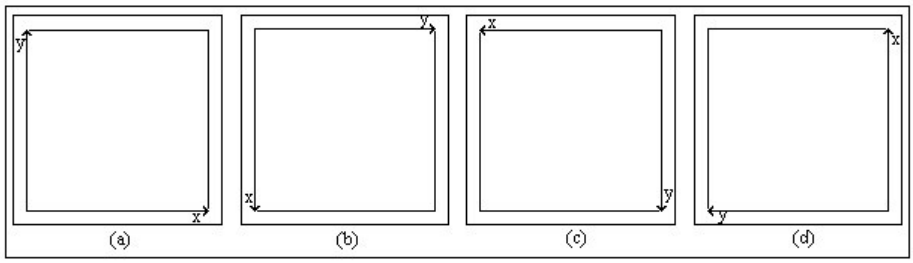


Figura N° 3.8 – Sentidos del Sistema de Coordenadas de Puerto

Si definimos $M_{i \leftarrow j}$ como la matriz de transformación que convierte la representación de un punto en el sistema de coordenadas del puerto j a su representación en el sistema de coordenadas del puerto i y se define P(i) como la representación de un punto en el sistema de coordenadas i, p(k) como la representación de un punto en el sistema de coordenadas k, y p(l) como la representación de un punto en el sistema de coordenadas de la letra. Entonces,

$$P(i) = M_{i \leftarrow l} \cdot P(l) \quad \text{y} \quad P(l) = M_{l \leftarrow k} \cdot P(k).$$

Al sustituir se obtiene

$$P(i) = M_{i \leftarrow l} \cdot P(l) = M_{i \leftarrow l} \cdot M_{l \leftarrow k} \cdot P(k) = M_{i \leftarrow k} \cdot P(k),$$

de manera que

$$M_{i \leftarrow k} = M_{i \leftarrow l} \cdot M_{l \leftarrow k}.$$

Por lo que la transformación de un punto en un sistema de coordenadas de un puerto al de otro será la composición de la transformación del sistema de coordenadas origen al sistema de coordenadas de la letra y la transformación del sistema de coordenadas de la letra al sistema de coordenadas destino.

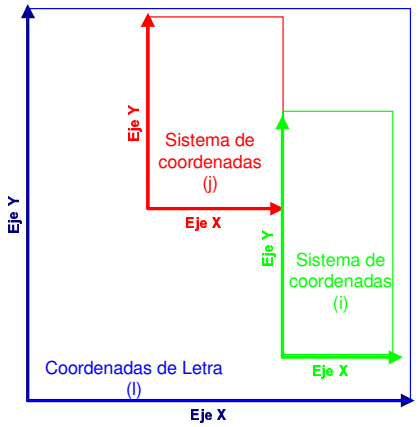


Figura N° 3.9 – Cambio del Sistema de Coordenadas

De esta forma, los puntos pueden ser traducidos entre distintos sistemas de coordenadas de puerto. Esto es importante, sobre todo, en el momento de utilizar las relaciones de entrada-salida entre dos ideogramas, dado que cada uno emplea el sistema de coordenadas de su puerto y para realizar operaciones globales resulta más útil el sistema de coordenadas de la letra.

Como todos los ideogramas toman su entrada sobre el eje x de su puerto, es que algunos puertos deben ser rotados para poder realizar la interconexión de ideogramas. En la siguiente figura se muestran los sistemas de coordenadas de cada puerto para la letra “a”, con la finalidad de que se

puedan apreciar cada una de las rotaciones que se han aplicado. Nótese como las relaciones de entrada-salida están sobre el eje x de cada puerto de entrada.

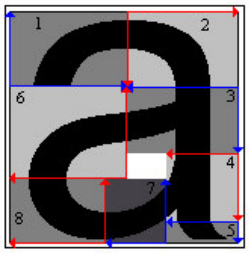


Figura N° 3.10 Sistemas de Coordenadas para la letra “a”

La rotación de puertos puede automatizarse, de forma tal que el diseñador no tenga que rotar explícitamente cada puerto. Esto puede resolverse con sencillez al analizar el lado del puerto de salida sobre el que está establecida la relación. El algoritmo que se utiliza es el siguiente:

```
rotar_Arbol_curva(Arbol_curva ac) : Arbol_curva
Comienzo

        Ideograma i, h;
        Puerto p_entrada, p_salida;
        Salida s;

        i = raíz(ac);

        Para cada salida (s) y subárbol hijo asociado (h) de i:

                Si los puntos de contacto de s están sobre el lado izquierdo:
                        rotar_Puerto(3, get_Puerto(raiz(h)));
                Si los puntos de contacto de s están sobre el lado derecho:
                        rotar_Puerto(1, get_Puerto(raiz(h)));
                Si los puntos de contacto de s están sobre el lado inferior:
                        rotar_Puerto(2, get_Puerto(raiz(h)));
                h = rotar_Arbol_curva (h);

        Fin Para
Fin
```

La función rotar_Puerto(n, p), rota n veces el puerto p en sentido horario, siendo el ángulo de rotación de 90°; mientras que la función raíz retorna el ideograma raíz del árbol_curva y get_puerto devuelve el puerto asociado al ideograma.

3.3 Entradas y Salidas

Las entradas y salidas permiten definir las relaciones que existen entre los ideogramas y éstas están determinadas por las coordenadas de los puntos de salida y sus direcciones.

Si bien una entrada y una salida son conceptualmente distintas, aunque están íntimamente relacionadas, definimos un nuevo concepto con el término *Pendiente*, que abstrae los dos conceptos en un único elemento que define dos puntos de contacto sobre un borde del puerto y una dirección para cada uno de ellos. Con una *Pendiente* una relación entre ideogramas puede

ser vista como un cambio en el sistema de coordenadas, pasando del sistema de coordenadas del puerto de salida, al del puerto de entrada.

Las direcciones de salida están determinadas por dos puntos: 1) el punto de salida y 2) otro punto sobre la dirección. Por lo cual para cada pendiente se deben determinar cuatro puntos, que definen los dos puntos de contacto y sus direcciones. En la figura N° 3.11 se pueden ver los puntos de contacto en azul y los otros dos puntos de dirección en rojo.

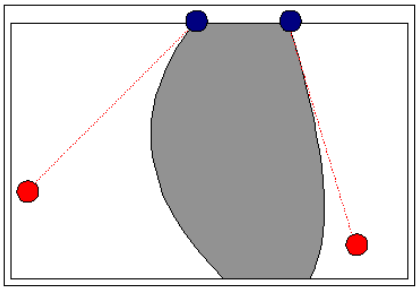


Figura N° 3.11 - Pendiente

Cuando un ideograma utiliza la salida de otro como su entrada, debe aplicarse una función que realice la transformación del sistema de coordenadas del ideograma de salida al sistema de coordenadas del ideograma de entrada y que realice las operaciones necesarias para obtener dos puntos de dirección que pertenezcan al nuevo puerto. Esta función es la siguiente:

Pendiente obtener_salida(Pendiente entrada, Puerto puerto_salida, Puerto puerto_entrada).

Dicha función realiza los siguientes pasos:

1. Transforma las coordenadas de salida al sistema de coordenadas de letra.
2. Obtiene las intersecciones de las rectas de dirección con el puerto de entrada, tomándolas como los nuevos puntos de dirección.
3. Transforma las nuevas coordenadas al sistema de coordenadas del puerto de entrada.

De esta forma se obtienen las coordenadas de entrada con los puntos de contacto y los dos nuevos puntos de dirección. En la figura N° 3.12 se muestra como se obtienen estos nuevos puntos.

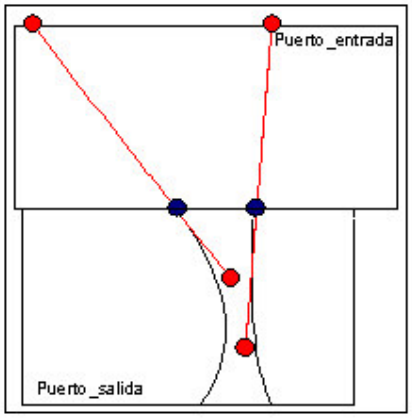


Figura N° 3.12 – Relación entrada-salida

Un punto de dirección para una salida, simplemente debe estar sobre la recta de dirección, y puede definirse con facilidad si la figura que se está representando está determinada por curvas de Bézier, ya que puede utilizarse como punto de dirección el primer punto de control de ésta.

A diferencia de las salidas, para una entrada los puntos de dirección deben estar sobre uno de los bordes del puerto, maximizando de esta manera la distancia entre estos y sus puntos de contacto. Al definir de esta forma los puntos de dirección, se permite luego la utilización de un parámetro de rectitud con el cual obtener fácilmente el primer punto de control para la curva de Bézier del nuevo ideograma. El valor del parámetro de rectitud define "que tanto" se pega el inicio del ideograma a las direcciones de entrada y esto puede implementarse al utilizar como primer punto de control un punto que se encuentre en el segmento de recta determinado por el punto de contacto y su punto de dirección. Más adelante se explica la forma en que se utilizan los parámetros de rectitud junto con las entradas para definir el comienzo de un ideograma.

El ideograma que toma la entrada recibe, de esta forma, los puntos de contacto y las direcciones iniciales a partir de las cuales debe comenzar su figura. Por supuesto que cada ideograma decide si se adapta o no a esta salida y de qué forma. Por ejemplo, podría decidir permitir o no la continuidad G^0 o G^1.

Siguiendo con el ejemplo de la letra "a", en la figura N° 3.13 pueden apreciarse cada una de sus salidas, en color azul, y las entradas tomadas a partir de estas, en rojo. Además, para cada relación de entrada_salida se muestra el nodo del árbol_curva que la define.

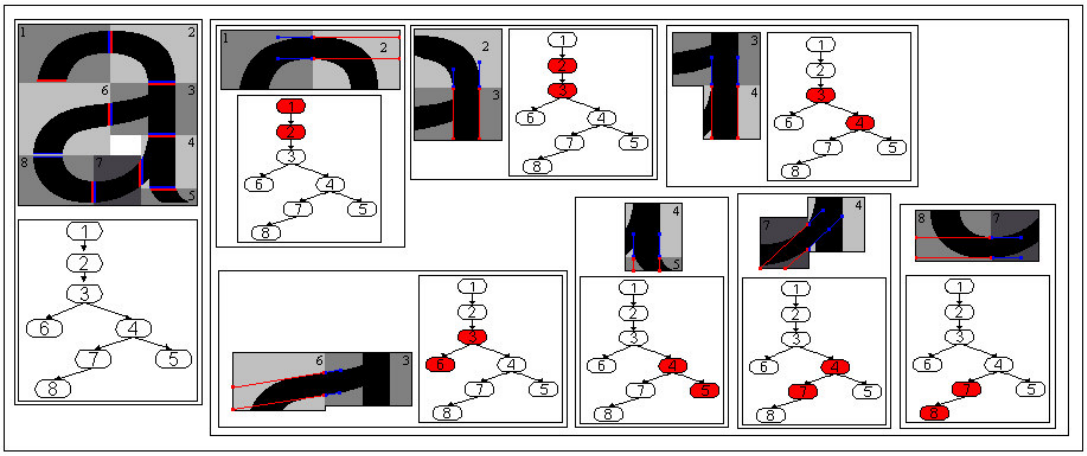


Figura N° 3.13 – Relaciones de entrada-salida para la letra a

3.4 Posiciones y anchos de salida

En general los puntos de contacto de la salidas de un ideograma están definidos por la posición que tendrá el centro de la salida y su ancho final. De esta forma, busca simplificársele al diseñador el control de las salidas, sobre todo porque luego otro ideograma puede tomar esta salida como su entrada y es importante que coincida con el segmento de recta común entre ambos puertos. En la figura N° 3.14 se muestra un ejemplo de cómo se utilizan estos parámetros.

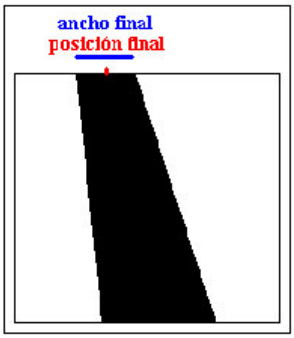


Figura N° 3.14 – Posiciones y anchos finales

La posición final es la media de las coordenadas en el eje x de los puntos de contacto de la salida, mientras que el ancho final es la diferencia entre las mismas, pero no en coordenadas de ideograma si no que en coordenadas de letra. Esto significa que los anchos no son establecidos en función del tamaño del puerto del ideograma en cuestión, si no que en función del sistema de coordenadas de la letra. Por lo tanto, un ancho final de salida de un ideograma es un número que determina la razón entre el ancho final de la salida y el tamaño del lado del puerto de la letra paralelo a la salida.
Con este criterio, se permite una fácil elección de la posición final sobre el puerto del ideograma y un fácil control de los anchos de cada curva, al ser definidos en función del puerto de letra. Es así que la elección de los anchos de cada curva se vuelve una tarea más sencilla a nivel global, ya que todos los anchos son determinados en función del mismo sistema de coordenadas de letra.

En la figura N° 3.15 pueden apreciarse las posiciones finales, en color rojo, y los anchos finales, en color azul, para cada uno de los ideogramas de la letra “a”. Puede verse como las salidas de los ideogramas ubicados en los puertos 7 y 8 pueden hacerse coincidir, utilizando el mismo ancho final y calculando las posiciones finales para que sus puntos medios queden ubicados en el mismo lugar. Más adelante, en la sección inclinaciones, podrá verse como es que se pueden hacer coincidir las direcciones de ambas salidas.

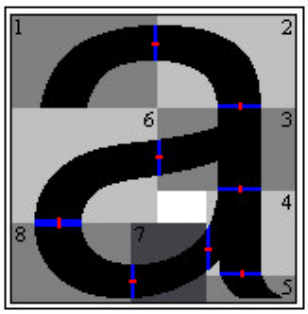


Figura N° 3.15 – Posiciones y anchos finales para la letra “a”

Con el parámetro que controla la posición final de una salida, se puede parametrizar fácilmente la inclinación global de la letra, al hacerse variar las posiciones finales de acuerdo a la inclinación deseada. Más adelante, en la sección Inclinaciones, se explica como puede lograrse la parametrización global de la letra.

3.5 Rectitudes

Cada ideograma decide cómo adaptarse a su entrada. Sin embargo, el método general que se utiliza para permitir la continuidad G^1 respecto a la entrada es seleccionar dos puntos dentro del puerto que pertenezcan a cada una de las rectas de dirección.

La distancia entre un punto de contacto y el nuevo punto de dirección debe ser directamente proporcional a la magnitud de la primer derivada de la curva en el punto de contacto. En la figura N° 3.16 puede apreciarse como la utilización de curvas de Bézier permite solucionar este problema de una manera natural, ya que ésta aproxima los puntos de control en la dirección establecida por estos y los puntos de contacto. En la figura N° 3.16 (b) se disminuye la distancia del punto P1 al punto P0 respecto a la que se presenta en la figura N° 3.16 (a) y puede apreciarse como es que la curva se aproxima más a la recta P_0P_1 en la primer figura, tomando una forma inicial más recta; por este motivo es que se escogió el nombre de "*rectitud*" para representar esta propiedad.

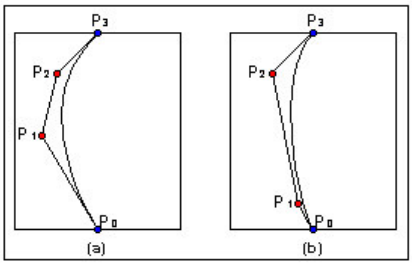


Figura N° 3.16 – Rectitud como un punto de control de una curva de Bézier

El método que se utiliza para parametrizar esta propiedad es mediante un parámetro de rectitud, que varía en el rango [0, 1]. Para esto se define la recta que va del punto de contacto al punto de dirección de la siguiente manera:

Sea P_0 el punto de contacto y P_1 el punto de dirección, entonces la recta definida por estos dos puntos se parametriza como: $(x(t), y(t)) = (x_0 + t(x_1 - x_0) , y_0 + t(y_1 - y_0))$. Al variar t, se obtienen todos los puntos de la recta, pero al restringirlo a [0, 1] se obtienen los puntos que pertenecen al segmento de recta determinado por $[P_0, P_1]$. Si el parámetro t es elegido por el diseñador, entonces la curva estará tan "pegada" a la dirección como t esté cercano a 1.

En la figura N° 3.17 se muestra cómo puede ser utilizado el concepto de rectitud inicial para la letra "a". En este caso, el ideograma que está ubicado en el puerto superior derecho de la letra, es determinado por dos curvas de Bézier que definen los bordes izquierdo y derecho de la curva, mostrándose en rojo los puntos que componen la curva superior y en azul la dirección inicial que debe tener esta como producto de la relación que tiene este ideograma con el vecino que se encuentra a su izquierda. Puede notarse como se va incrementando la rectitud inicial en la medida que el segundo punto de la curva de Bézier se va alejando del primero sobre la dirección de entrada.

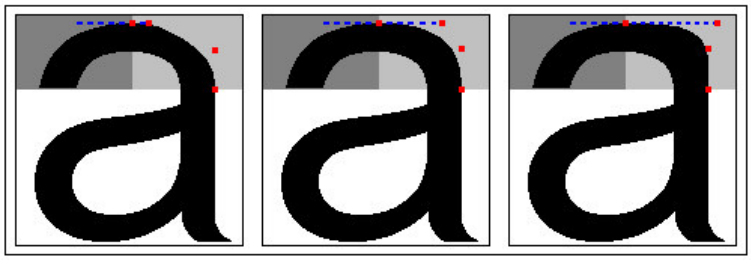


Figura N° 3.17 – Ejemplo de rectitud inicial para la letra "a"

Ya se vio como puede ser parametrizada la rectitud en un rango de valores [0, 1] de forma tal que su utilización resulte sencilla. Las rectitudes están íntimamente relacionadas con las direcciones, puesto que éstas son utilizadas para implementar el concepto de continuidad que es utilizado para definir las relaciones entre ideogramas. A continuación se explica como es utilizado este parámetro junto con uno nuevo, inclinación, para determinar los comportamientos iniciales y finales de un ideograma.

3.6 Inclinaciones

Las direcciones de salida están determinadas por los puntos de contacto y dos puntos que pertenecen a cada una de las rectas que definen cada dirección de salida. Por lo tanto, dejando los puntos de contacto fijos, podemos modificar los otros dos para determinar las direcciones de salida.

Para definir la inclinación se utiliza el parámetro *inclinación*, el cual varía en el rango [-1, 1], siendo la dirección perpendicular al lado de salida si el valor es 0. Para los casos en que las salidas están ubicadas sobre los lados verticales del puerto, al incrementarse el valor de inclinación, la pendiente de salida se va pronunciando más hacia arriba, hasta llegar a la vertical cuando su valor es 1; mientras que sucede lo opuesto cuando el valor del parámetro *inclinación* toma valores negativos. En el caso en que la salida esté ubicada sobre un lado horizontal, los valores negativos determinan una inclinación de salida en el sentido creciente de las x_s, mientras que un valor negativo determina una inclinación en el sentido contrario. Para el caso de las entradas sucede lo mismo que para las salidas sobre un lado horizontal, dado que todas las entradas son definidas sobre el eje x del puerto. En figura N° 3.18 se pueden apreciar, gráficamente, las inclinaciones posibles de acuerdo a los lados del puerto sobre las que son definidas las salidas.

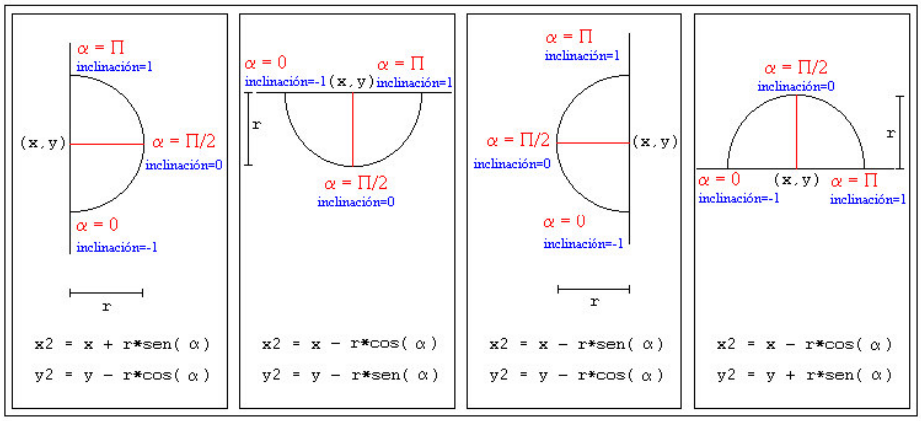


Figura N° 3.18 - Inclinaciones

Para obtener los puntos de dirección se realizan los siguientes pasos:

1- A partir de la inclinación recibida, se calcula el ángulo (α) que forma la dirección respecto al lado de salida.

2- Luego, se obtiene el radio (r) a partir de que tanto se pega la figura a la pendiente final (rectitud final), tomando en cuenta también las dimensiones del puerto.

3- Finalmente se obtienen las coordenadas del punto de dirección a través de la ecuación paramétrica de una circunferencia centrada en (x,y) y de radio r.

El radio (r) podría ser sustituido directamente por el valor del parámetro de rectitud, puesto que éste varía en el rango [0, 1] al igual que las coordenadas del los ejes x e y; sin embargo, esto puede provocar que el punto hallado quede fuera del puerto en el que debe representarse el ideograma, forzando a la curva a sobrepasar los límites del puerto. Por este motivo es que la rectitud es ponderada de forma tal que no sucedan casos indeseados, para luego ser utilizada junto con la inclinación en el cálculo del punto de control.

De esta forma, se puede ver que utilizando parámetros de rectitud, en el rango [0, 1], y de inclinación, de dominio [-1, 1], es posible describir los comportamientos de entrada y salida de cada ideograma con bastante facilidad por parte del diseñador, sin que éste deba poseer conocimientos matemáticos significativos.

Continuando con el ejemplo para la letra “a”, en la figura N° 3.19 se puede apreciar como es determinada la inclinación inicial para esta letra, obteniendo el segundo punto de Bézier de la curva a través de la *rectitud* que define el radio del círculo y la inclinación que determina un punto sobre éste.

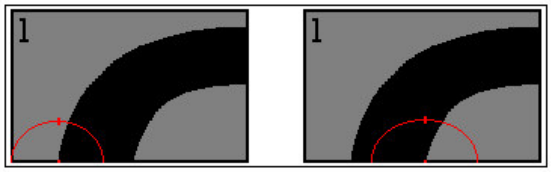


Figura N° 3.19 – Inclinación inicial para la letra “a”

Para las entradas, la información sobre la inclinación inicial está determinada implícitamente a través de los puntos de contacto y dirección; mientras que para las salidas esta información o es determinada por el diseñador a través del parámetro de inclinación, o es determinada directamente por cada ideograma, sin que se permita su manejo directo; mientras que las rectitudes no son determinadas por las salidas o entradas, si no que es el propio diseñador el que debe definirlas de manera explícita mediante la utilización de parámetros.

3.7 Detalles

En el diseño de fonts, los detalles juegan un rol muy importante, puesto que muchas veces la diferencia entre un tipo de letra y otro está basada principalmente en el tipo de detalle. Es por ello que los detalles son definidos por separado y utilizados por cada ideograma como si se tratase de una biblioteca.

El presente diseño de los detalles busca que estos puedan ser utilizados por ideogramas en los que se pretende adaptar los detalles a figuras rectas. Inicialmente vemos que existen dos tipos claros de detalles: 1) aquellos que se pegan a figuras rectas y que tienen una base horizontal y 2) aquellos que se adaptan a rectas horizontales, definiendo un detalle vertical, con algún tipo de inclinación. En la figura N° 3.20 se puede apreciar un ejemplo de cada uno de estos detalles, donde el primer detalle de la figura se corresponde con el detalle horizontal y el segundo con el vertical.

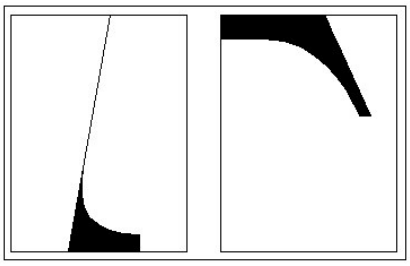


Figura N° 3.20 – Detalle horizontal y detalle vertical

Detalle horizontal

Comencemos analizando el primer tipo de detalle, aquí vemos que éste debe adaptarse a un segmento de recta cuyos extremos están ubicados en los bordes horizontales del puerto. De esta

manera determinamos el segmento de recta con las coordenadas x´s de cada uno de sus vértices. Podemos ver que existen 4 tipos de detalles posibles, como se muestra en la figura N° 3.21.

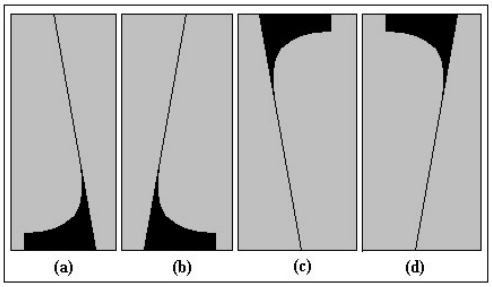


Figura N° 3.21 – Tipos de detalle horizontal

Explicaremos cada una de sus características para el segundo tipo, pues el diseño de los otros tres es análogo.

Hasta ahora sólo hemos visto detalles que forman una curva, sin embargo esta no es la única posibilidad que puede plantearse, por lo tanto, veamos primero cuales son las opciones a tomar en consideració. Podemos comenzar este análisis observando los cuatro tipos de letra más representativos en cuanto al detalle; estos son: Roman, Arial, Courier y Gothic. En la figura N° 3.22 vemos un ejemplo de cada uno de ellos.

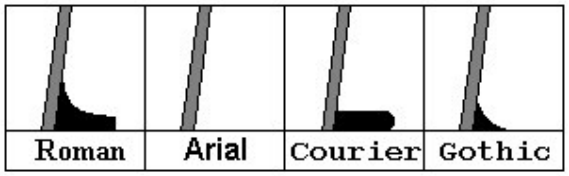


Figura N° 3.22 – Tipos de detalle horizontal para la fuente romana

El detalle Roman, comienza a cierta altura de la recta y termina a una altura menor, describiendo una figura curva; Por su parte Arial no define detalles; Courier utiliza una segmento recto y horizontal que termina en una punta curva; y finalmente Gothic utiliza un diseño similar al romano, pero más pequeño.

De esta forma es que se ha llegado al diseño de detalle que se muestra en la figura N° 3.23.

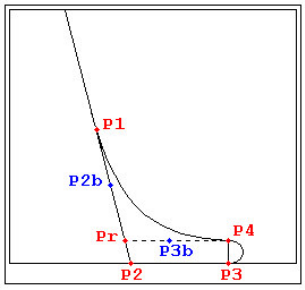


Figura N° 3.23 – Construcción del detalle horizontal

La función que construye cada uno de estos detalles es la siguiente:

```
detalle_Horizontal(int tipo, float detalle_curvo, float posicion_inicial,
  float posicion_final, float grosor, int final_curvo, float ancho_detalle,
  float alto_detalle, float ancho_final, Puerto puerto)
```

Los parámetros *posición_inicial* y *posición_final*, determinan el segmento de recta sobre el cual se apoyará el detalle y el valor de *tipo* define sobre que sector será dibujado el detalle; en este caso se asume el valor 2; de forma tal que éste estará ubicado en la parte inferior y a la derecha del segmento. Por lo tanto, el punto P_2 tendrá las coordenadas (posición_inicial, 0), luego con el parámetro ancho_detalle se determina el punto P3, pero debido a que este ancho está dado en función del sistema de coordenadas del puerto, primero debemos dividirlo por el ancho del puerto que se pasa por parámetro, por lo que las coordenadas del punto P_3 son: (x_2 + ancho_detalle/ancho_puerto). Luego, el parámetro ancho_final define la altura a la que terminará la parte curva del detalle, definiendo así las coordenadas del punto P_4 como (x_3, ancho_final/alto_puerto). Finalmente obtenemos el punto P1 interceptando el segmento de recta con la recta y=alto_detalle/alto_puerto. El punto P_r se utiliza para luego definir la parte curva del detalle y se obtiene interceptando la recta y = y_4 con el segmento de recta.

Inicialmente se "pinta" el paralelogramo definido por [P_2 P_3 P_4 P_r]. Luego, si el parámetro *detalle curvo* tiene el valor 1, se define el área comprendida entre el segmento de recta [P_1, P_4] y la curva de Bézier con puntos de contacto P_1 y P_4, y puntos de control determinandos de la siguiente manera:

$$P_{2b} = P_r + grosor*(P_1 - P_r)$$
$$P_{3b} = P_r + grosor*(P_4 - P_r)$$

De esta forma, cuanto menor sea el valor del parámetro *grosor*, más se aproximarán los puntos P_{2b} y P_{3b} al punto P_3, llegando a un grosor mínimo cuando el valor del parámetro sea 0. El caso opuesto se da cuando su valor es 1, dado que de esta forma los puntos de control coinciden con los puntos de contacto, definiéndose el segmento de recta que va de P_1 a P_4.

Por último, si el valor del parámetro *final_curvo* es uno, se "dibuja" el área comprendida entre el segmento de recta P_3P_4 y la curva de Bézier definida por los puntos [P_3, P_3 + (ancho_final, 0), P_4 + (ancho_final, 0), P_4]. Previamente se le resta al parámetro ancho_detalle el valor de ancho_final, para mantener el ancho del detalle.

Detalle vertical

En este caso, los detalles deben adaptarse a rectas horizontales y extenderse con una determinada inclinación. Esta recta viene dada por dos parámetros que definen su ubicación: *distancia*, que determina la distancia de la recta al borde del puerto opuesto al detalle y *ancho_inicial*, que define el ancho de la recta a la que tiene que adaptarse. En la figura N° 3.24 puede observarse un ejemplo del uso de estos parámetros.

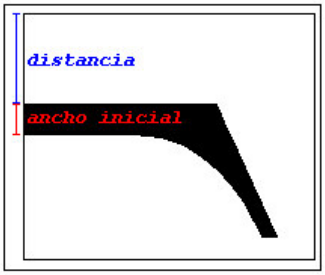


Figura N° 3.24 – Detalle vertical

Al igual que para el detalle horizontal, aquí también se pueden tener cuatro tipos de detalle posibles, como se muestra en la figura 3.25.

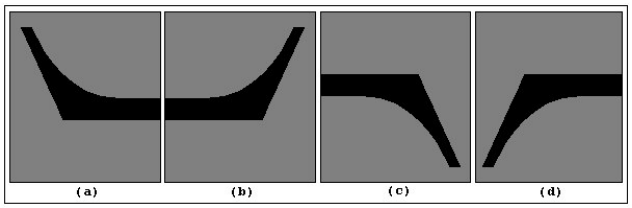


Figura N° 3.25 – Tipos de detalle vertical

Al igual que los detalles horizontales, los detalles verticales pueden tener más posibilidades de las que estamos observando. Por esto es que volvemos a analizar los detalles, esta vez verticales, para los tipos de letra: Roman, Arial, Courier y Gothic. En la figura N° 3.26 vemos un ejemplo de cada uno de ellos.

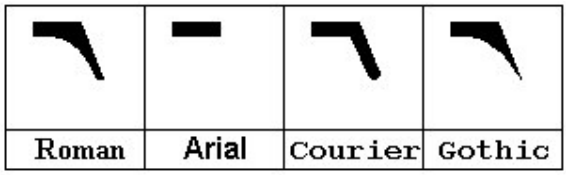


Figura N° 3.26 – Tipos de detalle vertical para la fuente romana

El detalle Roman, comienza a la altura de la recta y termina a una altura menor, describiendo una figura curva con una determinada inclinación; Por su parte Arial no define detalles; Courier utiliza una segmento recto que termina en una punta curva; y finalmente Gothic utiliza un diseño similar al romano, pero con una punta diminuta.

Luego de estas observaciones es que se llega al diseño para el detalle vertical, que se muestra en la figura N° 3.27. En este caso, sólo se presenta el diseño de tipo 3, ya que el método utilizado para los otros tipos es análogo.

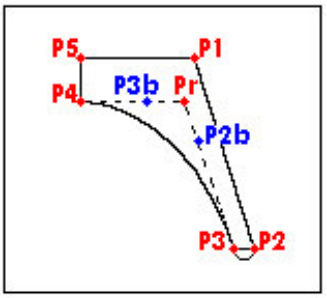


Figura N° 3.27 – Construcción del detalle vertical

La función que construye cada uno de estos detalles es la siguiente:

```
detalle_Vertical(int tipo, float distancia, float posicion_inicial, float
      grosor, int detalle_curvo, int final_curvo, float ancho_detalle,
      float alto_detalle, float ancho_inicial, float ancho_final,
      float inclinacion, Puerto puerto)
```

El parámetro tipo determina uno de los cuatro tipos vistos en la figura N° 3.25.

Los parámetros distancia y *ancho_inicial* tienen el significado que ya se ha indicado, mientras que el parámetro *posición_inicial* determina a partir de qué coordenada x es que se inicia el detalle; por lo tanto el punto P_5 tiene las coordenadas (posición_inicial, 1-distancia) y el punto P_4 las

coordenandas (x_5, y_5-ancho_inicial/alto_puerto) . El parámetro *ancho_detalle* determina el ancho que tendrá el detalle, por lo que podemos definir el punto P_1 como (x_5 + ancho_detalle/ancho_puerto).

Luego, se utilizan los parámetros *alto_detalle* e *inclinación* para definir el punto P_2 de forma similar a la que se explicó para determinar las inclinaciones finales de los ideogramas. Es así que se siguen los siguientes pasos:

1) A partir de la inclinación recibida, se calcula el ángulo (α) que formará el detalle con la horizontal, de la forma que se muestra en la figura N° 3.28.

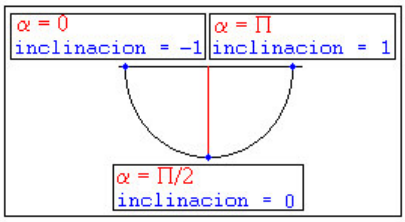


Figura N° 3.28 – Inclinación del detalle

2) Luego, se utiliza como radio (r) el resultado de la división entre alto_detalle y alto_puerto.

3) Finalmente se obtienen las coordenadas del punto P2 a través de la ecuación paramétrica de una circunferencia centrada en P1 y de radio r de la siguiente forma:

$$P_2 = (x_1 - r.\cos(\alpha), y_1 - r.\mathrm{sen}(\alpha))$$

Para calcular el punto P_3, se utiliza el *ancho_final*, obteniéndose las coordenadas (x_2-ancho_final/ancho_puerto, y_2). Y las coordenadas del punto P_r se obtienen como la intersección de la recta $y = y_4$, con la recta que pasa por P_3 y es paralela a P_1P_2.

Inicialmente se "dibuja" el cuadrilátero [P_1, P_r, P_4, P_5]. Luego, si alto_detalle es mayor que ancho_inicial, se dibuja el cuadrilátero formado por [P_1, P_2, P_3, $_{Pr}$]. Si la variable detalle_curvo tiene el valor 1, entonces se dibuja la el área comprendida entre el segmento de recta [P_r, P_3] y la curva de Bézier formada por los puntos [P_4, P_{2b}, P_{3b}, P_3], donde los puntos P_{2b} y P_{3b} son escogidos de la siguiente forma:

$$P_{2b} = Pr + grosor\ .\ (P_4 - P_r),$$
$$P_{3b} = Pr + grosor\ .\ (P_3 - P_r).$$

Finalmente, si el parámetro *final_curvo* tiene el valor 1, se dibuja el área comprendida entre el segmento de recta P_3P_2 y la curva de Bezier definida por los puntos [P_3, P_3 - (0, ancho_final), P_2 - (0, ancho_final), P_2]. Previamente se le resta al parámetro *alto_detalle* el valor de *ancho_final*, para mantener el largo del detalle

3.8 Ideogramas

Cada ideograma tiene su entrada sobre el eje x de su puerto, por lo cual para generar algunas figuras deben rotarse los puertos de forma tal que la entrada quede en el lugar deseado.

Un ideograma define su forma utilizando primitivas gráficas. Las primitivas gráficas son las encargadas de lidiar con los problemas de la computación gráfica, haciendo especial hincapié en

el manejo de la resolución, el cual es independiente de las definiciones de ideogramas, permitiendo que el tipo de resolución sea definido en el momento de procesar los árboles_curva.

Para cada ideograma deben proveerse las siguientes funciones: un constructor con el cual poder definir el ideograma con todos sus parámetros y una función que devuelva un conjunto de "Figuras" a partir del ideograma; donde cada figura representa una primitiva gráfica sobre el puerto. Entre los parámetros que obligatoriamente debe contener un constructor tenemos:

- Identificador de ideograma (Entero): éste permite asignarle un identificador al ideograma creado, de forma tal que luego pueda ser referenciado por otro para el intercambio de entradas-salidas.
- Puerto de ideograma (Puerto): representa el sector del área destinada a la letra, en la que el ideograma será representado.
- Identificador del ideograma padre (Entero): representa la relación de entrada-salida. Este identificador determina el ideograma al que se hace referencia.
- Número de salida (Entero): Determina qué salida del ideograma "padre" es la que se utilizará.

En caso de que se trate del ideograma raíz, la entrada debe ser creada explícitamente y establecida por medio de una función que se provee para tales efectos.

Ideogramas seleccionados

Para facilitarle el trabajo al diseñador de fonts, se debe definir un conjunto de ideogramas suficientemente representativo y donde cada ideograma tenga un conjunto de parámetros que permitan controlar su comportamiento.

De esta forma es que se distinguen, a priori, tres tipos de ideogramas:

- Los que definen figuras rectas.
- Los que definen figuras curvas.
- Los que definen figuras con componentes curvos y rectos.

1) Los ideogramas que definen figuras rectas deben ser utilizados en general para definir gran parte de las letras mayúsculas, aunque también se pueden utilizar con los demás caracteres. Estas figuras, además, pueden representar detalles sobre cualquiera de sus vértices. Si bien los detalles pueden ser curvos, la esencia de estos ideogramas es la de representar figuras rectas. Sin embargo, debido a que en general los detalles se "dibujan" sobre figuras rectas es que en algunos ideogramas se ha incluido la posibilidad de definirlos. Es así que se definieron los siguientes ideogramas, entre los cuales el primero y último pertenecen al grupo de los que ofrecen continuidad G^0, mientras que los otros dos pertenecen al de los que ofrecen continuidad G^1:

- semiplanos(`int id_ideograma, int tipo, float posicion_final, float ancho_final, int detalle_curvo, int final_curvo, float ancho_detalle, float alto_detalle, float ancho_final_detalle, float grosor, int ld, int rd, int ru, int lu, Puerto puerto, int id_entrada, int numero_entrada`);

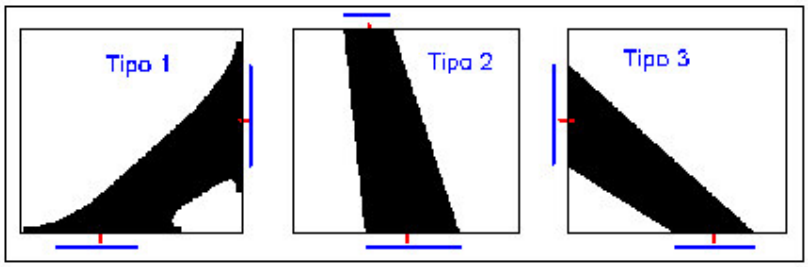


Figura N° 3.29 – Ideograma Semiplanos

Este ideograma representa una figura con curvas rectas (Figura N° 3.29) y recibe el nombre de semiplanos ya que puede interpretarse como la intersección de dos semiplanos. Todos los ideogramas definen su entrada sobre el eje x de su puerto, y de acuerdo al valor seleccionado para el parámetro tipo, se define la salida sobre los lados derecho, superior o izquierdo para los valores de tipo 1, 2 y 3 respectivamente.

Los parámetros id_ideograma, puerto, id_entrada y numero_entrada: cumplen las funciones que ya se han indicado. Mientras que los parámetros posición_final y ancho_final controlan su salida.

Por otra parte, están los parámetros que controlan los detalles sobre sus vértices. Los detalles se verán más adelante en la siguiente sección. Los parámetros ld, rd, ru y lu especifican si se deben "dibujar" o no los detalles sobre sus vértices inferior izquierdo, inferior derecho, superior derecho y superior izquierdo respectivamente.

- ```
  continuacion_Semiplanos(int id_ideograma, int tipo,
     int detalle_curvo, int final_curvo,
     float ancho_detalle, float alto_detalle, float ancho_final_detalle,
     float grosor, int ld, int rd, int ru, int lu,
     Puerto puerto, int id_entrada, int numero_entrada)
  ```

  Es igual al ideograma anterior, con la diferencia de que no se especifica la posición final, ni el ancho final, si no que las curvas simplemente continúan la pendiente de entrada.

- ```
  codo(int id_ideograma, int tipo,
        float posicion_final, float ancho_final,
        Puerto puerto, int id_entrada, int numero_entrada)
  ```

  Este ideograma define un codo recto sobre el puerto. Los parámetros id_ideograma, puerto, id_entrada y numero_entrada: cumplen las funciones que ya se han indicado. Mientras que los parámetros posición_final y ancho_final controlan su salida. El tipo define el lado de salida; izquierdo o derecho para los valores de tipo 1 y 2 respectivamente.

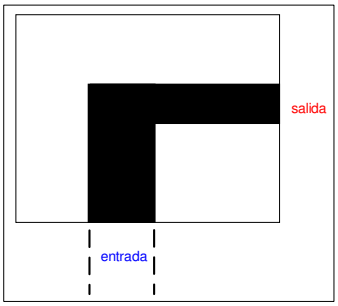


Figura N° 3.30 – Codo recto

❖
```
Vertice(int id_ideograma, float alto_inicial,
       float ancho_inicial_izq,  float ancho_inicial_der,
       float posicion_final_izq, float ancho_final_izq,
       float posicion_final_der, float ancho_final_der,
       float ancho_detalle, float alto_detalle, float ancho_final_detalle,
       int detalle_curvo, int final_curvo, float grosor, int l, int r,
       Puerto puerto, int id_entrada, int numero_entrada)
```

Representa un vértice que se apoya sobre el borde inferior con sus dos lados definiendo sus salidas sobre el borde superior del puerto, como se muestra en la figura N° 3.31.

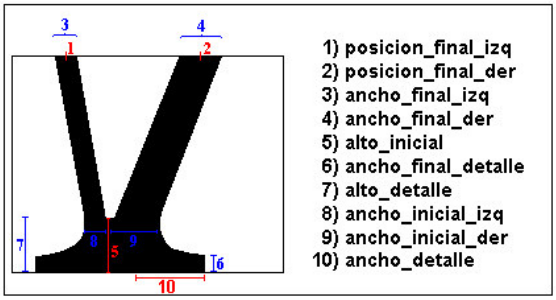


Figura N° 3.31 – Ideograma Vértice

Los parámetros id_ideograma, puerto, id_entrada y numero_entrada: cumplen las funciones mencionadas con aterioridad. Los parámetros posición_final_izq, posición_final_der, ancho_final_izq y ancho_final_der determinan las posiciones y anchos de cada una de las dos salidas, mientras que ancho_inicial_izq y ancho_inicial_der determinan el anco inicial de las ”rectas” que forman el vértice. Alto inicial se utiliza para que no ocurran casos en los que las dos “pseudorectas” sean tan angostas en su anchos iniciales, que el vértice quede separado en dos y esto hace que se pinte toda la zona que está comprendida entre las dos líneas. Finalmente están los parámetros que controlan las dimensiones y características de los detalles: ancho_detalle, alto_detalle, ancho_final_detalle, detalle_curvo y final_curvo; los cuales se dibujan sobre la zona izquierda o derecha si los parámetros l y r respectivamente son distintos de 0.

La posición inicial y el ancho inicial del vértice son ingresados implícitamente en la entrada.

Este ideograma genera dos salidas, la primera es la que corresponde a su lado izquierdo y la segunda a su lado derecho.

- `continuacion_Vertice(int id_ideograma, int tipo,`
  `float alto_final, float ancho_final, float ancho_inicial_der,`
  `float posicion_final_der, float ancho_final_der,`
  `float ancho_detalle, float alto_detalle, float ancho_final_detalle,`
  `int detalle_curvo, int final_curvo, float grosor, int l, int r,`
  `Puerto puerto, int id_entrada, int numero_entrada)`

Este ideograma es idéntico al anterior en cuanto a su forma, pero difiere en cómo es construido, ya que en lugar de iniciarse sobre el vértice, lo hace sobre uno de sus lados. Si el valor del tipo es 1, entonces se inicia como continuación del lado izquierdo, mientras que si es 2 lo hace sobre su lado derecho.

Los parámetros id_ideograma, puerto, id_entrada y numero_entrada cumplen las funciones que ya se han indicado. Por defecto se asume que el tipo es 1 y por lo tanto se utilizan los nombres posición_final_der y ancho_final_der para determinar la ubicación de la salida derecha, mientras que si el valor de tipo es 2 estos serán los que definan la ubicación de la salida derecha. El parámetro ancho_inicial_der cumple la misma función que el caso del ideograma anterior y finalmente los parámetros: ancho_detalle, alto_detalle, ancho_final_detalle, detalle_curvo y final_curvo son los que determinan las formas y magnitudes de los detalles, desplegándose si los parámetros l y r tienen los valores que se mencionaron anteriormente. En la figura N° 3.32 se muestra un ejemplo de definición para este ideograma, correspondiéndose la figura (a) con el vértice que se inicia sobre su lado derecho y la figura (b) con el que se inicia sobre su lado derecho.

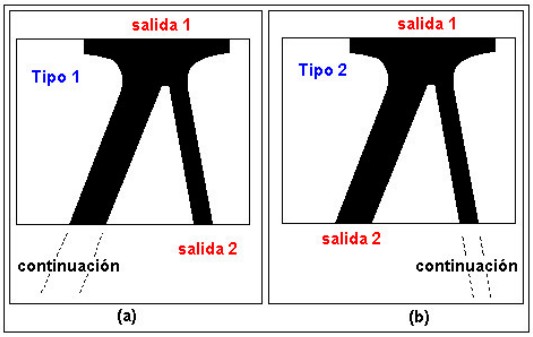


Figura N° 3.32 – Ideograma Continuación_vértice

2) Los ideogramas que definen figuras curvas son muy importantes y tienen que ser determinados con un conjunto de parámetros lo suficientemente flexible como para poder controlar su forma, pero sin que esto implique un manipulación compleja. Por lo tanto, se debe ser preciso a la hora de elegir sus parámetros y es así que además de los parámetros obligatorios, se consideraron los conceptos antes mencionados de: rectitud, inclinación, posición final y ancho final. Estos ideogramas son los siguientes:

- `curva(int id_ideograma, int tipo, float posicion_final, float ancho_final,`
  `float rectitud_izq, float rectitud_der,`
  `float inclinacion_izq, float inclinacion_der,`
  `float rectitud_final_izq, float rectitud_final_der,`
  `Puerto puerto, int id_entrada, int numero_entrada)`

Los parámetros id_ideograma, puerto, id_entrada y numero_entrada cumplen las funciones que ya se han indicado. Mientras que los parámetros posición_final y ancho_final controlan la posición de su salida. Los parámetros inclinación_izq e

inclinación_der controlan la inclinación final de los lados izquierdo y derecho de la curva respectivamente. Finalmente, los parámetros rectitud_final_izq y rectitud_final_der determinan las rectitudes finales de los lados izquierdo y derecho de la curva y los parámetros rectitud_izq y rectitud_der las rectitudes iniciales izquierda y derecha.

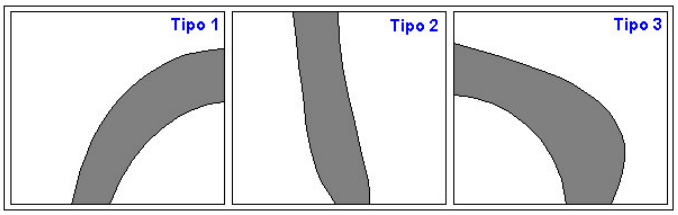


Figura N° 3.33 – Ideograma Curva

En la figura N° 3.33 se muestra un ejemplo para cada tipo, donde si tiene el valor 1 la salida es sobre el lado derecho, para el valor 2 sobre el lado superior y si es 3 sobre el lado izquierdo del puerto. Esta figura está definida por dos curvas de Bézier, una para el borde izquierdo y la otra para el derecho. La manera en que se determinan sus puntos de control, a través de los parámetros de inclinación y rectitud, es la siguiente:

Se divide cada coordenada "y" de los puntos de contacto de la salida entre dos y los resultados son utilizados para dividir el puerto en las zonas en que se ubicarán los primeros puntos de control (zonas inferiores). Luego se interceptan las direcciones de entrada con las rectas horizontales definidas por las coordenadas "y" halladas. Y en función de las rectitudes iniciales es que hallan las coordenadas finales de los primeros puntos de control. Para los segundos puntos de control se utiliza el mismo método, sólo que en el caso de los tipos 1 y 3 se hace respecto a la coordenada x.

En la figura N° 3.34 se puede apreciar la metodología recién explicada, para el tipo 1, dividida en tres pasos: a) determinación de las zonas de control; b) intercepción de las direcciones; y c) determinación de las coordenadas finales de los puntos de control. En este caso, por cuestiones de claridad, sólo se muestra la solución para el lado izquierdo del ideograma.

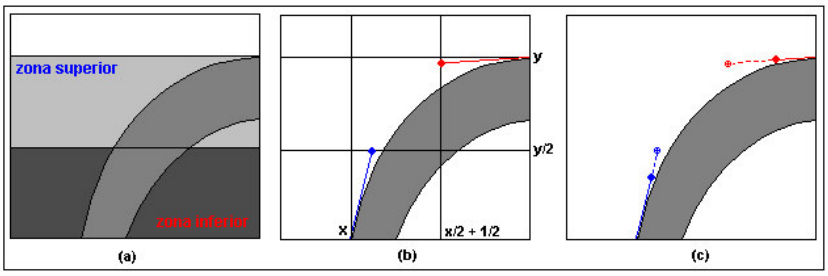


Figura N° 3.34 – Tipo 1 para el ideograma Curva

- `elipses_Concentricas(int id_ideograma, float ring, float elipse, float incl, float incl2, Puerto puerto, int id_entrada, int numero_entrada)`

Este ideograma representa la diferencia entre dos elipses concéntricas: las elipses exterior e interior. En la figura N° 3.35 se observan los parámetros ring y elipse que utiliza la representación de elipses concéntricas.

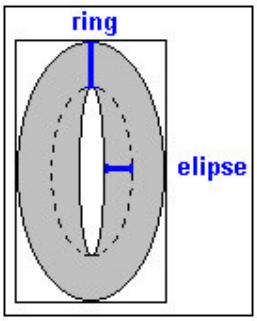


Figura N° 3.35 – Ideograma Elipses_concéntricas

El parámetro ring controla el diámetro, el parámetro elipse la deformación elíptica y los parámetros incl e incl2, las inclinaciones de las elipses menor y mayor respectivamente. Mientras que los parámetros id_ideograma, puerto, id_entrada y numero_entrada, cumplen las funciones que ya se han mencionado. En la figura N° 3.36 se pueden observar las diferencias de inclinación; la primer elipse concéntrica (a) no tiene inclinación, mientras que para la segunda (b) se utiliza una inclinación de valor 1 para la elipse menor, dejándose la elipse mayor sin inclinación; y en el tercer caso (c) se deja sin inclinación la elipse menor y se asigna un valor de uno para la elipse mayor. En caso de utilizarse valores negativos para las inclinaciones, estas serán en el sentido opuesto, inclinando las elipses hacia la izquierda.

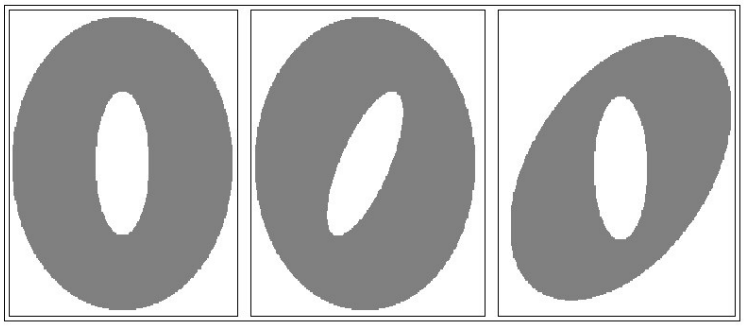


Figura N° 3.36 – Inclinaciones para el ideograma Elipses concéntricas

3) Los ideogramas que definen figuras con componentes curvos y rectos son utilizados en general para proveer más de una salida. Tener más de un componente curvo para un ideograma hace que el manejo de parámetros sea una tarea complicada, sobre todo porque el cálculo de las intersecciones entre ellos es más complicado que en el caso recto y si se "dibujan" dos veces las zonas interceptadas por dos curvas se está generando redundancia que ocupa espacio en el archivo de salida. Sin embargo, determinar las intersecciones de una curva con una recta es sensiblemente más sencillo. Los ideogramas que utilizan componentes curvos y rectos son los siguientes:

-
```
semiplanos_Curva(int id_ideograma, int tipo, int ignorar_curva,
  float posicion_final, float ancho_final,
  int tipo_salida_curva, int lado_salida_curva,
  float posicion_inicial_curva, float ancho_inicial_curva,
  float posicion_final_curva, float ancho_final_curva,
  float incl_inicial_curva, float incr_inicial_curva,
  float incl_final_curva, float incr_final_curva,
  float rectl_inicial_curva, float rectr_inicial_curva,
  float rectl_final_curva, float rectr_final_curva,
  Puerto puerto, int id_entrada, int numero_entrada)
```

Este ideograma define una intersección de semiplanos de la misma forma que el ideograma semiplanos, donde tipo, posición final y ancho_final tienen la misma función. Además los parámetros id_ideograma, puerto, id_entrada y numero_entrada, cumplen el objetivo general que ya se ha indicado. Este ideograma además especifica una curva que parte de uno de los lados de la intersección de semiplanos y define una salida que, de acuerdo al valor del parámetro lado_salida_curva, se apoya sobre uno de los bordes del puerto. En la figura N° 3.37 se puede ver un ejemplo de este ideograma pudiendo apreciarse la diferencia entre las dos curvas que lo conforman (a), y el ideograma final en negro (b).

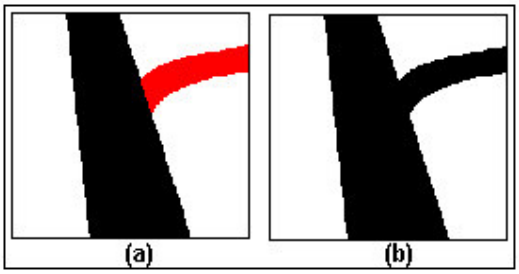


Figura N° 3.37 – Ideograma Semiplanos_Curva

La curva se inicia a la altura indicada por el parámetro posicion_inicial_curva y termina en la indicada por posicion_final_curva; el ancho de esta está determinado en su comienzo por el parámetro ancho_inicial_curva y en su parte final por ancho_final_curva. Las inclinaciones están determinadas inicialmente por los parámetros incl_inicial_curva para el lado izquierdo y por incr_inicial_curva para el derecho. Lo mismo ocurre para las inclinaciones finales con los parámetros incl_final_curva e incr_final_curva y la forma en que cada curva se adapta a las direcciones determinadas está dada por cada parámetro de rectitud. Las rectitudes iniciales están determinadas por los parámetros rectl_inicial_curva y rectr_inicial_curva para los lados izquierdo y derecho respectivamente, y las finales por los parámetros rectl_final_curva y rectr_inicial_curva de la misma forma para los lados izquierdo y derecho de la curva. Sin embargo, si el parámetro ignorar_curva tiene el valor 1, entonces se ignorarán los valores de inclinación y la curva será “dibujada” como si se tratase de una intersección de semiplanos.

La curva puede estar a la izquierda o a la derecha de la figura recta, lo cual está determinado por el valor del parámetro tipo_salida_curva. Si el valor del parámetro es 1, entonces su salida estará sobre el lado derecho del puerto (figura N° 3.38 (a)), en cambio si el valor es 2, su salida estará sobre el lado izquierdo (figura N° 3.38 (b)).

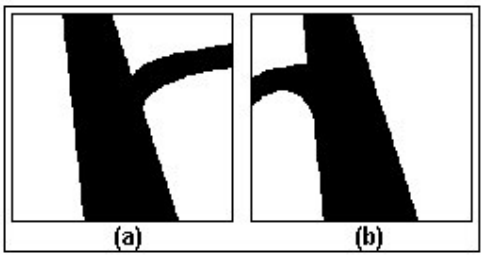


Figura N° 3.38 – Tipos del ideograma Semiplanos_Curva

- `continuacion_Recta_Curva(int id_ideograma, int ignorar_curva, int tipo_salida_curva, float largo_izquierdo, float largo_derecho, float posicion_inicial_curva, float ancho_inicial_curva, float posicion_final_curva, float ancho_final_curva, float incl_inicial_curva, float incr_inicial_curva, float incl_final_curva, float incr_final_curva, float rectl_inicial_curva, float rectr_inicial_curva, float rectl_final_curva, float rectr_final_curva, Puerto puerto, int id_entrada, int numero_entrada)`

  Este ideograma es idéntico al anterior, salvo que en lugar de determinar la posición y el ancho final de la intersección de semiplanos, se continúa la pendiente que se toma como entrada. Además se agregan los parámetros largo_izquierdo y largo derecho que determinan la longitud que tendrá la figura recta; de todas formas la salida será generada como si no se hubiese cortado. En la figura N° 3.39 puede apreciarse como se utilizan estos dos parámetros.

Figura N° 3.39 – Largos iniciales para el ideograma Semiplanos_Curva

Hasta ahora los detalles eran parte de los ideogramas rectos, sin embargo esto no es suficiente para expresar todos los tipos de detalle necesarios en el diseño, dado que es preciso utilizar detalles que por sí mismo sean definidos como ideogramas. A continuación se presentan los tres tipos de ideograma de detalle que se utilizan:

- `detalle_Inicial(int id_ideograma, float posicion_detalle, float largo_detalle, float ancho_detalle, float alto_final_detalle, int final_curvo, float posicion_I_curva, float ancho_I_curva, float posicion_F_curva, float ancho_F_curva, float inclinacion, float inclinacion_I_l, float inclinacion_I_r, float inclinacion_F_l, float inclinacion_F_r, float rectitud_inicial_l, float rectitud_inicial_r, float rectitud_final_l, float rectitud_final_r, Puerto puerto, int id_entrada, int numero_entrada)`

  En algunos caracteres, el detalle es una parte específica de la letra y no un agregado. Por ejemplo, la letra ”S” tiene sus detalles como el comienzo y el final de la curva. Es por esto que se definió éste ideograma que define el detalle como comienzo de una curva. En la figura N° 3.40 se pueden apreciar los detalles de las letras “S” y “C” para los tipos Roman, Arial, Courier y Gothic.

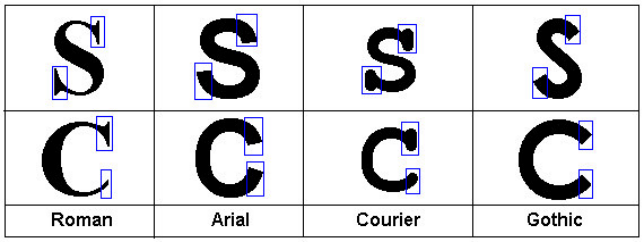


Figura N° 3.40 - Detalles iniciales y finales

Tomando en cuenta los detalles presentados en la figura anterior, es que se tomaron las decisiones de diseño que se pueden apreciar en la figura N°3.41.

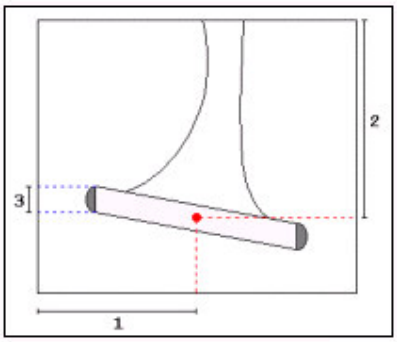


Figura N°3.41 – Detalle Inicial

El detalle está compuesto de un paralelogramo con dos lados verticales y los otro dos inclinados de acuerdo al parámetro *inclinación*, siendo horizontal si su valor es cero y girando en sentido horario para los valores positivos hasta llegar a la vertical; lo mismo sucede para los valores negativos pero en sentido antihorario. Luego se construye la salida que comienza sobre este paralelogramo y termina en el borde superior del puerto.

Esta salida comienza con las inclinaciones determinadas por los parámetros inclinación_I_l e inclinación_I_r y con rectitudes iniciales rectitud_inicial_l y rectitud_inicial_r. Las inclinaciones de salida están determinadas por los parámetros inclinación_final_l e inclinación_final_r; y sus rectitudes finales por los parámetros rectitud_final_l y rectitud_final_r. Finalmente, el ancho inicial de la salida está definido por ancho_I_curva y el ancho final por ancho_F_curva, cuyos centros están determinados por posición_I_curva para el inicio de la salida y posición_F_curva para la parte final de la salida.

Los otros parámetros que controlan el paralelogramo, de acuerdo a la numeración de la figura anterior, son: 1) posición_detalle, el cual determina la ubicación central del paralelogramo de acuerdo al eje x; 2) largo_detalle, que determina el largo que tendrá el detalle, desde el centro del paralelogramo hasta la salida; y 3) alto_final_detalle, el cual determina el largo de los lados verticales del paralelogramo.

Finalmente, si el valor del parámetro final_curvo es 1, entonces se dibujarán detalles curvos a los lados del paralelogramo. Estos detalles curvos se pueden apreciar, de color gris, en la figura anterior.

❖
```
detalle_Final(int id_ideograma,float posicion_detalle, float
  largo_detalle, float ancho_detalle, float alto_final_detalle, float
  posicion_curva, float ancho_curva, float inclinacion, float
  inclinacion_l, float inclinacion_r, float rectitud_inicial_l, float
  rectitud_inicial_r, float rectitud_final_l, float rectitud_final_r,
  Puerto puerto, int id_entrada, int numero_entrada)
```

El comportamiento de este ideograma es igual al del anterior, sólo que en lugar de ser el comienzo de una curva, es el final. Se adapta a una entrada y luego, define el detalle final. En la figura N°3.42 se puede apreciar un ejemplo del mismo.

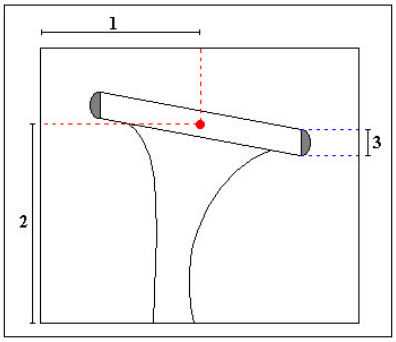


Figura N° 3.42 – Detalle Final

A diferencia del ideograma anterior, no se definen las inclinaciones iniciales del detalle, ya que éste se adapta a la entrada, determinando con los parámetros rectitud_inicial_l y rectitud_inicial_r las rectitudes iniciales del ideograma. El resto de parámetros tienen el mismo significado que para el ideograma anterior, a diferencia del cual, en este caso, no se define una salida.

- `detalle_Paralelo(int id_ideograma, float largo, float inclinacion, float ancho_detalle, float alto_detalle, float ancho_final_detalle, float grosor, int detalle_curvo, int final_curvo, int l, int r, Puerto puerto, int id_entrada, int numero_entrada)`

Este ideograma, a diferencia del ideograma anterior, ofrece una manipulación menos flexible pero más fácil de utilizar para letras con detalles finales sobre curvas rectas. En la figura N° 3.43 se puede apreciar una descripción de su comportamiento.

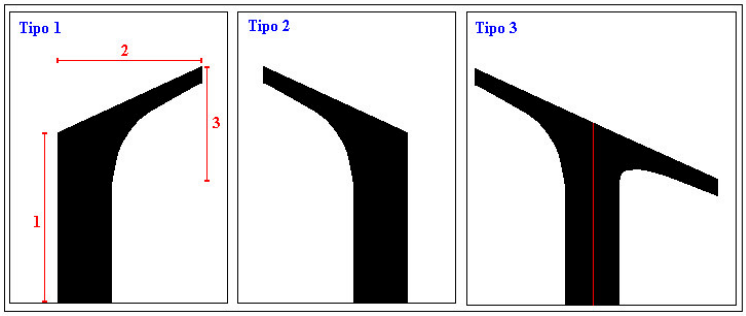


Figura N° 3.43 – Detalle Paralelo

Los parámetros l y r, definen como será el detalle de la siguiente forma: si el valor de l es 0 y el de r es 1, el detalle se dibujará hacia el lado derecho del ideograma; si en cambio el l es 1 y r es 0, lo hará sobre su lado izquierdo; y finalmente si los dos parámetros tienen el valor 1, el detalle se producirá sobre ambos lados del ideograma.

En la figura N°3.43 (a) se puede apreciar el significado de los siguientes parámetros: 1) largo, el cual determina la longitud recta del ideograma; 2) alto_detalle, que determina la longitud que tendrá el detalle final; 3) ancho_detalle, representa el ancho del detalle final; y 4) ancho_final_detalle, que es el ancho de la punta final del detalle.

La inclinación que tendrá el detalle depende directamente del valor del parámetro inclinación. Si su valor es cero, entonces el detalle final será horizontal, en la medida que se incremente su valor, éste irá girando en sentido horario para el detalle izquierdo y en sentido antihorario para el lado derecho, hasta que alcanza su posición vertical cuando el parámetro inclinación obtiene el valor 1; lo mismo sucede para los valores negativos pero en sentido inverso.

El parámetro grosor define cuál será el ancho del detalle final y su mecanismo es el mismo que se utiliza para los detalles comunes. El parámetro detalle_curvo determina si el detalle será curvo y final_curvo determina si se “dibujará” un detalle curvo al final del ideograma.

3.9 Primitivas Gráficas

Los ideogramas definen sus salidas utilizando un conjunto de primitivas gráficas que se ocupan de las engorrosas tareas de la computación gráfica. Éstas primitivas deben solucionar el problema de la resolución y la representación de fronteras entre otras tareas.

Las primitivas gráficas utilizan directamente la librería gráfica de C5, por lo que aquí es donde se manejan los conceptos de port, lista de ports y las funciones asociadas a ellos.

La primitiva principal es la que define el área encerrada entre dos curvas de Bézier, tal como se muestra en la figura N° 3.44.

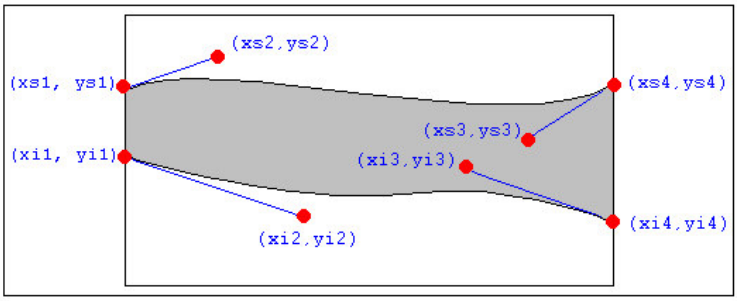


Figura N° 3.44 – Primitiva Bézier

Esta primitiva permite definir con bastante facilidad las dos curvas que determinan la figura, sin embargo, uno sólo puede definir figuras que terminan en el lado opuesto al de comienzo del port. Por lo tanto, no se pueden representar áreas que estén encerradas entre curvas que no sean funciones. Por este motivo es que se definió la función bezier_curvo, que define una curva de Bézier que termina en un lado del puerto, adyacente al de comienzo, como se muestra en la figura N° 3.45.

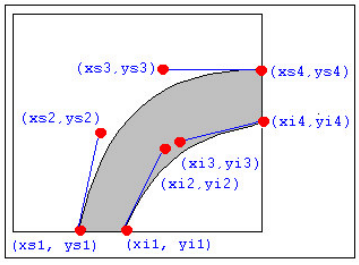


Figura N° 3.45 – Bézier-Curvo

De esta forma, utilizando más de una curva de Bézier, se pueden definir representaciones más complejas que el área encerrada entre dos funciones, como se muestra en la figura N° 3.46.

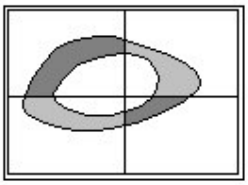


Figura N° 3.46 – Composición de primitivas

Como además se tenía la función opm_ellipses de C5 que representa el área encerrada entre dos elipses concéntricas, se decidió agregarla a las primitivas, ya que brinda un buen nivel de abstracción. En la figura N° 3.47 se puede ver un ejemplo de utilización de esta función, en la cual se "pinta" el área que está encerrada entre las dos elipses.

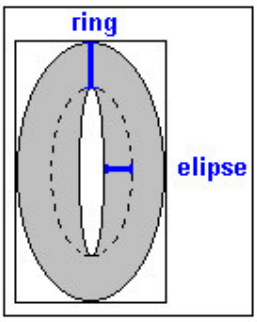


Figura N° 3.47 – Elipses concéntricas

La elipse exterior tiene sus dos radios principales paralelos a los lados del port en sus puntos medios. El parámetro ring define la diferencia entre los radios mayores exterior e interior y el parámetro elipse la deformación elíptica.

Finalmente, se decidió utilizar una primitiva que permitiera definir figuras rectas con facilidad. Es así que se definió la primitiva semiplanos, que se muestra en la figura N° 3.48.

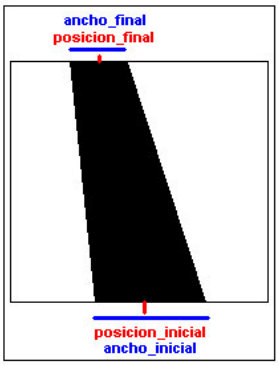


Figura N° 3.48 – Primitiva Semiplanos

El ideograma semiplanos recibe este nombre debido a que está formado por la intersección de dos semiplanos y a que no necesariamente define una recta, puesto que su ancho puede variar. Éste se inicia sobre el lado inferior del port y finaliza su diseño sobre el lado superior y está determinado por los parámetros de posición y ancho, como se muestra en la figura N° 3.48.

Manejo de la resolución

El manejo de la resolución debe ser sencillo para el usuario y las primitivas gráficas son las que deben encargarse de manejar este concepto para que pueda ser utilizado directamente por él, a través de la elección de un tipo de la letra, y de los diseñadores de fonts, a través de los

ideogramas. Por este motivo es que se decidió utilizar un parámetro entero que define recursivamente el tamaño mínimo de los ports que se usan para crear una imagen. La definición del tamaño mínimo de estos ports está dada por el parámetro rec_nr, en función del tamaño del port original.

Hay dos tipos para el manejo de la resolución: manual y automático. La función que se encarga de obtener el tamaño mínimo de los ports recibe un parámetro entero, rec_nr, a través del cual se determina la resolución automática cuando éste toma valores negativos, mientras que la resolución manual está definida para los valores positivos, resultando el port nulo en el caso de que el valor del parámetro sea 0. El valor de este parámetro es modificado recursivamente hasta obtener su valor final.

Si el valor del parámetro es positivo, el tamaño de los ports está determinado de la siguiente forma:

- Si rec_nr es 1, entonces el tamaño será igual al del port original,
- si es mayor que uno, entonces su tamaño será $1/2^{rec\_nr}$ del tamaño original.

Mientras que si el valor del parámetro es negativo, se utiliza la resolución automática, que determina el tamaño mínimo de los ports, a través del resultado de la función, de la misma forma que para los valores mayores a 1. La convención que se sigue para la obtención de un valor a través del parámetro rec_nr es la siguiente:

- Si recnr = -1 se utiliza el valor de resolución estándar para la puerta en cuestión.
- Si recnr < -1, se calcula el valor para (recnr-1) y se lo incrementa en 1.

Normalmente los valores de resolución automática utilizados son -1,-2 y -3. La función que engloba estos criterios para la determinación de la resolución es la siguiente:

```
if(rec_nr<0) rec_nr=set_recnr(port_size(opm_hd(pl)))-1-rec_nr
```

Finalmente, para poder determinar la resolución se debe establecer el tamaño mínimo de los ports, en función del valor retornado por la función, para lo cual se debe realizar la siguiente operación $1/2^{rec\_nr}$ (port_size(port)).

Como puede apreciarse en la función anterior, el cálculo de los tamaños mínimos de los ports que se deben utilizar para definir una figura, está basado en el tamaño del primer port de la lista que es pasado por parámetro, por lo que en el caso de que los restantes ports de la lista tuvieran un tamaño diferente, las representaciones sobre cada una de ellos tendrán distinto grado de resolución. Si bien podría resolverse este problema de otra forma, debido a que esta función es la que utiliza la librería de C5 y que es conveniente que las letras finales sean tratadas como cualquier otra figura gráfica de C5, se optó por utilizar este criterio buscando que la solución propuesta para el diseño de fonts sea coherente con la librería gráfica para lograr una integración completa.

La función port_size(Port p). retorna el máximo max(p_x , p_y), donde p_x es el número de pixels horizontales requeridos por el port y p_y el número de pixels verticales.

Para finalizar el ejemplo de la letra “a”, que se ha continuado a lo largo de este capítulo, en la figura N° 3.49 se puede ver como varía la resolución para los valores de resolución –3, –2 y –1.

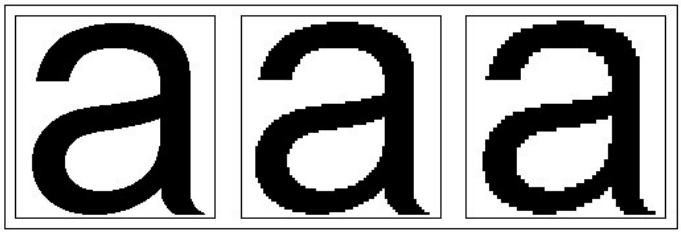


Figura N° 3.49 – Resolución parala letra “a”

Capítulo Nº 4
Prototipo Experimental

En este capítulo se explica la construcción de un prototipo experimental para la solución presentada en el capítulo anterior. En una primera parte se muestra la arquitectura general y cada uno de sus componentes y en la segunda, se realiza la descripción de las principales particularidades de su implementación.

4.1 Arquitectura

En esta sección se presenta una visión general de la arquitectura definida. Además, se identifican los principales componentes del sistema, las entradas, las salidas y el proceso de generación de fonts.

En la definición de la arquitectura, se tomaron como base los siguientes criterios, considerados fundamentales para el buen diseño:

- Abstracción: estudio del problema desde varios niveles de atracción.
- Modularidad: definición de la arquitectura buscando un diseño modular del sistema.
- Extensibilidad: El diseño debe ser lo suficientemente flexible como para que se puedan incorporar nuevas funcionalidades sin mucho esfuerzo.

4.1.1 Descripción general

En la figura N° 4.1 se muestra una visión general de la arquitectura. El rectángulo gris determina el motor lógico del sistema, donde se pueden apreciar los distintos componentes que lo integran, y el rectángulo punteado encierra las librerías que son utilizadas por el motor. El motor recibe dos entradas: una lista de ports y un Tipo de Pares Dependientes; la lista de ports define los sectores de la página sobre los cuales debe desplegarse un símbolo, el cual es especificado por el DPT a través de la determinación de una fuente de letras, un conjunto de parámetros y el caracter particular que debe representarse. La salida del motor está compuesta por una lista de ports, que representa el caracter sobre cada uno de los ports de la lista de entrada.

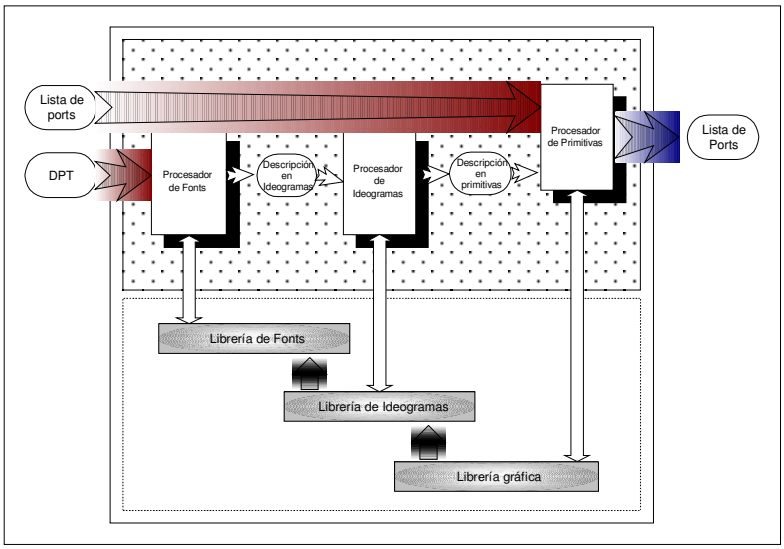


Figura N° 4.1 – Arquitectura general

La idea es definir una librería para cada nivel de abstracción: 1) nivel de primitivas, donde se resuelven los problemas de computación gráfica y se opera directamente con los componentes gráficos de C5; 2) nivel de ideogramas, aquí es donde se tienen los conceptos de curvas o partes de una letra; y 3) nivel de fuentes de letras, donde se ve la letra como la conjunción de ideogramas.

El Motor es el encargado de construir la letra final y para cumplir este objetivo debe utilizar las entradas y los componentes de las librerías. Cada definición de una font utiliza ideogramas y cada ideograma está definido por un conjunto de primitivas gráficas. De esta forma puede verse que el motor debe ir traduciendo las especificaciones de la letra, descendiendo de niveles de abstracción, utilizando para cada traducción un procesador dedicado a tales efectos. Inicialmente recibe la especificación de la letra por medio de un DPT de C5, por lo cual debe identificar cual es la fuente que debe utilizar, el caracter y los parámetros, para buscar en la librería de Fonts, por medio del procesador de fonts, la definición en base a ideogramas de la misma. Tras obtener la definición de la letra en ideogramas, debe realizar la traducción de estos a las primitivas gráficas que los representarán, para lo cual el Procesador de Ideogramas busca la definición de cada ideograma en la Librería de Ideogramas. Finalmente, para realizar la representación de la letra se debe buscar en la Librería de primitivas, a través del Procesador de Primitivas, la definición de cada una de ellas en función de las herramientas de la Open Port Machine de C5. Con las primitivas de C5 y la lista de ports ingresada inicialmente es que se “dibuja” la letra final sobre cada uno de estos.

Por lo tanto, la arquitectura general está dividida en dos grandes componentes: Las Librerías y el Motor. Las Librerías definen los elementos que luego deben ser utilizados por el Motor para generar las letras finales.

Para que el Motor pueda utilizar las librerías, éste debe poder reconocer cada uno de sus elementos, y para lograrlo, cada nuevo elemento de una librería debe registrarse en su procesador correspondiente. Es por este motivo es que los nombres de los procesadores y las librerías están relacionados.

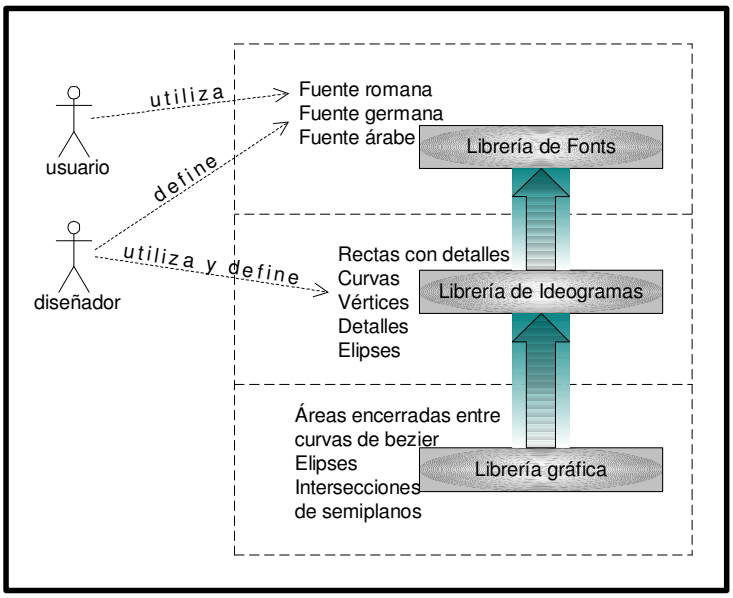
A continuación, se brinda una breve descripción de cada uno de estos dos componentes presentados en la arquitectura general:

4.1.2 Librerías

Las librerías contienen las herramientas que debe utilizar el motor, y estas están dividas en tres niveles de abstracción: Primitivas, Ideogramas y Fonts.

Este diseño se realizó con la finalidad de dividir en 3 capas bien diferenciadas la tarea del diseño de fonts. En la primera capa se manejan los conceptos de computación gráfica, como pueden ser la resolución y la representación de fronteras, mediante la interacción con la librería gráfica de C5. En la segunda capa se utilizan los servicios de la primera para poder definir los "ideogramas", con la finalidad de encapsular en cada uno el comportamiento de cada parte de una letra. Finalmente, la tercera capa es la que contiene los diseños de las fonts, describiendo cada parte de la letra y su comportamiento a través de la utilización de los ideogramas definidos en la segunda capa.

Podemos visualizar esta división en la figura N° 4.2; donde se puede apreciar, que el usuario simplemente utiliza las fuentes de letras definidas en la Librería de Fonts, la cual es mantenida por los diseñadores de fuentes de letras a través de las definiciones de cada font. Además, el diseñador puede llegar a necesitar un nuevo ideograma que modele otro tipo de figura, diferente a las que contiene la librería de Ideogramas. Sin embargo, para definir un nuevo tipo de ideograma, el diseñador debe tener un conocimiento matemático aceptable y además debe conocer la estructura interna del sistema.



Con el uso de estas librerías puede extenderse fácilmente la representación de fonts, ya sea definiendo nuevos ideogramas, en la medida que sean necesarios, o mediante el uso de nuevas primitivas que permitan una definición más completa de los ideogramas. Finalmente, para agregar una nueva fuente de letras, sólo debe agregarse su descripción en la librería de Fonts y registrase en el Procesador de Fonts, sin tener que lidiar con los problemas de más bajo nivel, ya que han sido resueltos por las capas inferiores.

De esta forma, la librería de Primitivas se encarga de utilizar las técnicas de computación gráfica para la representación de letras, la librería de Ideogramas de brindar los componentes constructivos y la librería de fonts de definir, de manera constructiva, cada fuente de letra utilizando los ideogramas existentes.

A continuación se brinda una breve descripción de cada una de las librerías:

***Librería Gráfica.* -** Esta librería contiene las primitivas gráficas de representación y es la herramienta gráfica de más bajo nivel. Opera al nivel de la librería gráfica de C5, y permite la definición de áreas encerradas entre curvas de Bézier, elipses e intersecciones de semiplanos.

Librería de Ideogramas. - Esta librería brinda un nivel más de abstracción que la anterior, de la cual utiliza sus servicios. Es utilizada por el diseñador de fonts para realizar sus diseños y si bien está destinada como una herramienta de diseño de alto nivel, un diseñador experto podría agregar nuevos ideogramas a la librería si lo creeyera necesario.

Librería de Fonts. - Aquí es donde están almacenados los diseños de las fuentes de letras. En estas definiciones se utilizan los ideogramas definidos en la Librería de Ideogramas. Esta es la librería de más alto nivel y es mantenida por los diseñadores de fonts.

4.1.3 Motor

El Motor es el encargado de procesar la entrada para generar el caracter de salida y para esto debe interactuar con las Librerías. Este procesamiento empieza con una descripción en alto nivel de la Letra y realiza sucesivas transformaciones, bajando de nivel en cada caso, hasta llegar al nivel más bajo de abstracción que directamente utiliza las herramientas gráficas de C5. Para cada nivel de procesamiento se define un componente llamado *Procesador*, y cada uno de ellos realiza una transformación entre dos niveles de abstracción, utilizando su librería asociada.

La entrada de cada Procesador define una especificación de la letra en un nivel determinado y los procesadores realizan una traducción de niveles, desde el de mayor al de menor nivel. Por lo tanto, el uso de las librerías se realiza en orden descendente de abstracción, desde la Librería de Fonts hasta la Librería Gráfica, pasando por la Librería de Ideogramas.

Los Procesadores que utiliza el motor son los siguientes:

- **Procesador de Fonts.** – El objetivo de este componente es el de obtener el conjunto de árboles de ideogramas (árboles_curva) que conforman la letra. Éste debe buscar en la librería de Fonts, a partir del DPT de entrada, la fuente de letras y dentro de éste el caracter, para finalmente crear cada árbol de ideogramas de la letra. La salida contiene la letra descripta a través de sus árboles de ideogramas. En la siguiente figura se pueden apreciar gráficamente las entradas y salidas que tiene el Procesador de Fonts.

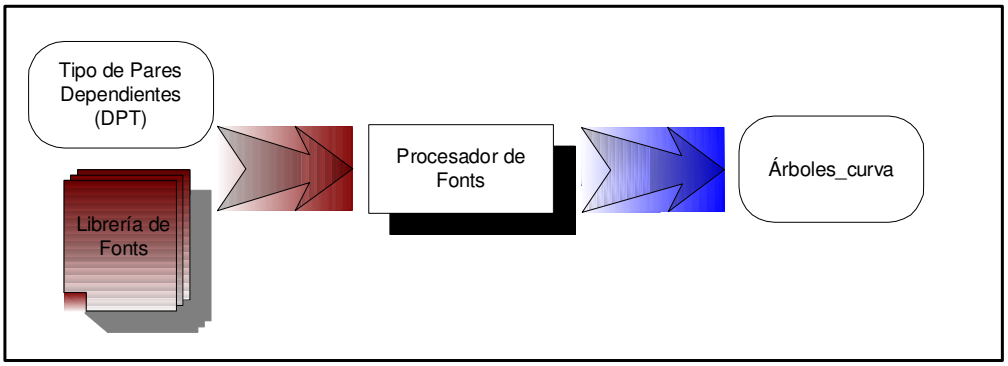


Figura N° 4.3 – Procesador de Fonts

Por lo tanto, la tarea del Procesador de Fonts es bajar un nivel en la descripción de la letra, desde una definición simple que contiene el nombre de la fuente, un conjunto de parámetros y el caracter que se debe definir, a la creación de las instancias de los árboles de ideogramas que conforman la letra. La creación de estos árboles se realiza consultando la librería de Fonts, la cual contiene las descripciones de cada letra en función de las curvas_árbol que la conforman.

- **Procesador de Ideogramas.** – Este componente recibe un conjunto de árboles_curva, los cuales debe recorrer en preorden, hallando en cada paso la lista de figuras que representa cada ideograma. Al ser cada árbol recorrido en preorden, se obtienen las salidas de un nodo antes de procesar sus hijos, por lo tanto en cada procesamiento se le deben establecer a sus hijos las entradas que provienen de las salidas recién halladas. Cada figura representa una primitiva gráfica sobre un sector de la letra, por lo que la lista de figuras que se obtiene luego de procesar un nodo se corresponde con la representación del ideograma a través de las primitivas gráficas. En la figura N° 4.4 se muestran las entradas y salidas del Procesador de Ideogramas.

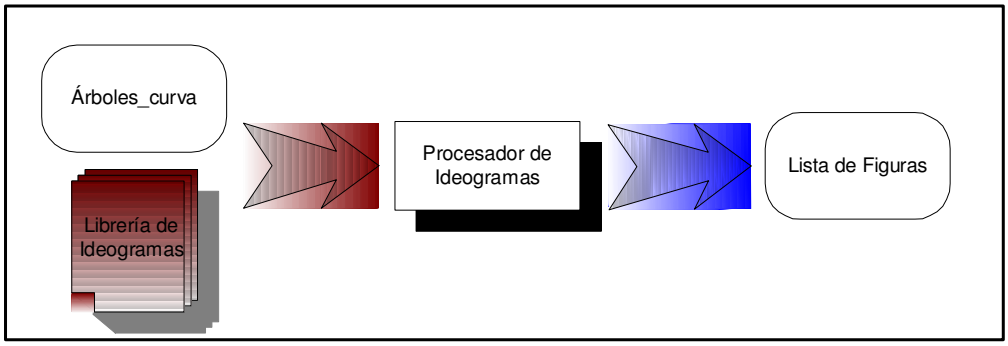


Figura N° 4.4 – Procesador de Ideogramas

Puede apreciarse entonces, que el Procesador de Ideogramas disminuye el nivel de abstracción de la representación de la letra, al traducir la descripción de esta en función de árboles_curva a una representación como lista de primitivas gráficas, para lo cual utiliza la Librería de Ideogramas.

- **Procesador de Primitivas.** – Luego de haber obtenido la lista de primitivas, éstas deben utilizarse para describir la letra sobre la lista de ports ingresada y para ello es que el Procesador de Primitivas utiliza la librería gráfica. Las entradas y salidas de este procesador pueden apreciarse en la figura N° 4.5.

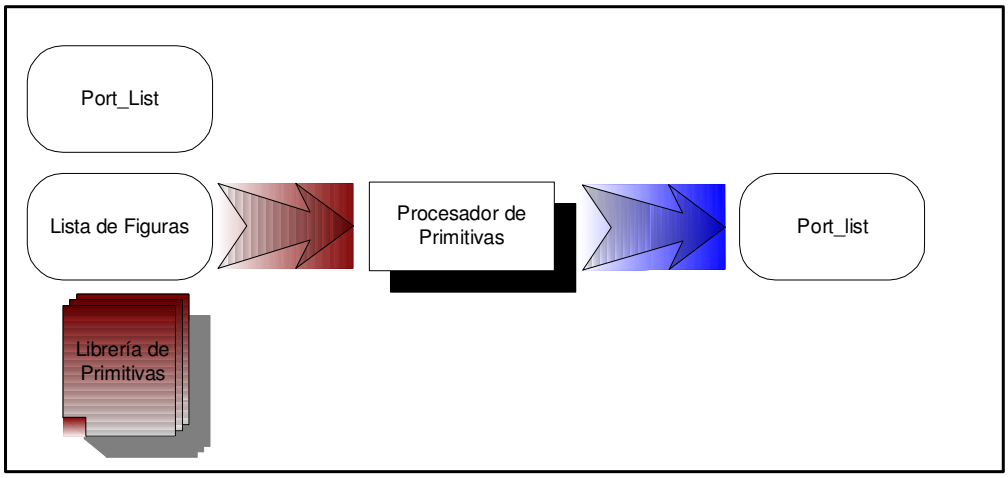


Figura N° 4.5 – Procesador de Primitivas

Este procesador representa el último eslabón en la cadena de traducciones del Motor, produciendo como resultado la lista de ports que define la letra como elemento gráfico.

Cada una de las figuras de la lista de entrada está compuesta por la especificación de un puerto y la identificación de una función de C5 y los valores de los parámetros que esta utiliza. La tarea del Procesador de Primitivas es simplemente la de aplicar cada función sobre la lista ports, a través de las definiciones contenidas en la Librería de Primitivas.

Cada puerto, representa el port que debe obtenerse de cada port de la lista para poder utilizar la primitiva sobre él. Por lo tanto, aquí puede verse que un Puerto de la solución se corresponde con un port de C5.

4.2 Implementación

La implementación fue realizada en el lenguaje de programación C5; sin embargo, debido a que no existe un debugger apropiado para C5 y a que éste es una extensión de C, se decidió realizar la mayor parte de la codificación en C y restringir a las primitivas gráficas y al Procesador de Primitivas la utilización de C5. De esta manera se encapsula en los módulos de la Biblioteca de Primitivas, y en los procesadores de Primitivas y Fonts las funciones gráficas y la utilización de Tipos de Pares Dependientes, permitiendo depurar de forma más sencilla el resto del código.

A continuación se describe la estructura de la implementación.

4.2.1 Descripción de paquetes

El código está estructurado en 3 paquetes principales:

- Estructura: contiene todos los tipos abstractos de datos que utilizan el Motor y las Librerías.
- Librerías: Este paquete está compuesto de tres sub–paquetes, cada uno de los cuales está dedicado a cada una de las librerías.
- Motor: Aquí es donde se definen las funciones lógicas del motor y está dividido en cuatro sub-paquetes: uno para el motor general y de los tres restantes uno para cada procesador.

La lógica de la aplicación está contenida en el paquete Motor, el cual define un sub-paquete para cada procesador y un paquete *Principal* que representa el motor global, el cual utiliza cada uno de los procesadores para lograr la salida esperada.

En la figura N° 4.6 se puede apreciar la clasificación en paquetes del sistema.

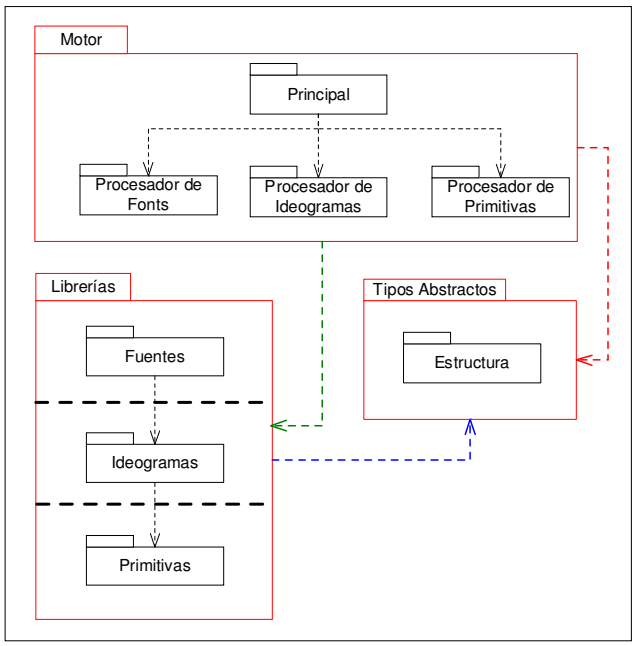


Figura N° 4.6 – Diagrama de Paquetes

El Motor es el encargado de generar las letras y, por lo tanto, tiene visibilidad sobre cada uno de los restantes módulos, mientras que la librería de Primitivas sólo tiene visibilidad sobre el paquete Tipos Abstractos; y el paquete Tipos Abstractos no tiene visibilidad sobre ningún otro paquete, debido a que su única función es la de proveer los tipos abstractos de datos necesarios para la implementación de la solución.

En la figura N° 4.7 pueden apreciarse, más en detalles, las vinculaciones entre paquetes.

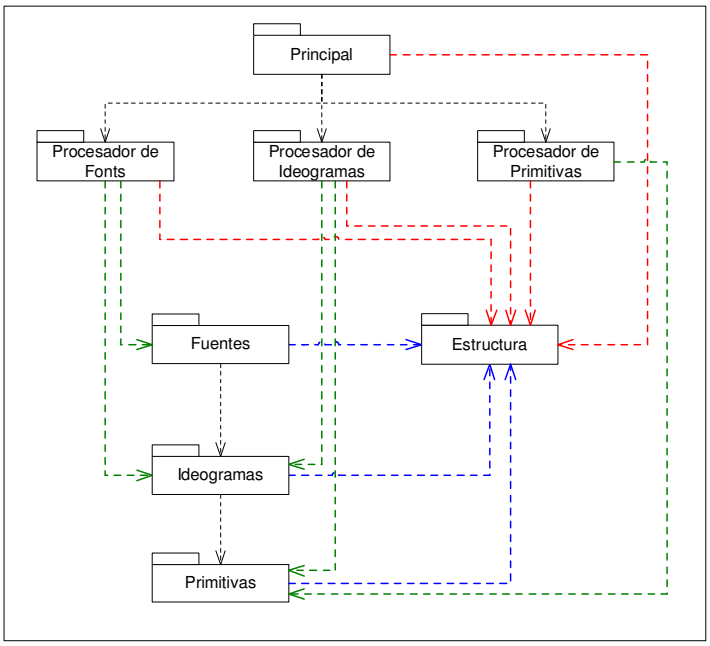


Figura N° 4.7 – Dependencias entre paquetes

Todos los paquetes deben tener visibilidad sobre el paquete Estructura, puesto que éste contiene todos los tipos abstractos de datos que permiten las relaciones entre módulos. El paquete Principal debe tener visibilidad sobre los cuatro procesadores, pues éste es el que simula el motor general y por lo tanto debe utilizar los procesadores en secuencia para lograr el resultado final. A su vez, cada paquete Procesador debe tener visibilidad sobre las librerías gráficas que utiliza, y por ello, el paquete Procesador de Fonts tiene visibilidad sobre el paquete Fuentes y el Paquete de Ideogramas; el paquete Procesador de Ideogramas sobre los paquetes Librería de Ideogramas y Librería de Primitivas y el paquete Procesador de Primitivas sobre la Librería de Primitivas. Finalmente, las librerías tienen las visibilidades, entre ellas, establecidas por el diseño de capas.

Como los paquetes *Motor* y *Librerías* simplemente realizan las funciones lógicas que ya se han explicado en la arquitectura, a continuación se dará una breve descripción de los tipos abstractos de datos que son los que modelan cada una de las particularidades de la solución.

4.2.2 Tipos Abstractos de Datos

Se ha definido un tipo abstracto de datos para modelar cada una de las particularidades de la solución:

- Letra
  Con este TAD se modela el conjunto de árboles_curva que define una letra para la fuente.
- Ideograma
  Representa un ideograma, permitiendo la identificación de cada uno de ellos y el modelado de sus particularidades.
- Función
  Define una primitiva gráfica y sus parámetros, para lo cual tiene un identificador de primitiva que debe ser establecido con el criterio que utiliza el procesador de primitivas para poder detectar a cual de ellas se hace referencia.
- Puerto
  Este TAD modela un puerto sobre el área en que debe desplegarse la letra.
- Pendiente
  Se utiliza para definir una entrada o una salida.
- Curva
  Modela uno de los árboles_curva que componen la letra.
- Detalle
  Contiene las características de detalle que se deben utilizar.

Luego de ver estos tipos abstractos de datos es que se debieron definir los siguientes:

- Diccionario_Ideogramas
  Para poder describir las relaciones de entrada_salida entre los ideogramas debe haber una forma de identificarlos en el árbol_curva. Por esto es que se creó este tipo que debe ser utilizado por el tipo *curva*.
- Curvas
  Como cada letra puede tener la cantidad de árboles_curva que se desee, es que se utiliza este tipo que define una lista de árboles_curva.
- Figura
  Una figura representa una primitiva gráfica sobre un sector de la letra y, por lo tanto, debe utilizar los tipos *Función* y *Puerto*.

- Parámetros
  Los *Ideogramas* y las *Primitivas* deben almacenar un conjunto de parámetros de tipos enteros y de punto flotante; por este motivo se creó el tipo *Parámetros* que contiene una cantidad arbitraria de ellos.

En la figura N° 4.8 se muestran las relaciones entre los tipos abstractos de datos.

Como puede verse en el siguiente diagrama, los ideogramas no utilizan directamente las primitivas gráficas, si no que lo hacen a través de un identificador y una colección de parámetros.

Luego el procesador de ideogramas obtiene esta descripción y la vincula con las primitivas de la Biblioteca Gráfica, que son representadas como una lista de figuras. De la misma forma, cada figura contiene la descripción de las funciones de C5 y su vinculación es resuelta por el procesador de primitivas.

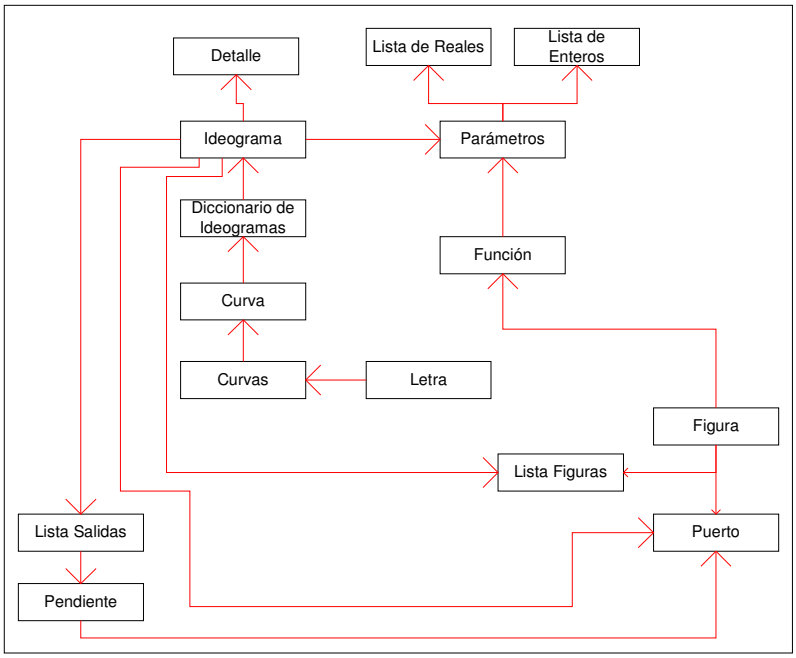


Figura N° 4.8 – Tipos Abstractos de Datos

Capítulo Nº 5
Experimentación

Como se señaló en los capítulos anteriores, uno de los objetivos de este proyecto es la construcción de un prototipo experimental; el cual se debe utilizar como herramienta para la evaluación del diseño realizado.

Al inicio de este documento nos habíamos planteado una serie de objetivos y luego de haber analizado el problema y diseñado una solución debemos preguntarnos si estos se han cumplido.
Las preguntas que debemos contestar son las siguientes:

- ¿Es posible representar distintos tipos de fuentes de letras sin que esto signifique un gran esfuerzo?
- ¿Los diseños de fonts pueden parametrizarse con suficiente flexibilidad?
- ¿El diseñador puede hacer sus representaciones de manera natural sin tener que manejar muchos conceptos matemáticos?
- ¿Qué nivel de resolución y calidad tiene las representaciones?
- ¿Qué tan costosa es la generación de letras?

Estás preguntas sólo pueden contestarse por medio de la práctica y por eso es que en este capítulo intentaremos diseñar un conjunto representativo de caracteres para una fuente de letras romana, que permita evaluar las preguntas.

5.1 Generación de la fuente romana

La fuente de letras que se definirá es de tipo romano, dado que es la más utilizada. En la creación de esta fuente se buscará seleccionar un conjunto de caracteres lo suficientemente representativo, de forma tal que la definición de cualquier otra letra, no incluida en este conjunto, sea fácilmente especificada a partir de las características comunes con estas.

Inicialmente se analizará la representación general de un carácter sobre una zona de la página, determinando las posiciones que deben respetarse para producir un resultado homogéneo.

También se buscará definir un conjunto de parámetros que permita parametrizar las propiedades globales más significativas de la fuente. Se hará especial hincapié en los detalles de las letras y en los diferentes parámetros que se deben utilizar para modelar los anchos de cada una de sus partes y controlar la inclinación general.

Luego se explicarán los criterios utilizados para seleccionar el subconjunto de letras y se mostrará como es que cada una de ellas fue construida.

Finalmente, se explicará una posible representación de los caracteres por medio de un string, para poder apreciar y evaluar los resultados.

Se aclara que las figuras que se muestran en esta sección no son fruto de la salida de la solución, si no que son utilizadas como modelo para la selección y definición del subconjunto escogido.

5.1.1 Consideraciones Generales

En esta sección se presentan las características generales que deberá tener la fuente. Inicialmente ser verán los criterios generales de representación, luego se explicarán las decisiones que se tomaron para definir los parámetros que se deben utilizar y finalmente la forma en que se utilizarán los detalles.

5.1.1.1 Representación general

En la representación de los caracteres deben establecerse las posiciones y comportamientos generales de una forma tal que no se produzcan resultados indeseados y se mantenga la coherencia entre las letras.

Para la representación de los caracteres, vemos que las letras mayúsculas tienen el mismo punto superior e inferior entre sí, mientras que para las letras minúsculas esto no es así y algunas tienen su punto superior igual al de las mayúsculas y otras no; además algunas letras llegan a puntos que están debajo de lo que sería el renglón. Si bien algunas letras son más altas que otras, podemos apreciar que todas tiene su punto más alto en una de dos posiciones posibles. Letras como la 'l', 'f' o 'h' llegan a la misma altura que las letras mayúsculas, pero otras como: 'a', 'c', 'e' o 'n' tienen un punto superior inferior. Por otra parte, letras como la 'g' o 'j' tienen un punto inferior menor que el resto.

En la figura N° 5.1 se muestra la determinación de los niveles de control para las letras. El nivel_inferior determina el punto más bajo que puede tener una letra y el nivel_medio_inferior lo que sería el renglón de un cuaderno, donde se apoyan las definiciones de cada una de ellas. Todas las letras mayúsculas tienen su punto superior coincidente con el nivel_superior, mientas que las minúsculas pueden tener su punto superior al igual que el de las mayúsculas o a la altura determinada por el nivel_medio_superior (por ejemplo; a, c o e).

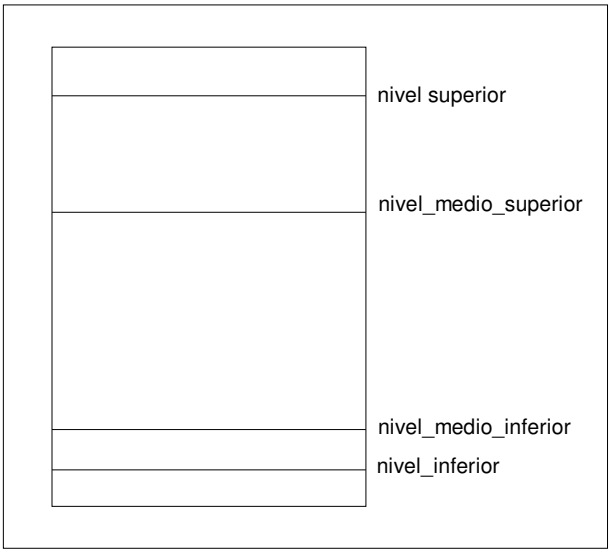


Figura N° 5.1 – Niveles de representación

En la figura N° 5.2 se puede apreciar un ejemplo para las letras 'A', 'a' y 'q'.

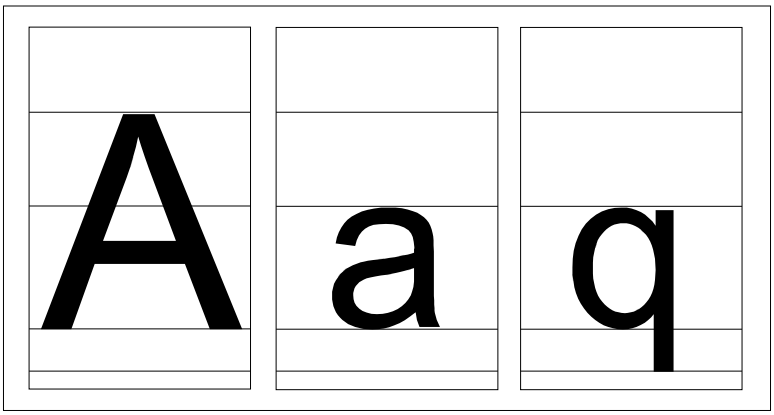


Figura N° 5.2 – Representación general

De esta forma, las características verticales de las letras están determinadas por estos parámetros de nivel. No es así el caso de las dimensiones horizontales, pues no tienen una característica común y el ancho de cada una de ellas depende exclusivamente de su definición.

Por lo tanto, para cada caracter se debe tener en cuenta en su definición que los puntos superiores e inferiores de cada letra deben respetar los niveles, los cuales son determinados en el momento de definir el tipo de fuente a través de un parámetro particular para cada uno.

5.1.1.2 Elección de los parámetros

Los parámetros para controlar la forma de una fuente son los que luego serán utilizados por el usuario para determinar el tipo de letra que desea, puesto que en este contexto un tipo es una instancia de la fuente, y cada instancia está determinada por los valores de los parámetros.

Inicialmente podemos apreciar que existen dos clases de parámetro claramente diferenciadas: una para aquellos que determinan el comportamiento general de la letra y otra para los que definen las características de los detalles. Por eso, dividimos la parametrización en estos dos grupos, que se explican a continuación.

Parámetros generales

Para poder controlar la forma de la letra se deben proveer parámetros que permitan modificar la inclinación general, los anchos de cada parte, la resolución con la cual desplegar cada carácter y la determinación de los niveles que se deben representar para obtener un comportamiento homogéneo de estas. A continuación se presenta la determinación de los parámetros para cada caso.

Inclinación

Entre los parámetros generales de la fuente podemos ver que es importante definir uno que controle la inclinación de las letras y, de esta manera, poder simular las propiedades usualmente utilizadas de itálica o cursiva. Sin embargo, se pretende realizar algo más general y no quedarse simplemente en una inclinación booleana del tipo {con inclinación, sin inclinación}. Por este motivo, se decidió utilizar un parámetro de punto flotante cuyo dominio es [-1, 1] y, de esta forma, definir la inclinación tanto hacia la izquierda como hacia la derecha. Los valores negativos son los que definen la inclinación hacia la izquierda, mientras que los positivos los hacen hacia la derecha, dejando el valor 0 para indicar la falta de inclinación. En la figura N° 5.3 se puede apreciar un ejemplo para la letra H, con los valores de inclinación –1, -0.5, 0, 0.5 y 1.0 para las figuras a, b, c, d y e respectivamente.

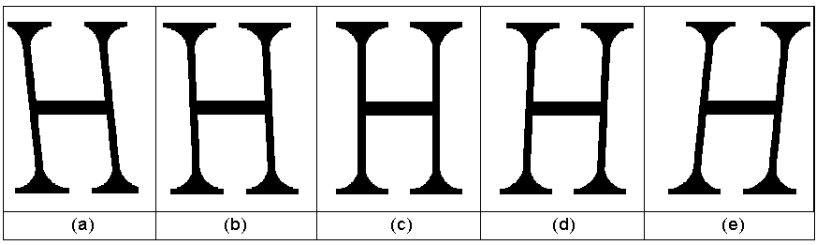


Figura N° 5.3 - Inclinaciones

Para implementar la inclinación se debe considerar que todas las letras de una misma fuente deben variar su inclinación de la misma forma. Para esto se definió un valor constante *"ancho_inclinacion"*, con el cual se determina la magnitud que deben variar las coordenadas x´s de los puntos de control de la letra, teniendo en cuenta además las coordenadas y. La forma en que se deben utilizar las coordenadas de cada punto junto con el parámetro de inclinación y el ancho_inclinación es la siguiente:

x = x + (inclinación) . (ancho_inclinacion) . (y – (nivel_superior + nivel_medio_inferior)/2).

Por lo que si el valor del parámetro inclinación es 0, la coordenada mantiene su valor; mientras que si el valor es positivo, aumenta si y>(nivel_superior + nivel_medio_inferior)/2 y disminuye si y<(nivel_superior + nivel_medio_inferior)/2; sucediendo lo contrario si el parámetro inclinación es negativo. Si el punto pertenece a la recta horizontal que está justo entre el nivel_medio_inferior y el nivel_superior, entonces tampoco varía su coordenada x .

Resolución

Todos los ideogramas tienen un parámetro común llamado rec_nr, que calcula recursivamente la resolución que deben tener en función del tamaño del puerto. Sin embargo, el control de la resolución a nivel de ideogramas lo único que hace es utilizar las primitivas gráficas con éste parámetro, ya que a nivel de la Librería de Primitivas es manejada la resolución. Por lo tanto, a nivel de fonts haremos lo mismo que a nivel de ideogramas, simplemente definiremos un parámetro rec_nr, el cual será utilizado en cada ideograma para establecer la resolución esperada.

El comportamiento de este parámetro es el mismo que el que se utiliza a nivel de primitivas:

- Números positivos.- resolución manual, donde cada valor indica la dimensión el tamaño mínimo de los ports con los cuales se debe representar cada primitiva. Un valor de 1, representa cada primitiva con un tamaño igual al de su port original, mientas que un valor mayor representa cada primitiva con ports de tamaño $1/2^{i}$ del port original.

- Números negativos.- resolución automática, donde un valor de –1 utiliza el valor de resolución estándar para la representación de la letra y un valor menor utiliza el valor obtenido de recnr-1 y lo incrementa en 1. El valor de rec_nr determina el tamaño mínimo de los ports que se deben utilizar en función del tamaño del port en el que se debe representar la letra y el tamaño de estos ports es $1/2^{rec\_nr}$ del tamaño original del port. Normalmente los valores de resolución automática que se utilizan son –1 , –2 y –3, pues con valores mayores se puede producir una resolución tan pequeña que no sea representable sobre la página.

Luego de ver los posibles valores que pueden utilizarse para el parámetro rec_nr, se hace evidente que el uso de un valor positivo no permite controlar eficazmente la resolución final de la letra, puesto que este valor indica el tamaño mínimo de los ports que utiliza cada primitiva en función de los ports originales sobre los cuales debe desplegarse; y como no se utiliza un tamaño de port uniforme para todas las primitivas, si no que cada una es definida sobre el port más conveniente, la letra final podría estar compuesta por partes con distinta resolución.

Este problema podría solucionarse si en la definición de cada ideograma se tuvieran en cuenta los tamaños de los puertos en que debe ser representada cada parte de éste por medio de una primitiva; sin embargo, para que esto pueda ser una solución al problema, en la Librería de Fonts debería utilizarse el mismo procedimiento, ya que de nada sirve que se "arreglen" las resoluciones a nivel de ideogramas, si ya vienen desajustadas desde las definiciones de letras.

Como uno de los objetivos del proyecto es el de brindarle al diseñador la posibilidad de realizar sus diseños de una forma natural y sin muchos problemas, se decidió restringir el manejo de la resolución a un valor automático. Es así que el rango de valores para el parámetro rec_nr que debe utilizarse en la instanciación de una fuente de letras debe ser negativo, y para evitar resultados indeseados, internamente en la definición de los ideogramas, si el valor ingresado es positivo simplemente se utiliza su opuesto para realizar las representaciones. Por lo tanto, podrán emplearse tanto valores positivos como negativos en la utilización de la fuente, siendo el resultado de un valor ingresado el mismo que el de su opuesto.

Anchos

Otro de los parámetros importantes a definir para controlar el comportamiento de las fuentes, son los relacionados con las dimensiones de cada parte de la letra. El criterio que se utiliza es el de dividir la letra en clases de direcciones, determinando los siguientes parámetros:

- Anchos de líneas horizontales.
- Anchos de líneas verticales.
- Ancho de líneas inclinadas hacia la derecha (desde inferior izquierda a superior derecha).
- Ancho de líneas inclinadas hacia la izquierda (desde inferior derecha a superior izquierda).
- Ancho de los vértices.

Puntos superiores, puntos inferiores y altura del renglón

La representación general de un caracter debe estar controlada por el nivel del renglón y por los puntos de altura máxima y mínima. Para el caso de las letras más altas, su punto máximo estará sobre el nivel_superior del puerto y en el caso de las letras que van más abajo del "renglón", estas tendrán su punto inferior sobre el nivel_inferior del puerto. Mientras que el renglón" y el nivel superior de las letras minúsculas más pequeñas están determinados por los parámetros nivel_medio_inferior y nivel_medio_superior respectivamente.

Detalles

Finalmente se definieron los parámetros que determinan el tipo detalle que se debe utilizar; estos son los siguientes:

- Ancho del detalle.- determina el ancho que tendrá el detalle como proporción del ancho del puerto de la letra.
- Altura del detalle.- determina la altura del detalle como proporción del largo del puerto.
- Ancho final del detalle.- determina al ancho final que tendrá el detalle como proporción del ancho o alto del puerto, según el tipo de detalle que sea.
- Detalle curvo.- Indica si se debe o no realizar un detalle curvo.
- Final curvo.- Determina que se dibuje una curvatura sobre el final del detalle o que este sea recto.
- Grosor.- Determina que tan "grueso" es el detalle curvo.

Con estos parámetros se parametriza cada letra de la fuente. En cada parte representada por un ideograma, es necesario determinar aquellos anchos, inclinaciones y detalles que se deben utilizar y de qué forma relacionarlos entre sí.

5.1.2 Selección del conjunto de caracteres

Como exponer la definición de cada una de estas letras sería muy extenso, en este documento sólo definimos aquellos caracteres más representativos, en los que podamos resumir las particularidades de cada uno de los restantes caracteres que componen la fuente.

La selección del subconjunto de caracteres es muy importante, ya que debe ser representativo de todos los caracteres de la fuente, de forma tal que en la definición de uno de los restantes no aparezcan nuevas particularidades.

Esta decisión se tomó teniendo en cuenta los tipos de curva y de detalles presentes. Entre las características generales distinguimos las siguientes:

- Curvas (aquellas que no necesariamente están acotadas por límites rectos).
- Intersecciones de semiplanos con distintas inclinaciones, diferenciando entre aquellas verticales, horizontales, inclinadas hacia la izquierda e inclinadas hacia la derecha.
- Detalles horizontales, Detalles verticales, detalles iniciales y detalles finales.

A continuación, se realizará una clasificación de letras en función de las clases de curvas que contienen, identificando en cada letra aquellas partes que se corresponden con cada una de las características recién mencionadas. Puede apreciarse como cada una de estas partes se corresponde con uno de los ideogramas definidos, demostrando la eficiencia que se obtuvo al haber realizado la selección de los ideogramas del punto 3.8.

Curvas

Las letras mayúsculas que contienen curvas son: ‘B‘, ‘C‘, ‘D‘, ‘G‘, ‘J‘, ‘Ñ‘, ‘O‘, ‘P‘, ‘Q‘, ‘R‘, ‘S‘, ‘U‘. Para las minúsculas se tienen más letras y esto se debe a que las minúsculas son figuras con una forma más curva que las mayúsculas. Las minúsculas son: ‘a‘, ‘b‘, ‘c‘, ‘d‘, ‘e‘, ‘f‘, ‘g‘, ‘h‘, ‘j‘, ‘m‘, ‘n‘, ‘ñ‘, ‘o‘, ‘p‘, ‘q‘, ‘r‘, ‘s‘, ‘t‘, ‘u‘, ‘y‘. En la figura N° 5.4 se muestran las curvas, para cada una de las letras mencionadas, en colores rojo y naranja.

Figura N° 5.4 Partes curvas de la fuente romana

Intersecciones de semiplanos

Las intersecciones de semiplanos son ampliamente utilizadas en las letras mayúsculas debido a que estas son más rectas que las minúsculas. Sin embargo, estas otras también tienen figuras rectas, sobre todo en aquellas partes que definen detalles. Entre las mayúsculas tenemos las siguientes letras: ‘A‘,‘B‘,‘E‘, ‘F‘, ‘G‘, ‘H‘, ‘I‘, ‘J‘, ‘K‘, ‘L‘, ‘M‘, ‘N‘, ‘Ñ‘, ‘P‘, ‘R‘, ‘T‘, ‘U‘, ‘V‘, ‘W‘, ‘X‘, ‘Y‘, ‘Z‘; y entre las minúsculas: ‘e‘, ‘f‘, ‘h‘, ‘i‘, ‘j‘, ‘k‘, ‘l‘, ‘m‘, ‘n‘, ‘ñ‘, ‘p‘, ‘q‘, ‘r‘, ‘u‘, ‘y‘. En la figura N° 5.5 se muestran, en rojo y naranja, las partes de cada letra que están compuestas por intersecciones de semiplanos.

Figura N° 5.5 – partes rectas de la fuente romana

Detalles Horizontales

Los detalles horizontales son los más utilizados tanto en las letras minúsculas como en las mayúsculas. Entre las mayúsculas tenemos las siguientes: 'A', 'B', 'D', 'E', 'F', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'Ñ', 'P', 'R', 'T', 'U', 'V', 'W', 'X' e 'Y'; y entras las minúsculas: 'f', 'h', 'i', 'k', 'l', 'm', 'n', 'ñ', 'p', 'q', 'r', 'u', 'v', 'w', 'x' e 'y'. Se les puso el nombre de horizontales, debido a que sus bases se apoyan sobre los lados horizontales del puerto. En la figura N° 5.6 se muestran los detalles horizontales de la fuente romana, en color rojo.

Figura N° 5.6 – Detalles horizontales de la fuente romana

Detalles Paralelos

Estos detalles son los que se producen sobre rectas horizontales y definen un ángulo respecto a la vertical. Si bien este tipo de detalle se utiliza en pocas letras, su comportamiento es tal que debe ser diferenciado del resto de detalles. La única letra minúscula que utiliza este detalle es la

‘z‘; mientras que entre las mayúsculas tenemos las siguientes: ‘E”, ‘F‘,‘L‘,‘T‘ y ‘Z‘. En la figura N° 5.7 se muestran los detalles paralelos, para las letras mencionadas, en color rojo.

Figura N° 5.7 – Detalles paralelos para la fuente romana

Detalles iniciales y finales

Los detalles iniciales y finales son aquellos que no son parte de una recta particular, si no que determinan el comienzo o el final de una curva. Las únicas letras que utilizan estos tipos de detalles son las siguientes: ‘C‘, ‘S‘, ‘c‘ y ‘s‘. Estos detalles también podrían ser utilizados en lugar de los detalles paralelos; sin embargo, cada uno está controlado por un conjunto de parámetros que permiten manejar los cambios de su forma para el tipo de detalle especifico. En la figura N° 5.8 se muestran los detalles iniciales y finales de la fuente romana, en color rojo.

Figura N° 5.8 – Detalles iniciales y detalles finales para la fuente romana

Clases de letras e ideogramas

Luego de observar los distintos tipos de figuras que puede contener una letra, tenemos que tener en cuenta los ideogramas disponibles para realizar la definición.

Existe un ideograma para cada concepto básico de los nombrados: curvas, intersecciones de semiplanos y detalles, Además, se agregó un ideograma que define la diferencia entre elipses concéntricas, con el cual se pueden “dibujar“ curvas cerradas sin tener que realizar una composición por sectores.

Lo ideal sería tener un ideograma para cada parte de la letra; sin embargo, debido a que un ideograma debe estar contenido en un rectángulo y que sus relaciones con los demás ideogramas están determinadas sobre los lados de éste, no se puede realizar una descomposición tan sencilla en ideogramas. Para resolver este inconveniente, en la etapa de diseño se consideraron las posibles formas que producirían estos problemas, definiéndose un ideograma para resolver cada uno de ellos. Finalmente, sólo dos particularidades no podían ser representadas con los ideogramas existentes, por lo que se definieron ideogramas que permiten resolver estos problemas sin mucha complicación y de una forma bastante natural.

Dos de estos ideogramas definen un vértice con sus dos “rectas” dentro del mismo puerto (Vertice_Recto_Detalles y Continuación_Vértice_Detalles). Estos facilitan la creación de vértices sin que ocurran los problemas que se muestran en la figura N° 5.9, dado que no puede definirse un vértice con “rectas” de distinto tamaño, pues ellas deberían unirse sobre el segmento de recta común entre sus dos puertos . Uno de estos dos ideogramas continúa una de las “rectas” hasta llegar al vértice, brindando la posibilidad de elegir la posición final de salida de la otra recta, mientras que el otro comienza en la definición del vértice y tiene como salidas cada una de las “rectas”.

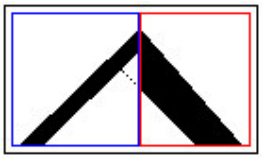


Figura N° 5.9 – Problema en la definición de vértices

El otro ideograma complementario define una intersección de semiplanos y una curva en el mismo puerto (Semiplanos_Curva). De esta forma se permite utilizar estas figuras al mismo tiempo sin que haya que tener en mente que deben unirse sobre el lado común de sus puertos. Éste es muy útil para definir esquinas y algunos tipos de vértice que no es posible definir con el ideograma anterior. Por ejemplo, la esquina superior izquierda de la letra F (Figura N° 5.10 (a)) define un vértice entre los dos segmentos de recta, y una primera solución sería pensar en definir dos ideogramas como intersecciones de semiplanos (figura 5.10 (b)); sin embargo, al definir una inclinación para la letra se presentaría el problema que se muestra en la Figura 5.10 (c). Para solucionar este inconveniente es que utiliza el ideograma recién mencionado, el cual define dos curvas que se unen sobre el mismo ideograma.

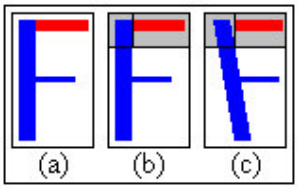


Figura N° 5.10 – Problemas con la inclinación de vértices

Teniendo en cuenta la clasificación realizada para las letras y los ideogramas disponibles para representarlas, es que se definió el siguiente subconjunto: {‘A‘, ‘V‘, ‘M‘, ‘E‘, ‘S‘, ‘O‘, ‘n‘, ‘b‘, ‘j‘}. En estas letras se puede observar que tienen las características de semiplanos (‘A‘, ‘V‘, ‘M‘, ‘E‘, ‘n‘, ‘b‘, ‘j‘), curvas (‘S‘, ‘O‘, ‘n‘, ‘b‘, ‘j‘), elipses concéntricas (‘O‘), detalles paralelos (‘E‘), detalles horizontales (‘A‘, ‘V’ ‘E‘, ‘M‘, ‘n‘), y detalles iniciales y finales (‘S‘). También puede observarse como es posible utilizar vértices para las letras ‘A‘, ‘V‘ y ‘M‘ e intersecciones de semiplanos y curvas en el mismo puerto (‘A‘, ‘E‘, ‘n‘, ‘b‘).

5.1.3 Representación del conjunto de caracteres

En esta sección explicaremos como fue definido cada caracter del subconjunto seleccionado. Sin embargo, sólo se dará una explicación precisa para la letra A, mientras que para el resto se explicarán los rasgos generales de cada definición.

A continuación se explica la definición del subconjunto seleccionado.

Letra ‘A’

En la definición de la letra “A” podemos ver que se tienen dos rectas inclinadas, y una horizontal; además, también podemos ver que se debe utilizar un vértice en su parte superior. En la figura N° 5.11 podemos apreciar los ideogramas que componen la letra.

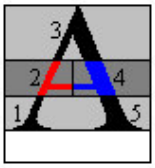


Figura N° 5.11 – Letra “A”

Esta letra la definiremos con un único árbol_curva, que empezará con la recta izquierda y terminará con la derecha. Inicialmente definimos la recta izquierda (ideograma 1) con una inclinación dada que se obtiene del parámetro inclinación, luego su salida, será tomada por otro ideograma (ideograma 2) que continúa la recta pero definiendo además una salida sobre el lado derecho del puerto a una altura determinada. La salida que está ubicada sobre el lado superior del puerto será tomada luego por un ideograma que continúa la recta para formar un vértice (ideograma 3); la salida de este ideograma, que representa la otra recta del vértice debe ser escogida sabiendo que ésta determina la inclinación de la segunda recta de la letra. Luego la salida es tomada por el ideograma 4, el cual define también una recta horizontal a la misma altura y del mismo ancho que la del ideograma 2, formando de esta forma la recta horizontal de la letra. Finalmente, el ideograma 5 continúa la salida inferior del ideograma anterior para formar la segunda “pata” de la letra.

En la figura N° 5.12 se pueden apreciar los puntos de control escogidos para formar la letra.

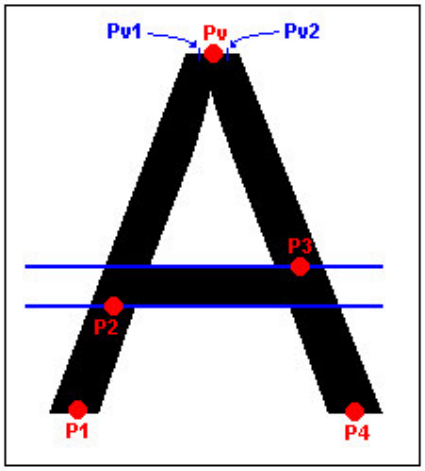


Figura N° 5.12 – Punto de control para la letra “A”

Estos puntos de control son los que definen el esqueleto de la letra y la elección de ellos debe ser realizada teniendo en cuenta la forma de la letra y además las posibles inclinaciones que esta puede tener.

Los puntos P1 y P4 son determinados inicialmente para definir el ancho de la letra, estando los dos a la misma distancia del centro horizontal del puerto. Mientras que el punto Pv está sobre el nivel superior y ubicado horizontalmente en el centro.

Para definir los puntos P2 y P3, debemos primero determinar los puntos del nivel superior (Pv1 y Pv2) que pertenecen a las rectas [P1, P2] y [P3, P4] respectivamente, de forma tal de respetar la posición del punto Pv junto con los anchos de cada recta. Es así que se llega a la siguiente definición de los puntos de control:

P1 = (x_left, nivel_medio_inferior)
P4 = (x_right, nivel_medio_inferior)
Pv = ((x_right + x_left)/2, nivel_superior)
Pv1 = (xv-ancho_vertice/2+ancho_right/2, yv)
Pv2 = (xv+ancho_vertice/2-ancho_left/2, yv)
P2 = (x1 + (xv1-x1)*((y2-y1)/(yv-y1)), 0.5-ancho_horizontal/2)
P3 = (xv2 + (x4-xv2)*((y3-yv)/(y4-yv)), 0.5+ancho_horizontal/2)

El manejo de las inclinaciones debe realizarse contemplando el parámetro inclinación. La transformación de los puntos de control de acuerdo a la inclinación se realiza utilizando la siguiente función:

xn = xn + inclinacion* ancho_inclinacion* (yn-((nivel_superior+ nivel_medio_inferior)/2))

Como esta es la función que se debe utilizar para implementar la inclinación, es necesario definir el ancho_inclinación que se debe utilizar para definir la fuente de letras. En general se realiza por medio de una constante, que en este caso llamaremos ancho_inclinación_romanas.

Dado que la figura cambia su inclinación en función de sus puntos de control, primero se definen estos en función de la letra sin inclinación, y luego se utiliza la función de cálculo, arriba mencionada, para acomodarse a la nueva inclinación. De esta forma el cambio en la inclinación es controlado con facilidad y mediante una cuenta sencilla.

Para la definición de la fuente romana se utiliza un ancho de inclinación correspondiente al 20% del ancho del puerto.

Como ya se mencionó, los parámetros que se utilizan en la definición de los caracteres son los siguientes:

- nivel_superior
- nivel_medio_superior
- nivel_Medio_inferior
- nivel_inferior
- Inclinación,
- ancho_horizontal,
- ancho_vertical,
- ancho_left,
- ancho_right,
- ancho_vertice,
- ancho_detalle,
- altura_detalle,
- ancho_final_detalle,
- detalle_curvo,
- final_curvo,
- grosor

Por lo tanto, el segmento izquierdo de la letra debe tener un ancho, en sus dos extremos, igual al ancho_right, mientras que el segmento derecho debe tener un ancho igual a ancho_left; el ancho del vértice debe ser igual a ancho_vértice y el ancho del segmento horizontal igual a ancho_horizontal. De esta forma es que se definen los anchos de cada parte de la letra, variando estos en función de los parámetros.

A continuación se puede apreciar el código de definición de la letra A.

```
Letra A_romana(float Inclinación, float nivel_superior, float nivel_medio_superior, float
       nivel_medio_inferior, float nivel_inferior, float ancho_horizontal, float   ancho_vertical,
       float ancho_left, float ancho_right, float ancho_vertice, float ancho_detalle, float
       altura_detalle, float ancho_final_detalle, float detalle_curvo, float final_curvo, float
       grosor) {

   /* Defino los tipos necesarios para la descripción de la letra */
   Letra l = crear_Letra(); Curva c = crear_Curva(); Pendiente pen;
   Puerto puerto1, puerto2, puerto3, puerto4, puerto5;
   float x1, y1, x2, y2, x3, y3, x4, y4, xv, yv, xm, xv1, xv2;
   Ideograma id1, id2, id3, id4, id5;

   /* Determino los puntos de control para la letra */
   float x_left = margen_Romano(1, 'A')+ancho_detalle/2.0+ancho_right/2.0;
   float x_right = 1.0-margen_Romano(2, 'A')-ancho_detalle/2.0-ancho_left/2.0;

   x1 = x_left;       y1 = nivel_medio_inferior;
   x4 = x_right;      y4 = nivel_medio_inferior;
   xv = 0.5;          yv = nivel_superior;

   xv1 = xv-(ancho_vertice/2.0)+(ancho_right/2.0);
   xv2 = xv+(ancho_vertice/2.0)-(ancho_left/2.0);

   y2 = 0.5-ancho_horizontal/2.0;  x2 = x1 + (xv1 -x1)*((y2-y1)/(yv-y1));
   y3 = 0.5+ancho_horizontal/2.0;  x3 = xv2 +(x4-xv2)*((y3-yv)/(y4-yv));

   /* Inclino la letra, modificando las posiciones de los puntos de control*/
   x1 = x1 + inclinacion*ancho_inclinacion*(y1-((nivel_superior+nivel_medio_inferior)/2.0));
   x2 = x2 + inclinacion*ancho_inclinacion*(y2-((nivel_superior+nivel_medio_inferior)/2.0));
   x3 = x3 + inclinacion*ancho_inclinacion*(y3-((nivel_superior+nivel_medio_inferior)/2.0));
   x4 = x4 + inclinacion*ancho_inclinacion*(y4-((nivel_superior+nivel_medio_inferior)/2.0));
   xv = xv + inclinacion*ancho_inclinacion*(yv-((nivel_superior+nivel_medio_inferior)/2.0));

   xm =  (x2+x3)/2.0;

   /* Defino los puertos sobre los que se definirá cada ideograma */
   puerto1  = cons_Puerto(0.0, nivel_medio_inferior, xm, y2-y1);
   puerto2  = cons_Puerto(0.0, y2, xm, y3-y2);
   puerto3  = cons_Puerto(0.0, y3, 1.0, yv-y3);
   puerto4  = rotar_Puerto(2, cons_Puerto(xm, y2, 1.0-xm,ancho_horizontal));
   puerto5  = rotar_Puerto(2, cons_Puerto(xm, nivel_medio_inferior, 1.0-xm, y2-y4));

   /* Creo la pendiente inicial para el árbol_curva */
   pen = cons_Pendiente((x1-ancho_right/2)/xm, 0.0, (x1+ancho_right/2)/xm, 0.0,
                        (x1-ancho_right/2)/xm, 1.0, (x1+ancho_right/2)/xm, 1.0);

   /* Creo los ideogramas */
   id1 = Semiplanos_Detalles(1, 2, 1, x2/xm, ancho_right, ancho_detalle, alto_detalle,
         alto_final_detalle, grosor, final_curvo, 1, 1, 0, 0,  puerto1, 0, 0);

   id2 = Contiunacion_Recta_Curva(2, 1, 1, 1.0, 1.0, 0.5, ancho_horizontal, 0.5, ancho_horizontal,
         0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, puerto2, 1, 1);

   id3 = Continuacion_Vertice_Detalles(3, 1, alto_final_detalle, ancho_vertice, ancho_left, x3,
         ancho_left, ancho_detalle, alto_detalle, lto_final_detalle, detalle_curvo, grosor,
         final_curvo,0,0, puerto3, 2, 1);

   id4 = Contiunacion_Recta_Curva(4, 1, 1, 1.0, 1.0, 0.5, ancho_horizontal, 0.5, ancho_horizontal,
         0.0,0.0,0.0,0.0,1.0,1.0,1.0,1.0, puerto4, 3, 2);

   id5 = Semiplanos_Detalles(5, 2, 1, (1.0-x4)/(1.0-xm), ancho_left, ancho_detalle, alto_detalle,
         alto_final_detalle, grosor, final_curvo, 0, 0, 1, 1, puerto5, 4, 1);

/* Seteo la pendiente inicial y los ideogramas a la curva */
   id1 = set_Pendiente(id1, pen);
   c = Agregar_Ideograma_Curva(id1, c);
   c = Agregar_Ideograma_Curva(id2, c);
   c = Agregar_Ideograma_Curva(id3, c);
   c = Agregar_Ideograma_Curva(id4, c);
   c = Agregar_Ideograma_Curva(id5, c);

/* Agrego el árbol_curva a la letra y devuelvo la letra final*/
   l = agregar_Curva_Letra(c,  1, l);
   return l;
}
```

Letra ‘V’

La letra ‘V’ es tal vez la de más fácil definición, ya que está formada por un vértice y dos continuaciones de semiplanos, como se puede apreciar en la figura N° 5.13.

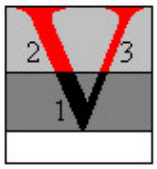


Figura N° 5.13 – Letra ‘V’

Con la definición del vértice, definimos las inclinaciones de cada recta y los anchos que tendrán tanto el vértice como las rectas en sus inicios y finales dentro del puerto. Cada una de estas salidas serán tomadas luego por dos intersecciones de semiplanos que las continuaran hasta el final de sus puertos, en los que se “dibujarán” los detalles a ambos lados.

Letra ‘E’

Para la letra E tenemos dos tipos de detalles: horizontales y paralelos. En la figura N° 5.14 podemos ver que la letra está compuesta por tres curvas con semiplanos (numeradas como 1, 3 y 5), las que están alineadas sobre sus salidas superiores Cada uno de estos ideogramas tiene una salida recta sobre el lado derecho de cada puerto, las cuales serán continuadas por tres detalles paralelos. La inclinación es definida por los puntos de control, y los detalles paralelos deberán adaptarse a las nuevas inclinaciones variando el largo de la continuación de sus entradas.

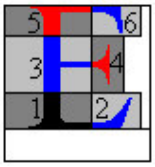


Figura N° 5.14 – Letra ‘E’

Letra ‘S’

La letra S debe ser tal vez la más difícil de definir, debido a que posee muchas curvas. En ella podemos apreciar que aparecen el detalle inicial y el detalle final, y que sus formas son tan curvas que no pueden adaptarse a otra figura, si no que deben ser definidos cada uno como un único ideograma. Como se muestra en la figura N° 5.15, la letra se descompone en un conjunto de ideogramas que conforman un único árbol_curva. Los ideogramas inicial y final de la curva son los detalles inicial y final, mientras que los otros tres son los encargados de controlar su forma.

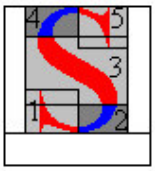


Figura N° 5.15- Letra ‘S’

La letra comienza con un detalle inicial y es continuada por tres ideogramas curvos, para finalmente terminar con la definición de un detalle final. Puede apreciarse que las continuaciones entre ideogramas son perpendiculares a los puertos.

Letra 'M'

Aunque la letra M pueda parecer complicada en un primer momento, su definición es bastante sencilla. La letra se compone de cinco ideogramas, tres de los cuales son vértices. El primer ideograma define su inclinación inicial y dibuja los detalles inferiores sobre su base; luego, su salida es tomada por un vértice que continua su salida hasta llegar al extremo superior del puerto, donde define su detalle superior izquierdo y una inclinación de salida para el otro segmento del vértice. Esta salida es continuada luego por otro vértice que a su vez define otra inclinación de salida para el otro segmento. Luego esta salida es tomada por el tercer vértice, que define el detalle derecho del vértice y la salida de su otro segmento de forma tal, que las dos "patas" de la letra queden paralelas, para finalmente terminar en un quinto ideograma, que en este caso es una intersección de semiplanos, el cual continúa la salida hasta llegar al otro extremo del puerto, en donde se desplegarán los detalles finales sobre sus dos lados.

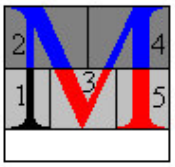


Figura N° 5.16 – Letra 'M'

Letra 'O'

La letra O se define de una forma bastante sencilla, ya que simplemente se utiliza un único ideograma que "dibuja" la diferencia entre dos elipses concéntricas, como se puede apreciar en la figura N° 5.17.

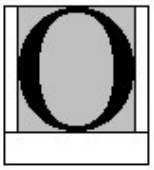


Figura N° 5.17 – Letra 'O'

Estas elipses concéntricas son definidas de forma tal que los bordes superior e inferior respeten el parámetro ancho_horizontal, mientas que los bordes derecho e izquierdo se adaptan al ancho_vertical.

Letra 'n'

En la figura N° 5.18 se puede apreciar como la letra 'n' comienza con una intersección de semiplanos (Ideograma 1) y es continuada por un ideograma (Ideograma 2) que define una salida sobre el lado superior del puerto y otra sobre su lado derecho; la primera es continuada por un detalle final (Ideograma 3), y la segunda por una curva (Ideograma 4). Finalmente la curva del ideograma 4 es continuada por una intersección de semiplanos (Ideograma 5) que define sus detalles finales sobre ambos lados.

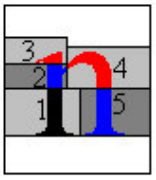
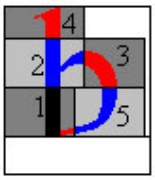


Figura N° 5.18 – Letra 'n'

Letra 'b'

La letra comienza con una intersección de semiplanos (ideograma 1 de la figura N° 5.19) que define una salida curva sobre su lado derecho. La salida superior es continuada por una intersección de semiplanos con salida curva (Ideograma 2), mientras que la salida derecha por una curva (Ideograma 5). La salida superior del ideograma 2 es tomada por un detalle final y la derecha por una curva (Ideograma 3). Las salidas de los ideogramas 3 y 4 tienen sus posiciones y anchos finales coincidentes, de forma tal de producir el efecto de continuidad, que no es posible implementar directamente, debido a que los árboles_curva no pueden contener ciclos.



Letra N° 5.19 – Letra 'b'

Letra 'j'

Esta letra tiene la particularidad de que es una de las minúsculas cuya definición llega hasta el nivel_inferior, sobrepasando el nivel_medio_inferior que representa el renglón. Sin embargo, su definición es sencilla y está divida en dos árboles_curva: uno con cuatro ideogramas y el otro con uno solo. Inicialmente se define su punta inferior (Ideograma 1), la cual está definida por una detalle inicial rotado 90°, cuya salida es tomada por una curva (Ideograma 2) que define su salida de acuerdo con la inclinación debida. Esta salida es tomada como entrada por un tercer ideograma (intersección de semiplanos), que simplemente la continúa. Luego, su salida es tomada por un detalle paralelo que la continua, dibujando un detalle sobre su lado izquierdo. Finalmente, la segunda curva_árbol se "dibuja" sobre la parte superior del puerto y define un aelipse que representa el punto de la j.

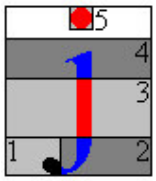


Figura N° 5.20 – Letra 'j'

5.1.4 Representación de una cadena de caracteres "String"

Para poder representar un conjunto de caracteres sobre una página, es que se definió una representación de Strings que justifica sus caracteres sobre el port en el que se deben desplegar.

La función que imprime un String en un lista de ports es la siguiente:

Port_List obtener_cadena(Port_list pl, DPT string).

Esta función toma como parámetros una lista de ports y un par de tipos dependientes (DPT); siendo la lista de ports las zonas de la página sobre las cuales se debe dibujar la cadena de caracteres determinada por el DPT. El DPT contiene la cadena de caracteres en su segundo parámetro, y de su primer argumento se obtienen los parámetros ingresados como una expresión de inicialización de tipo (TIE). Estos parámetros podrían definir, por ejemplo, si los caracteres deben dibujarse con subrayado y en ese caso el grosor del mismo, o también podrían definir, por ejemplo, el espaciado entre caracteres.

La representación de una cadena de caracteres no es tan sencilla como la división en ports de idéntico tamaño del área en que debe desplegarse el string, ya que cada carácter tiene su propio ancho, y si se utiliza este modelo en algunos casos la diferencia entre la distancia de dos caracteres puede ser muy grande y así obtener una distribución no deseada de los mismos. En la figura N° 5.21(a) se presenta un ejemplo de los caracteres 'w' y 'a' cuyos puertos se muestran adyacentes, mientras que en la N° 5.21(b) se puede ver la separación entre las letras i y t, con las mismas características que la anterior. Finalmente, en la figura N° 5.21(c) se puede ver el string "wait" formado por las cuatro letras recién nombradas, donde se aprecia el efecto que tiene la utilización de un espaciado entre caracteres que tienen ports de igual tamaño.

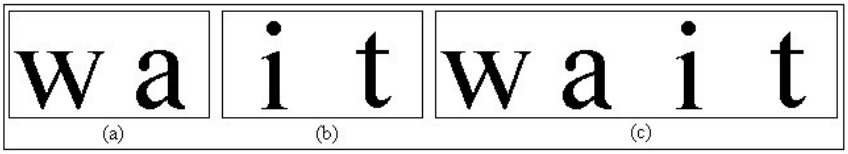


Figura N° 5.21 Distancia entre caracteres

Se hace evidente, luego de observar la figura anterior, que si los puertos de los caracteres son del mismo tamaño, la separación entre caracteres debe contemplar el ancho de cada una de las letras. Como los ports de los caracteres deben tener las mismas dimensiones, ya que los parámetros generales, como por ejemplo el ancho, son definidos en relación a las dimensiones del port, es que el diseñador de fuentes debe proveer una función que dada un letra devuelva los márgenes que se deben utilizar en su representación para el string.

Estos márgenes determinan cuanto espacio debe reducirse de cada lado de un port para poder ubicar correctamente los caracteres. De esta forma, los puertos dejan de ser adyacentes y pasan a interceptarse. En la figura N° 5.22 (a) se muestra en azul el margen establecido para cada letra y en la siguiente figura (b) se muestra como queda el texto final luego de eliminar los márgenes de cada puerto. En estas figuras no se utiliza una separación entre caracteres.

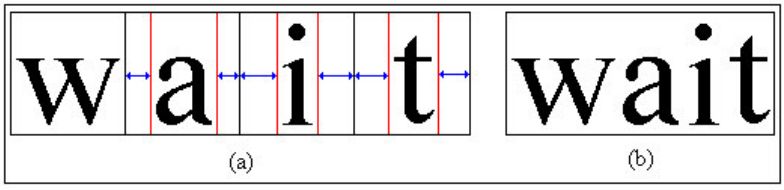


Figura N° 5.22 – Márgenes

Como ya se mencionó, el espaciado entre caracteres se obtiene de los parámetros ingresados en la TIE del tipo de pares dependientes *String*.

La forma en que se ingresan los parámetros en la expresión de inicialización de tipo para un string es la siguiente:

Primero se define el conjunto de parámetros que determina la fuente y el tipo para la misma y luego los parámetros que corresponden al string,

```
#define TIE_caracter {fuente, inclinación, ancho_horizontal, ancho_vertical,
             ancho_right, ancho_left, ancho_vértice, nivel_superior,
             nivel_medio_superior, nivel_medio_inferior, nivel_inferior,
             ancho_detalle, alto_detalle, alto_final_detalle, detalle_curvo,
             final_curvo, grosor};

#define TIE_string {espacio};
```

Finalmente se crea el tipo del string con las características deseadas,

```
DT_typedef char TIE_caracter String[Max_Char] TIE_string;
```

Como ya se mencionó, el diseñador de fonts debe proveer una nueva función de la cual obtener el margen de cada letra. Esta función debe recibir un caracter y devolver el margen que se debe utilizar. Luego, esta función será utilizada por el generador de strings, el cual utiliza el motor de fonts para generar cada letra en su port correspondiente.

Para obtener los ports en las cuales se definirán los elementos del string, también se debe calcular la ubicación de cada port teniendo en cuenta la dimensión del espaciado que debe utilizarse entre caracteres. Además, no todos los márgenes deben eliminarse, ya que el primer margen izquierdo y el último derecho deben ser representados, pues que de otra forma los ports correspondientes estarían fuera del port original.

Es así que el ancho del puerto del string es igual a la suma de los anchos de los puertos de sus caracteres, menos los márgenes que deben eliminarse, y menos el espaciado definido para ellos, por lo que llegamos a la siguiente ecuación con la cual pretendemos obtener el tamaño de puerto de caracter en función del port original:

$$1 = \sum_{i=1}^{n} \text{ancho\_puerto} + \sum_{i=1}^{n} (\text{ancho\_puerto} \cdot \text{espacio}) + \left[\sum_{i=1}^{n} -[\text{margen\_izq}(s[i]) - \text{margen\_der}(s[i])] + \text{margen\_izq}(s[i]) + \text{margen\_der}(s[i]) \right] \text{ancho\_puerto}$$

Haciendo cuentas llegamos a,

$$\text{ancho\_puerto} = \frac{1}{n + (n-1)(\text{espacio}) - \sum_{i=1}^{n} [\text{margen\_izq}(s[i]) + \text{margen\_der}(s[i])] + \text{margen\_izq}(s[i]) + \text{margen\_der}(s[i])}$$

Finalmente, la posición inicial es igual al margen izquierdo del primer caracter. Cada caracter debe eliminar su margen izquierdo en el momento de ser desplegado, para poder implementar el espaciado requerido, por lo cual el port del primer carácter queda ubicado en el lugar correcto.

La regla para calcular cada posición es la siguiente:

Posición(i) = posición(i-1) +ancho_puerto -margen_izquierdo(i-1) -margen_derecho(i-1) +espacio

El algoritmo que obtiene una nueva lista de ports, a partir de la lista de ports en las que debe definirse el string, es el siguiente:

```
obtener_Puertos_String(Port_List pl, String st, float espacio) : Port_List
{
   Port_List string = opm_nil();
   float margenes = 0.0;
   float margen_izquierdo, float margen_derecho, float posición, ancho_puerto;
   int size, i;
   String s;
   size = tamaño(st);

   Para cada caracter (c) del string (st){
     margenes = margenes + margen(izquierdo, c) + margen(derecho, c);
   }

   margenes = margenes - margen(izquierdo, primer_caracter(st)
                       - margen(derecho, último_caracter(st);

   ancho_puerto = 1.0/(size - margenes + (size-1.0)*espacio);
   espacio = ancho_puerto*espacio;
   posicion = margen(izquierdo, primer_caracter(st))*ancho_puerto;

   Para cada carácter (c) del string (st) {
     margen_izquierdo = margen(izquierdo, c)*ancho_puerto;
     margen_derecho = margen(derecho, c)*ancho_puerto;

     if (c <> ' ') {
        string = opm_cat(string, obtener_Puertos_Letra(-3, DT_pair(Romanas,c),
             opm_scale(1.0-posicion+margen_izquierdo, posicion+ancho_puerto-
             margen_izquierdo, 1.0, 1.0, pl)));
     }
     posicion= posición +ancho_puerto -margen_izquierdo -margen_derecho +espacio;
   }

     retornar string;
}
```

Este algoritmo es una simplificación de la función que obtiene los puertos para un string, ya que en realidad debe obtener los parámetros de un DPT y luego utilizar la fuente definida.

5.2 Evaluación de los resultados

Al comienzo de este capítulo nos habíamos planteado una serie de preguntas que intentaremos responder para evaluar si se han cumplido los objetivos planteados en un primer momento.

La primera pregunta pretendía evaluar si es posible representar distintas fuentes de letras sin que esto significara un gran esfuerzo por parte del diseñador de fonts. Luego de observar las definiciones de la fuente de letras romana pudimos comprobar no sólo que pueden generarse las letras sin mucho esfuerzo, sino que además el código generado es compacto y fácil de entender. Además la selección de ideogramas es los suficientemente genérica como para poder definir nuevos símbolos más complejos.

Luego nos preguntábamos si podría lograrse una parametrización flexible y fácil de implementar, lo cual pudimos comprobar luego de definir la fuente de letras romana. Sin embargo, los parámetros escogidos fueron útiles para la fuente en cuestión, pero otra nueva fuente podría requerir otro tipo de parametrización para la cual debiera definirse un nuevo conjunto de ideogramas. De todas formas, la arquitectura general de la solución posibilita que sea agreguen nuevas primitivas, ideogramas y fuentes de letras sin mucho dificultad.

Otro de los objetivos planteados era el de minimizar el conocimiento matemático indispensable que debe poseer el diseñador para realizar sus definiciones. En este punto se puede decir que el avance ha sido muy bueno, ya que las primitivas matemáticas, como las curvas de Bézier, están encapsuladas en las librerías inferiores y el diseñador sólo debe entender el concepto de continuidad y poder descomponer la letra en los ideogramas que conforman los árboles_curva. Sin embargo, deben realizarse algunas operaciones para determinar las posiciones de los puntos de control y la forma en que cambian las dimensiones de cada parte de la letra respecto a la inclinación, pero que de todas formas no representan demasiada complejidad.

También nos preguntábamos qué nivel de resolución tendrían las representaciones de las fonts y además que calidad tienen las letras finales. Para responder esta pregunta, primero debemos ver la salida generada. En las figuras N° 5.23, N° 5.24 y N° 5.25 podemos ver la representación como String de las letras seleccionadas para los tres tipos de resolución más comunes, con una selección arbitraria de los restantes parámetros.

Figura N° 5.23 – Salida del subconjunto de la fuente romana con un valor de 1 para rec_nr

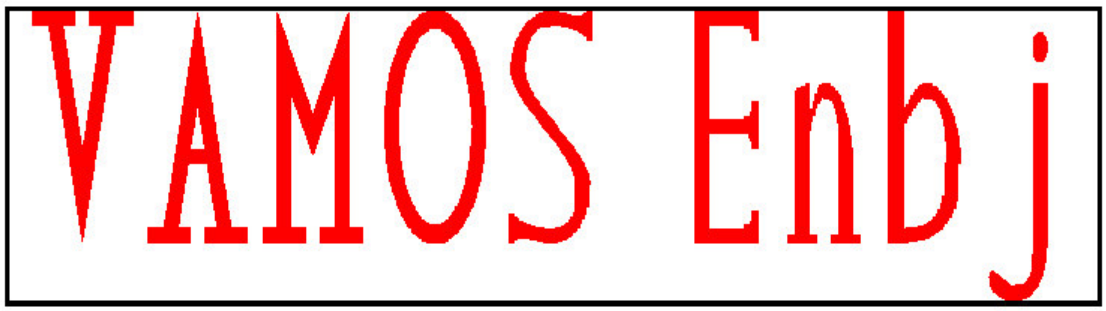


Figura N° 5.24 – Salida del subconjunto de la fuente romana con un valor de 2 para rec_nr

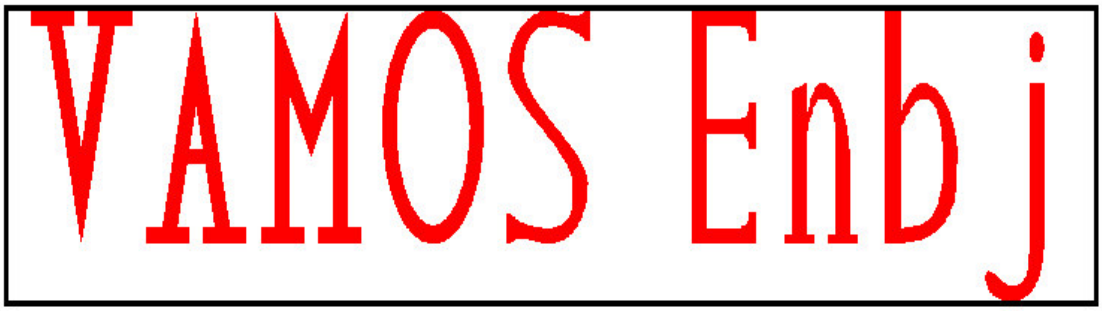


Figura N° 5.25 – Salida del subconjunto de la fuente romana con un valor de 3 para rec_nr

La primera impresión que se tiene al ver las letras finales, es que no pueden apreciarse los límites entre ideogramas, por lo cual el concepto de continuidad pudo ser representado con fidelidad. Además la resolución final se mantiene uniforme no sólo en cada letra en particular, si no también en todo el String.

En las figuras anteriores se podía ver el String para el subconjunto de letras seleccionado, pero sin inclinación; veamos ahora cual es el resultado cuando se agregan valores distintos de 0 para el parámetro inclinación. En las figuras N° 5.26 a N° 5.29 se puede apreciar que el cambio en la inclinación es uniforme para todas las letras.

También se pretendía poder diferenciar cuatro tipos de letras para la fuente romana (Roman, Arial, Courier y Ghotic), para lo cual definimos el siguiente conjunto de parámetros generales: nivel_superior = 1.0, nivel_medio_superior = 0.75, nivel_medio_inferior = 0.2 y nivel_inferior = 0.0.

Y para cada uno de los tipos de letras los siguientes valores en cada uno de sus parámetros:

- Roman
  - ancho_horizontal = 0.05
  - ancho_vertical = 0.1
  - ancho_left = 0.1
  - ancho_right = 0.05
  - ancho_vértice = 0.025
  - ancho_detalle = 0.1
  - altura_detalle = 0.1
  - ancho_final_detalle = 0.05
  - detalle_curvo = 1
  - final_curvo = 0
  - grosor = 0.5
- Arial
  - ancho_horizontal = 0.1
  - ancho_vertical = 0.1
  - ancho_left = 0.1
  - ancho_right = 0.1
  - ancho_vértice = 0.12
  - ancho_detalle = 0.0
  - altura_detalle = 0.0
  - ancho_final_detalle = 0.0
  - detalle_curvo = 0
  - final_curvo = 0
  - grosor = 0.0
- Courier
  - ancho_horizontal = 0.05
  - ancho_vertical = 0.05
  - ancho_left = 0.05
  - ancho_right = 0.05
  - ancho_vértice = 0.12
  - ancho_detalle = 0.15
  - altura_detalle = 0.05
  - ancho_final_detalle = 0.05
  - detalle_curvo = 1
  - final_curvo = 0
  - grosor = 0.5
- Ghotic
  - ancho_horizontal = 0.15
  - ancho_vertical = 0.15
  - ancho_left = 0.15
  - ancho_right = 0.15
  - ancho_vértice = 0.2
  - ancho_detalle = 0.05
  - altura_detalle = 0.05
  - ancho_final_detalle = 0.02
  - detalle_curvo = 1
  - final_curvo = 0
  - grosor = 0.5

Para poder representar estos tipos de letras en un string se debe, además, asignar un valor de inclinación. En las figuras N° 5.26 a N° 5.29 se pueden apreciar los resultados para los tipos de letra definidos con y sin inclinación, utilizando para estas últimas las inclinaciones extremas en cada sentido.

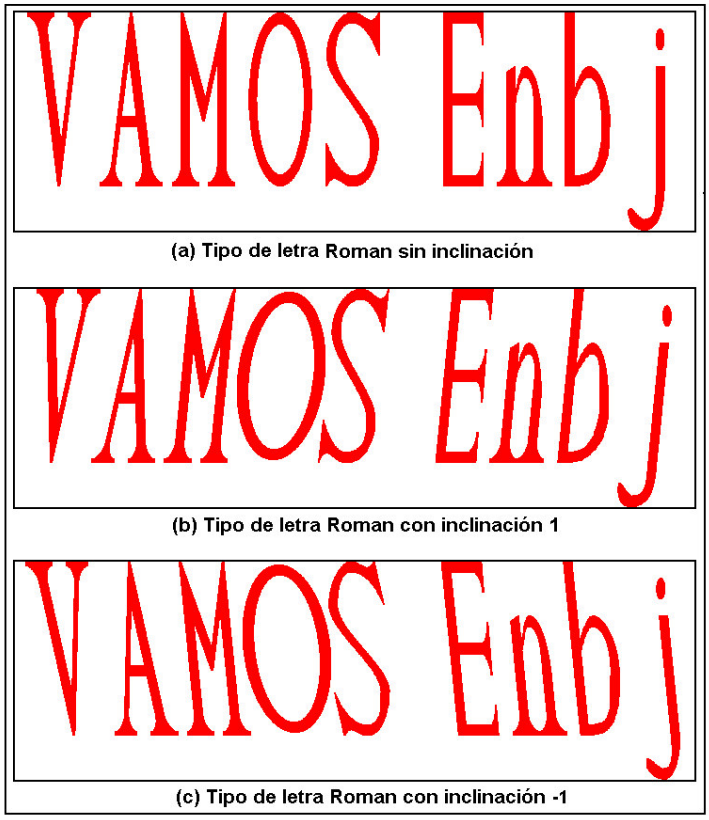


Figura N° 5.26 Tipo de letra Roman para la fuente romana

Figura N° 5.27 – Tipo de letra Arial para la fuente romana

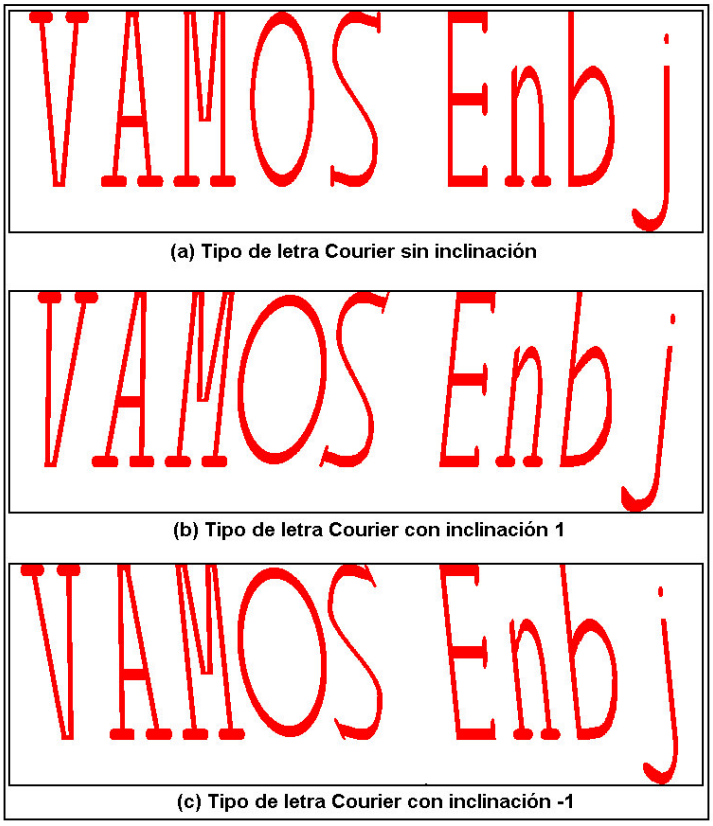


Figura N° 5.28 - Tipo de letra Courier para la fuente romana

Figura N° 5.29 - Tipo de letra Gothic para la fuente romana

En las anteriores figuras se puede comprobar que efectivamente los valores para los parámetros definen los tipos de letra acorde con la idea original, y que las formas representativas de cada tipo se mantienen a pesar de las inclinaciones.

Finalmente, pretendíamos establecer qué tan costosa era la generación de las letras, haciendo especial hincapié en la cantidad de memoria utilizada y el tamaño, en bytes, de la letra final. Además resulta importante determinar si el tiempo de ejecución no es prohibitivo, y permite concluir la viabilidad práctica de la solución. Con la finalidad de realizar estas averiguaciones es que dividimos la página en 80 puertos, que conforman cuatro columnas y 20 filas con una pequeña separación entre ellos y desplegando en cada uno el string “VAMOS VAMOS” que contiene el tipo romano para la fuente definida. En la figura N° 5.30 se puede observar la página obtenida.

Figura N° 5.30 – Representación sobre 80 ports del string “VAMOS VAMOS”

Para realizar la evaluación se utilizó una computadora Pentium III de 733 MHz y 192 MB de memoria RAM. Los datos obtenidos en la generación de esta página arrojan los siguientes resultados:

Máxima memoria utilizada: 64 MB
Tamaño del archivo final: 7.2 MB
Tiempo de ejecución: 165 s

Estos datos ponen de manifiesto la viabilidad práctica de la solución, puesto que en una computadora estándar se obtienen tiempos de respuesta razonables para la generación de una página de letras, que en este caso contenía 800 caracteres. Además, la memoria utilizada en esta prueba no fue excesiva y el archivo final tiene un tamaño medianamente manejable para la cantidad de letras utilizadas.

Para tener una cantidad de pruebas mayor que permitan respaldar los valores generados en el ejemplo anterior, realizamos una representación similar para 600, 400, 200 y 100 caracteres sobre la página, para los tres tipos de resolución automática posibles. A continuación se muestran los resultados obtenidos que reafirman la conclusión anterior.

		800 caracteres	***600 caracteres***	***400 caracteres***	***200 caracteres***	***100 caracteres***
Resolución: 1	***Máxima memoria utilizada***	64 MB	54 MB	48 MB	27 MB	15 MB
	Tamaño del archivo final	7.2 MB	6.2 MB	5.1 MB	4.5 MB	2 MB
	Tiempo de ejecución	165 s	95 s	50 s	15 s	8 s
Resolución: 2	***Máxima memoria utilizada***	42 MB	35 MB	27 MB	10.4 MB	6.5 MB
	Tamaño del archivo final	7.1 MB	6.2 MB	4.2 MB	2.2 MB	1.6 MB
	Tiempo de ejecución	25 s	18 s	13 s	8 s	6s
Resolución: 3	***Máxima memoria utilizada***	24 MB	20 MB	16.1 MB	8 MB	5.5 MB
	Tamaño del archivo final	3.3 MB	2.9 MB	1.5 MB	1.2 MB	0.9 MB
	Tiempo de ejecución	10 s	8 s	7 s	4 s	2 s

En el anexo [Instalación del Prototipo y ejemplo de prueba] se puede ver como instalar y utilizar el compilador de C5, el Prototipo construido y algunos ejemplos de prueba.

Capítulo Nº 6
Conclusiones

El objetivo principal de este trabajo era el de investigar el dominio de la tipografía digital y diseñar una solución de generador de fuentes de letras para C5 que superara los problemas para el manejo de curvas y la complejidad en la definición de letras del actual generador del lenguaje. La conclusión general a la que se llegó es que se han cumplido los objetivos planteados en un comienzo, superando incluso las expectativas en cuanto a la facilidad del diseño de fonts y manteniendo las principales ventajas del generador actual, como son la parametrización de las letras y la compilación de estas al mismo tiempo que la página en que se presentan.

En cada una de las siguientes secciones se presentan las conclusiones particulares del Proyecto.

6.1 Sobre la tipografía digital

La tipografía digital ha brindado nuevas posibilidades en el diseño de letras, dado que antes de su aparición la utilización de distintos símbolos se realizaba por medio de herramientas mecánicas cuya utilización en muchos casos era sumamente compleja, alcanzando su punto de máxima dificultad en la representación de fórmulas matemáticas. Utilizando descripciones matemáticas y una impresora que utiliza pequeños puntos de tinta, puede aproximarse un símbolo sobre una matriz mediante líneas de rastreo que depositan correctamente cada gota sobre la posición que deben ocupar mediante pulsos eléctricos.

Existen muchos generadores de fonts, y cada uno tiene sus ventajas y desventajas; dependiendo de cual sea el objetivo de una representación es que será preferible utilizar uno u otro. Para los procesadores de texto en los cuales lo importante es poder desplegar una gran cantidad de texto, tal vez la mejor opción sea utilizar TrueType, ya que con una elección adecuada entre los tipos disponibles de su librería será suficiente. Si se está pensando en la utilización de símbolos complejos, como pueden ser formulas matemáticas, podría preferirse la utilización de Metafont, dado que tiene un gran dominio de definición para formas complejas. Mientras que si se piensa en combinar texto con representaciones gráficas, la mejor opción es utilizar algún generador de fuentes que permita la utilización de las letras como cualquier otro elemento gráfico y este es el caso de PostScript o del actual generador de C5. La ventaja que ofrece el generador de fonts diseñado en este proyecto, es que no es necesario compilar cada tipo de letra para que luego puedan ser utilizados, si no que cada tipo es generado en el momento de compilar la página, con lo cual el usuario puede decidir las características del tipo que desea en el mismo momento en que crea la página.

6.2 Sobre la solución propuesta

La solución propuesta presenta la gran ventaja de proveer un fácil diseño de fonts con un gran nivel de abstracción. Sin embargo, como todo modelo constructivo, el dominio de la definición de fonts se ha restringido al de la utilización de los componentes disponibles. De todas formas, la visión de la composición de letras como una secuencia de traducciones desde un lenguaje de alto nivel y bastante intuitivo (nivel de usuario) hasta un nivel de representación (primitivas gráficas de C5), permite que para la definición de fonts se incluyan nuevos componentes (ideogramas) para la definición de los cuales el diseñador sí debe tener un conocimiento matemático y del sistema. Esta restricción en el dominio, entonces, está relacionada con los componentes disponibles y el conocimiento sobre el sistema y el manejo de conceptos matemáticos que posea el diseñador.

El otro gran aporte que brinda la solución, es el de posibilitar la creación de fuentes de letras propiamente dichas, puesto que el resto de generadores de fuentes del ambiente del diseño de fonts (a menos del generador actual de C5), simplemente definen tipos de letras y el usuario sólo puede seleccionar el tipo que prefiere de un conjunto preestablecido.

La parametrización de fonts para brindarle al usuario una real fuente de letras y que éste pueda seleccionar un tipo entre ellas, ya estaba contenida en el actual generador de fuentes de C5. Sin embargo, las mejoras de este nuevo generador respecto al anterior están en el mejor manejo de curvas y en la posibilidad de definir letras minúsculas, además de una definición más compacta y natural.

La solución, además, tiene en cuenta las principales ventajas de cada uno de los generadores estudiados. En comparación con MetaFont, se utilizan curvas de Bézier que permiten un fácil manejo de curvas y se utilizan parámetros, como las rectitudes, para controlar las formas de la letra. Al igual que PostScript, aunque es más una característica de C5 que del generador, las letras obtenidas continúan siendo objetos gráficos del lenguaje, por lo que pueden aplicarse operaciones sobre la página, rotándolos, escalándolos o utilizando cualquier otra operación. De forma similar a la que en TrueType se realiza el diseño de fonts, describiendo los contornos internos y externos de la letra, también con el nuevo generador se va describiendo el contorno, pero de a trozos, permitiendo al igual que con TrueType, la fácil identificación de las partes internas y externas de las letras para poder aplicar patrones sobre estas.

Más allá de las ventajas que ofrece en el diseño de fonts, también podemos concluir que la solución es viable en la práctica, ya que los costos en tiempo de ejecución de su implementación están dentro de un rango aceptable. Además, el tamaño (en bytes) del archivo final es muy bueno y la memoria que utiliza es ampliamente inferior a la que posee un computador estándar.

Finalmente, un aspecto muy importante que cubre el generador de letras construido es que la compilación de los tipos de letra se realiza al mismo tiempo que la creación de la página final, mientras que para el resto de los generadores (menos el generador actual de C5) la compilación debe ser realizada antes de su utilización por los lenguajes de representación gráfica a través de librerías.

6.3 Aportes del proyecto

El principal aporte del proyecto es el de proveer una solución puramente constructiva, que constituye un nuevo enfoque en la definición de fonts, tomando de los generadores más representativos del ambiente sus principales ventajas.

Esta solución, además, brinda una puerta de salida para el problema de la generación de fonts en C5, permitiendo la utilización de símbolos que hasta el momento no podían ser representados.

6.4 Trabajo futuro

Como trabajo a futuro existen algunos aspectos de la solución que pueden ser mejorados, entre los cuales mencionamos los siguientes:

- Ampliación del conjunto de componentes constructivos.
- Ampliación del conjunto de primitivas gráficas.
- Implementación de la rotación automática de puertos.

Además, se pueden estudiar aspectos de diseño e implementación que puedan ser mejorados para disminuir los tiempos de respuesta, la utilización de memoria o el tamaño final de los resultados.

Finalmente, el trabajo a futuro más importante que podría realizarse es el de completar el Prototipo e incorporarle las mejoras mencionadas en los puntos anteriores, con el objetivo de reemplazar el generador actual de fonts de C5. Para que la solución pueda ser utilizada, también debería extenderse el subconjunto de caracteres definido hasta completar la totalidad de símbolos del teclado.

Bibliografía

[JJC02] *"The C5 programming Language"*, Juan José Cabezas
Reporte Técnico INCO 02-01, Universidad de la República, Uruguay (2002)

[ICG] *"Introduction to Computer Graphics"*, James D. Floey, Andries Van Dam, Steven K. Feiner, John F. Hughes y Richard L. Phillips
Addison Wesley Publishing Company (1996)

[TTF] *True Type Reference Manual*
http://fonts.apple.com/TTRefMan/index.html

[GF99] *"GenFont, Un generador de fonts para Ventanas Tipadas",* Daniel López y Fabricio Pirez
Proyecto de Grado, INCO, Universidad de la República, Uruguay (1999)

[D&D94] *"C How to Programm 2/Ed"*, Harvey M. Deitel y Paul J. Deitel
Prentice Hall Inc. (1994)

[TIC00] *"Thinking in C++, 2nd ed. Volume 1"*, Bruce Eckel
Â© 2000 MindView, Inc. All rights reserved.

[JTG69] *"Lo aparente y lo concreto en el arte"* (1947), Jaquín Torres García
Capítulo 41 – Centro Editor de América Latína, 1969

[GPL92] *"Historia de la Pintura Uruguaya. Torres García de Barcelona a París"*, Gabriel Paluffo Linari
Ediciones de la Banda Oriental, 1992

[PSRM90] *"POSTSCRIPT Language Reference Manual, 2nd ed"*, Adobe Systems Incorporated
Addison – Wesley Publishing Company, 1990

[TMFB] *"The METAFONTbook"*, Donald E. Knuth
ISBN 0-201-13444-6

[DTDKn] *"Digital Typography"*, Donald E. Knuth
ISBN 1-57586-010-4

Anexos

A1 Curvas cúbicas paramétricas

Las curvas y superficies se pueden aproximar por trozos de primer grado que conforman polilíneas y polígonos respectivamente. Si las curvas o superficies no son lineales por trozos, hay que crear y almacenar una gran cantidad de coordenadas de puntos extremos para lograr una precisión razonable. Esta manipulación interactiva de los datos para aproximar una curva es algo tediosa, puesto que hay que ubicar con precisión una gran cantidad de puntos.

Tratando de resolver este problema es que se empezaron a utilizar representaciones más compactas y manejables de curvas suaves por trozos. En general se utilizan funciones de un grado mayor que el de las funciones lineales. Estas funciones sólo aproximan la forma deseada, pero requieren menos espacio de almacenamiento y ofrecen mayor facilidad de manipulación interactiva que las funciones lineales.

Podría elegirse una representación con funciones explícitas de x, de la forma: y=f(x) y z=g(x). Sin embargo, este método presenta varios problemas, entre ellos: 1) sólo admiten representaciones como funciones de x, de forma tal que curvas como los círculos y elipses precisan varios segmentos de curva para ser representadas; 2) esta definición no es rotacionalmente invariante, por lo que para rotar una curva definida de esta forma se requiere mucho esfuerzo.

También podrían utilizarse representaciones como soluciones de ecuaciones implícitas de la forma f(x,y,z)=0. Entre sus principales defectos se puede ver que la ecuación puede tener más soluciones de las que deseamos y que si se unen dos segmentos de curva puede ser difícil determinar si sus direcciones tangentes concuerdan en el punto de unión.

Una mejor opción resulta ser una representación paramétrica de las curvas: x=x(t), y=y(t), z=z(t). Con este método se superan los problemas ocasionados por las formas funcionales e implícitas y además se ofrece un conjunto de atractivos que resultan ampliamente favorables para la aproximación de curvas. Por ejemplo, las curvas paramétricas reemplazan la utilización de pendientes geométricas con vectores tangente paraméticos, evitando el problema de las pendientes infinitas.

Con este nuevo concepto de curva paramétrica, cada segmento de la curva global Q está definido por tres funciones x(t), y(t) y z(t), que son polinomios cúbicos de parámetro t. En la figura N° A1.1 se puede apreciar la curva Q formada por tres segmentos de curva; en este caso los vectores tangentes a P_2 en Q_1 y Q_2 tienen la misma dirección, de forma tal que se mantenga la continuidad G^1 en P_2. Lo mismo sucede para P_3 con las curvas Q_2 y Q_3

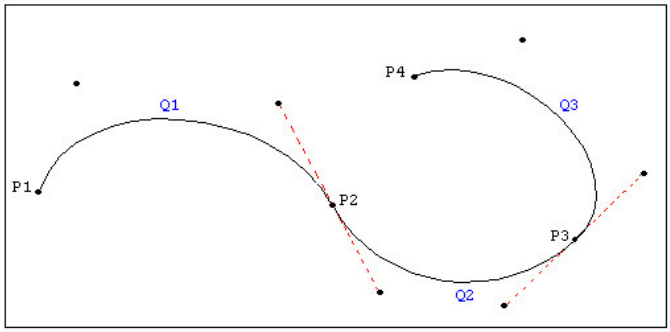


Figura N° A1.1 - Composición de curvas cúbicas paramétricas

En general se utilizan aproximaciones a cada segmento por medio de polinomios cúbicos, ya que los polinomios de grado menor no ofrecen mucha flexibilidad para controlar la forma de la curva y los de grado mayor pueden introducir ondulaciones indeseadas, además de requerir más cálculos. Asimismo, los polinomios de menor grado no permiten que un segmento de curva interpole dos puntos extremos y las derivadas en estos.

Características básicas

Los polinomios cúbicos que definen un segmento de curva $Q(t) = [x(t), y(t), z(t)]^T$ tienen la siguiente forma:

$$\begin{aligned} x(t) &= a_x t^3 + b_x t^2 + c_x t + d_x, \\ x(t) &= a_y t^3 + b_y t^2 + c_y t + d_y, \\ x(t) &= a_z t^3 + b_z t^2 + c_z t + d_z, \qquad 0 < t < 1 \end{aligned} \tag{A1.1}$$

El parámetro t se limita al intervalo cerrado [0, 1] para poder utilizar segmentos finitos de curva.

Siendo $T = [t^3 \; t^2 \; t \; 1]^T$ y la matriz de coeficientes de los tres polinomios

$$C = \begin{bmatrix} a_x & b_x & c_x & d_x \\ a_y & b_y & c_y & d_y \\ a_z & b_z & c_z & d_z \end{bmatrix}, \tag{A1.2}$$

rescribimos la ecuación A1.1 como:

$$Q(t) = [x(t) \;\; y(t) \;\; z(t)]^T = C.T. \tag{A1.3}$$

Cada polinomio cúbico de la ecuación A1.3 tiene cuatro coeficientes, por lo que se requieren cuatro restricciones para poder resolverla. Para ver la forma en que los coeficientes de la ecuación A1.1 dependen de las cuatro restricciones, rescribimos la matriz de coeficientes de la ecuación A1.3 como C = G.M, donde M es una matriz base 4x4 y G es una matriz de cuatro elementos de restricciones geométricas, llamada matriz geométrica. Las restricciones geométricas son las condiciones de la curva. La ecuación final es:

$$Q(t) = \begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix} = [G1 \; G2 \; G3 \; G4] \begin{bmatrix} m11 & m21 & m31 & m41 \\ m12 & m22 & m32 & m42 \\ m13 & m23 & m33 & m43 \\ m14 & m24 & m34 & m44 \end{bmatrix} \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix}$$

Los tres principales tipos de curva son: las de ***Hermite***, definidas por dos puntos extremos y los vectores tangentes a estos; las de ***Bézier***, definidas por dos puntos extremos y dos puntos que controlan los vectores tangentes a los dos puntos extremos; y las tipo ***Splines***, que tienen continuidad C1 y C2 en los puntos de unión, pero no interpolan los puntos extremos

A continuación se analizan las curvas de Hermite y de Bézier, pero no así las de tipo Spline; esto se debe a que necesitamos definiciones de curvas que faciliten su manipulación y que además interpolen los puntos extremos de control.

Curvas de Hermite

La forma de Hermite del segmento de curva polinomial cúbica está determinada por las restricciones de los puntos extremos P1 y P4 y los vectores tangente R1 y R4 en los puntos extremos.

Para hallar la matriz base hermitiana Mh que relaciona el vector geométrico hermitiano Gh con los coeficientes de los polinomios se utilizan cuatro ecuaciones, una para cada restricción en los cuatro coeficientes polinomiales desconocidos, con lo cual se resuelven las incógnitas.

Definiendo Gh_x, el componente x de la matriz hermitiana, como

$$Gh_x = [P1_x, P4_x, R1_x, R4_x], \quad \text{(A1.3)}$$

y rescribiendo x(t) como

$$X(t) = a_x t^3 + b_x t^2 + c_x t + d_x = C_x.T = G_{Hx}.M_H.T = G_{Hx\,.\,MH}\,.\,[t^3\ \ t^2\ \ t\ \ 1]^T, \quad \text{(A1.4)}$$

se hallan los valores de x(0) y x(1) en la ecuación (A1.4):

$$x(0) = P_{1x} = G_{Hx}\,.\,M_{Hx}\,.\,[0\ 0\ 0\ 1]^T, \quad \text{(A1.5)}$$

$$x(1) = P_{4x} = G_{Hx}\,.\,M_{Hx}\,.\,[1\ 1\ 1\ 1]^T. \quad \text{(A1.6)}$$

De la misma forma que hallamos x(0) y x(1), diferenciamos la ecuación (A1.4) para obtener $x'(t) = G_{Hx}.M_H[3t^2\ \ 2t\ \ 1\ \ 0]^T$, y finalmente obtener los valores para x´(0) y x´(1):

$$X'(0) = R_{1x} = G_{Hx}\,.\,M_H\,[0\ 0\ 1\ 0]^T, \quad \text{(A1.7)}$$

$$X'(1) = R_{4x} = G_{Hx}\,.\,M_H\,[3\ 2\ 1\ 0]^T. \quad \text{(A1.8)}$$

Reescribimos las ecuaciones (A1.5) – (A1.8) de forma matricial:

$$[P_{1X}, P_{4X}, R_{1X}, R_{4X}] = G_{HX} = G_{HX}\,.\,M_H\,.\begin{bmatrix} 0 & 1 & 0 & 3 \\ 0 & 1 & 0 & 2 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}. \quad \text{(A1.9)}$$

Para satisfacer también las expresiones correspondientes a z e y, M_H debe ser la inversa de la matriz de la ecuación (A1.9):

$$M_H = \begin{bmatrix} 0 & 1 & 0 & 3 \\ 0 & 1 & 0 & 2 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix}^{-1} = \begin{bmatrix} 2 & -3 & 0 & 1 \\ -2 & 3 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 1 & -1 & 0 & 0 \end{bmatrix}. \quad \text{(A1.10)}$$

Ahora podemos hallar x(t), al introducir M_H en $x(t) = G_{HX}\,.\,M_H\,.\,T$. De la misma forma hallamos y(t) y z(t), quedando en su forma matricial como

$$Q(t) = [x(t)\ y(t)\ z(t)]^T = G_H\,.\,M_H\,.\,T, \quad \text{(A1.11)}$$

Donde G_H es la matriz $[P_1\ P_4\ R_1\ R_4]$.

El producto $M_H.T$ define las funciones de mezcla de Hermite B_H como los polinomios que ponderan cada elemento de la matriz de geometría:

$$Q(t) = G_H.M_H.T = G_H.B_H = (2t^3 - 3t^2 + 1)P_1 + (-2t^3 + 3t^2)P_4 + (t^3 - 2t^2 + t)R_1 + (t^3 - t^2)R_4. \quad \text{(A1.12)}$$

En la figura N° A1.2 se pueden apreciar las cuatro funciones de mezcla. Observe que la suma siempre es igual a 1.

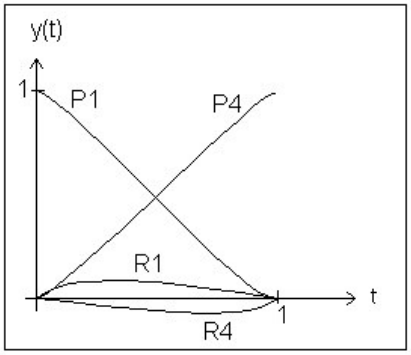


Figura N° A1.2

En la figura N° A1.3 se puede apreciar una serie de curvas hermitianas, siendo la única diferencia entre ellas la longitud del vector tangente R1 y manteniéndose su dirección.

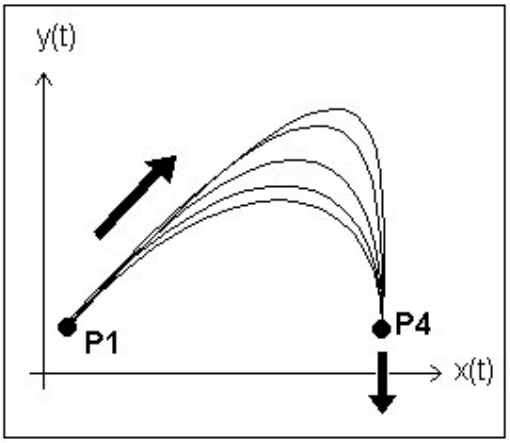


Figura N° A1.3 – Familia de curvas cúbicas hermitianas

Curvas de Bézier

La forma de Bezier del segmento de curva polinomial cúbica especifica de manera indirecta los vectores tangentes en los puntos extremos a través de la definición de dos puntos que no pertenecen a la curva (figura N° A1.4).

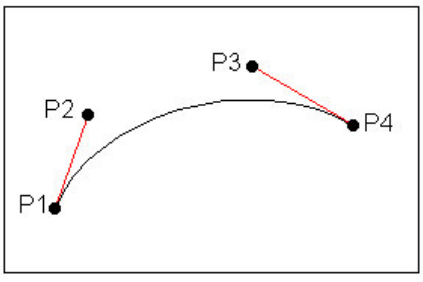


Figura N° A1.4 – Curva de Bézier [P_1, P_2, P_3, P_4]

Los vectores tangente inicial y final se determinan por medio de los vectores P_1P_2 y P_3P_4. Los vectores P_1P_2 y P_3P_4 se relacionan con los vectores R_1 y R_4 de la curva de Hermite mediante la siguiente ecuación:

$R_1 = Q'(0) = 3(P_2 - P_1)$, $R_4 = Q'(1) = 3(P_4 - P_3)$.

La curva de Bézier interpola los dos puntos de control extremos y aproxima los otros dos. La matriz de geometría de Bézier G_B, compuesta de los cuatro puntos es:

$G_B = [P_1\ P_2\ P_3\ P_4]$.

De esta forma, la matriz M_{HB} que define la relación $G_H=G_B.M_{HB}$ entre la matriz de geometría hermitiana G_H y la matriz de geometría de Bezier G_B es:

$$G_H = [P_1\ P_4\ R_1\ R_4] = [P_1\ P_2\ P_3\ P_4] \cdot \begin{bmatrix} 1 & 0 & -3 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & -3 \\ 0 & 1 & 0 & 3 \end{bmatrix} = G_B \cdot M_{HB}.$$

Y finalmente la matriz base de Bezier M_B:

$$M_B = M_{HB} \cdot M_H = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Haciendo cuentas se obtiene:

$Q(t) = G_B.M_B.T = Q(t) = (1 - t)^3P_1 + 3t(1 - t)^2P_2 + 3t^2(1 - t)P_3 + t^3P_4$.

Los cuatro polinomios B_B son denominados polinomios de Bernstein y son las ponderaciones de la ecuación (A1.7).

Puede observarse que la suma de los polinomios B_B siempre es igual a la unidad y que cada polinomio siempre es no negativo para 0<= t <1 (figura N° AN.N). Por lo que Q(t) es un promedio ponderado de los cuatro puntos de control. Ésto significa que cada segmento de curva está totalmente contenido en la envolvente convexa de los cuatro puntos de control.

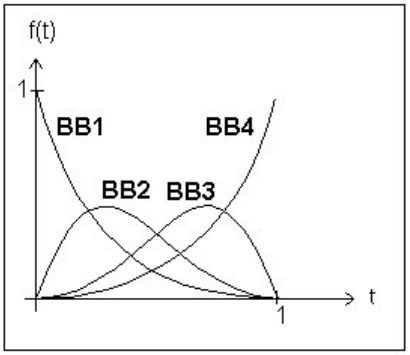


Figura N° A1.5 – Polinomios de Bernstein

Al ser la suma de los cuatros polinomios la unidad y cada uno de ellos no negativo para 0<=t<=1, Q(t) es simplemente un promedio ponderado de los cuatro puntos de control, lo que implica que cada segmento de curva está totalmente contenido en la envolvente convexa de los cuatro puntos de control. Esta propiedad es útil porque se puede determinar el valor del cuarto polinomio, para cualquier valor de t, restando los tres primeros a la unidad, pudiéndose aprovechar para reducir el tiempo de ejecución. Además, se pueden recortar segmentos de curva con mayor facilidad. En la figura N° A1.6 se pueden apreciar dos curvas de Bézier y sus envolventes convexas determinadas por los puntos de control.

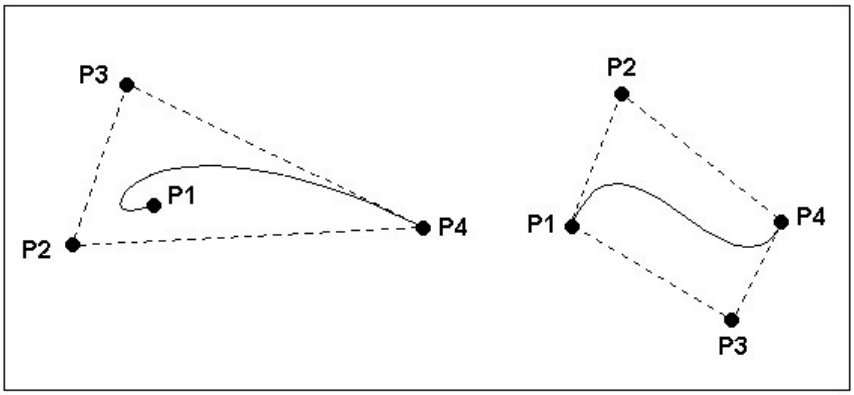


Figura N° A1.6 - Envolventes convexas para curvas de Bézier

Las curvas de Bézier pueden utilizarse para implementar fácilmente la continuidad G^1, puesto que es suficiente con alinear los dos puntos de control que determinan la tangente para cada curva en su punto de unión. En la figura N° A1.7 puede apreciarse la unión de dos curvas de Bézier utilizando la linealidad en su punto de contacto.

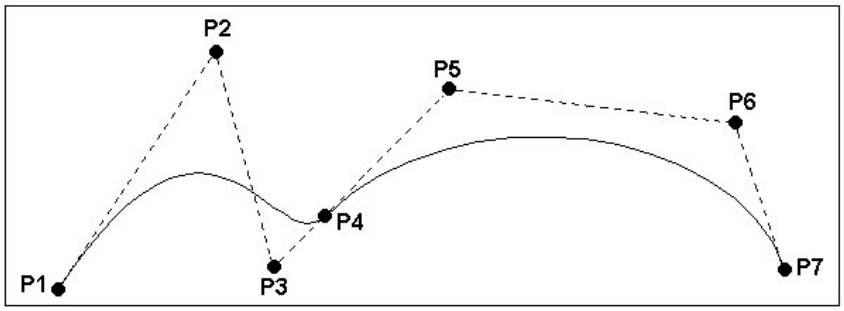


Figura N° A1.7 - Dos curvas de Bézier con P4 como punto común. P3, P4 y P5 son coolineales.

A2 Instalación del Prototipo y ejemplo de prueba

Para poder utilizar el prototipo se debe tener instalado el compilador de C5. Como C5 sólo corre sobre Linux o Solaris, el uso del prototipo construido está restringido a estos sistemas operativos. Además, C5 puede ejecutarse sólo si está disponible el compilador gcc (GNU).

Para instalar el compilador de C5 se deben realizar los siguiente pasos:

1- Asegurarse que el compilador gcc (GNU) está instalado en su computadora y que el directorio en el que se encuentra está presente en el path.

2- Copiar el compilador a cualquier directorio del disco.

3- Configurar el compilador:
* C5LIB - Editar los archivos bin/ <linux | solaris> /c5 y lib/<linux|solaris>/C5_makefile y establecer el path de la librería C4LIB correctamente.
* Editar su ".login" or ".bash_profile" o su equivalente y agregar al path del directorio bin de C5.

Ejemplos

```
set path=($path ~/home/c5/bin/solaris)
PATH=$PATH:/home/c5/bin/linux
```

Para instalar el prototipo del generador de fonts simplemente se debe agregar su módulo (Fonts.c5) en la librería de C5. Luego, para poder utilizar sus funcionalidades debe importarse su módulo desde el archivo (.c5) que lo utiliza.

Para generar la página con formato Postcript, a partir de un archivo de C5, previamente se debe compilar el código utilizando el comando c5. Por ejemplo, para compilar el archivo Prueba_FONTS.c5 se debe utilizar el comando c5 Prueba_FONTS.c5. Finalmente se debe crear la página utilizando el comando c5show.

En el archivo Prueba_FONTS.c5 se tiene un módulo de prueba para el generador. A continuación se explica su código, el cual puede ser modificado para poder apreciar los distintos resultados que se obtienen de las combinaciones de parámetros.

```
1. #include <stdio.h>
2. #include "C5_usrdefs.h"
3. #include "C5_defs.h"
4. #include "TW_libdefs.h"
5. #include "Fonts.c5"
6.
7. #define fuente_romana 1
8. #define Max_Char 40
9.
10. /* Creo algunos tipos de letra para la fuente romana, cuyos parametros son los siguientes:
11.  {fuente, inclinacion, ancho_horizontal, ancho_vertical, ancho_right, ancho_left, ancho_vertice,
12.   nivel_superior, nivel_medio_superior, nivel_medio_inferior, nivel_inferior,
13.   ancho_detalle, alto_detalle, alto_final_detalle, detalle_curvo, final_curvo, grosor}*/
14.  #define Roman_char {fuente_romana, 0.0, 0.05, 0.1, 0.1, 0.05, 0.025, 1.0, 0.75, 0.2, 0.0, 0.1, 0.1, 0.05, 1, 0, 0.5}
15.  #define Courier_char {fuente_romana, 0.0, 0.05, 0.05, 0.05, 0.05, 0.12,1.0, 0.75, 0.2, 0.0, 0.15, 0.05, 0.05, 1, 0, 0.5}
16.  #define Arial_char {fuente_romana, 0.0, 0.1, 0.1, 0.1, 0.1, 0.12, 1.0, 0.75, 0.2, 0.0, 0.0, 0.0, 0.0, 0, 0, 0.0}
17.   #define Gothic_char {fuente_romana, 0.0, 0.15, 0.15, 0.15, 0.15, 0.2, 1.0, 0.75, 0.2, 0.0, 0.05, 0.05, 0.02, 1, 0, 0.5}
18.   #define TIE_caracter {fuente_romana, 0.0, 0.1, 0.1, 0.1, 0.1, 0.1, 1.0, 0.75, 0.2, 0.0, 0.1, 0.1, 0.02, 1, 0, 0.5}
19.
```

```
20.  #define TIE_string {0.2}
21.
22.  DT_typedef char Roman_char String[Max_Char] TIE_string;
23.
24.  main() {
25.
26.  /* Creo el port sobre el que se debe representar el string */
27.  Port_List lp_string =  opm_scale(1.0,1.0,0.62,0.62, opm_page(Black, NULL));
28.  Port_List lp =  opm_scale(1.0,1.0,0.62,0.62,opm_page(Red, Blue, Green, Orange, Black, NULL));
29.
30.  /* Asigno un valor para el string */
31.  String mystring = "VAMOS Ebjn";
32.
33.  /* Despliego el resultado */
34.  opm_print(lp_string);
35.  opm_print(obtener_Puertos_String(lp, DT_pair(String, mystring),-2));
36.  }
```

Las líneas 1 a 4 definen las 4 importaciones obligatorias que se deben realizar para utilizar el compilador de C5. Mientras que la línea 5 importa el módulo del prototipo.
La línea 7 define el identificador de la fuente romana.
La línea 8 define el largo máximo del string.
Las líneas 14 a 18 definen 5 tipos de letra para la fuente romana.
La línea 20 define el único parámetro que se utiliza para el manejo de strings: la separación entre caracteres como razón del ancho del puerto de un caracter.
La línea 22 define el tipo String con una TIE para el tipo char, que define los parámetros de letra, y una TIE para el string, que define los parámetros del mismo.
Las líneas 26 a 28 definen el port donde se desplegará el string y otro idéntico para utilizar como fondo y poder apreciar las dimensiones del mismo.
En la línea 31 se asigna la cadena de caracteres al string.
En las líneas 34 y 35 se despliegan el puerto y el string.

Los tipo de letra definidos son Roman, Courier, Gothic y Arial. Además se tiene la definición de un tercer tipo, el cual puede modificarse para comprobar los resultados.

Como en C5 sólo pueden importarse módulos que se encuentren presentes en la librería del compilador, si se pretende modificar el diseño del conjunto de caracteres definido, se debe recompilar el código del prototipo y reemplazar el archivo Fonts.c5 de la librería con su resultado.
Esta compilación no debe generar código objeto, si no que debe reagrupar el código estructurado en un único módulo. El comando que realiza esta compilación es el siguiente:

gcc -E -o Dir_Compilador_C5/Fonts.c5 Dir_Prototipo/Motor/Principal.c,

donde Dir_Compilador_C5 es la ubicación de la carpeta en la que está instalado el compilador de C5 y Dir_Prototipo la ubicación de la carpeta en la que está el código fuente del compilador.