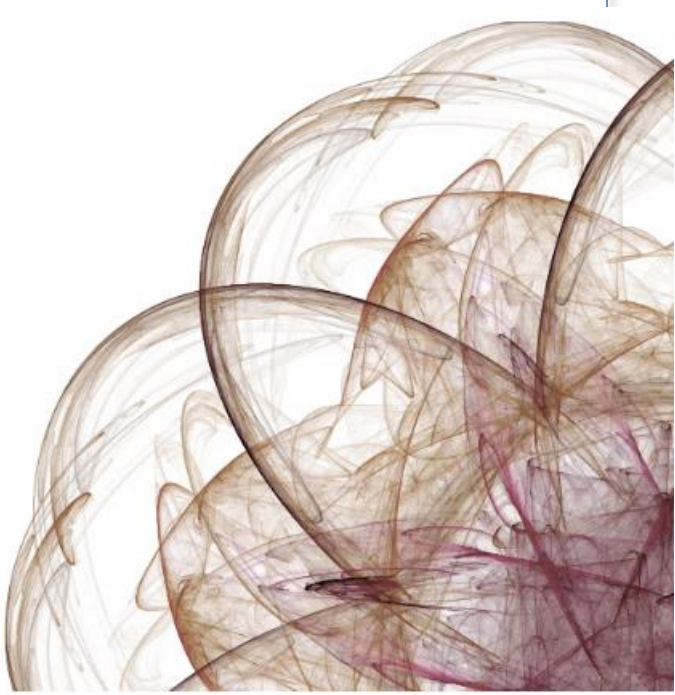
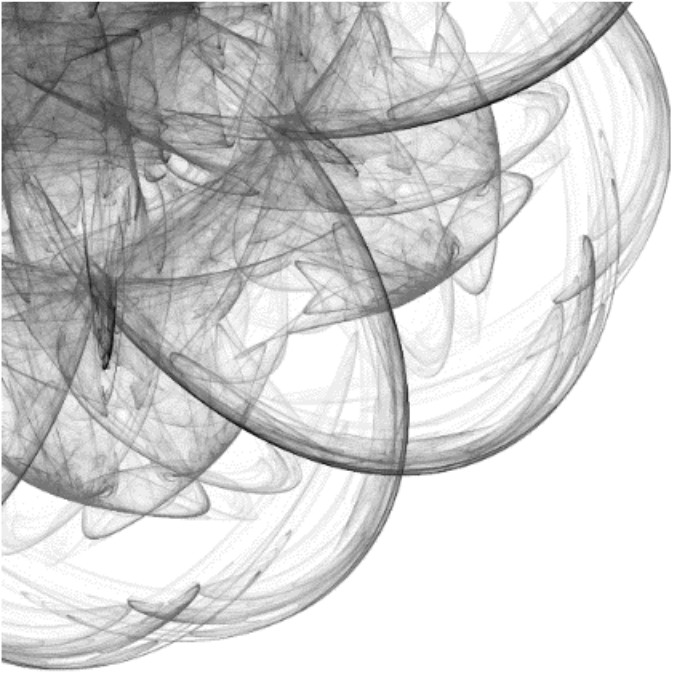
DEMOSCENE

El Arte Digital

Vladimir Kameñar





Demoscene, el Arte Digital

Vladimir Kameñar

CelerSMS

Bogota, Colombia

CelerSMS S.A.S. es una editorial independiente.
ISNI 0000 0005 0265 593X
ISIL OCLC-COCEL
Ringgold ID 598732

Para más información: info@celersms.com
Sitio web: www.celersms.com

Datos de catalogación bibliográfica

Kameñar, Vladimir.
Demoscene, el arte digital, 1a ed.
CelerSMS, Bogotá, 2021
ISBN: 978-958-53602-1-1
Materia: Informática 681.3
Páginas: 51

ISBN 978-958-53602-1-1
OCLC 1269270953

Fecha de publicación: 30 de septiembre de 2021

Contenido

Prefacio

Tengo la convicción de que demoscene es un universo de creatividad. Este universo, creado por una comunidad internacional de artistas gráficos, músicos y programadores talentosos, influenció la industria de los videojuegos, el *hardware*, las tecnologías de visualización y animación, la música electrónica.

De hecho, creo que demoscene impulsó la evolución de toda la ciencia de la computación desde sus inicios. La subcultura de demoscene se originó en los años 80, cuando los aficionados a la computación y programación no teníamos acceso a Internet (muchos ni siquiera teníamos computador). Las redes punto-a-punto como *FidoNet* aún no tenían cobertura global en ese entonces [Bush]. Por lo tanto, el conocimiento era compartido en pequeños grupos o clubes locales. Demoscene introdujo los primeros eventos masivos, a los cuales podían asistir miles de personas de diferentes ciudades, regiones. Esto ayudó a dinamizar el intercambio de conocimiento y herramientas, motivó a muchas más personas para que se dedicaran a la computación.

Hoy en día UNESCO reconoce a demoscene como patrimonio cultural en Finlandia y Alemania. Además, es importante destacar que este patrimonio es considerado como primer ejemplo de cultura digital [Kopka].

Según la definición comúnmente aceptada, aunque no enciclopédica, demoscene es una subcultura, la cual se desarrolló en torno a la creación de contenidos audio-visuales digitales, generalmente de tamaño muy compacto, y los eventos artísticos donde compiten dichas obras. Este libro pretende ampliar la definición, exponer demoscene como una forma de arte, presentar sus orígenes y tradiciones, destacar sus logros. También se busca desvirtuar algunos mitos. El mayor propósito es que quienes no conocen demoscene se animen a conocerla.

El público objetivo es quien tenga interés en la computación y el arte digital. La mayor parte del contenido no requiere tener conocimientos en programación. Solamente el último capítulo, en el cual se profundiza en las técnicas de optimización y compactación, requiere tener al menos conocimientos básicos de programación. Demoscene tiene su propia filosofía de programación, basada en el perfeccionismo. Conocerla puede ayudar a comprender los problemas de programación desde un ángulo distinto.

Quienes deseen incursionar en demoscene y crear sus propias demos, pueden encontrar en este libro las bases y referencias para hacerlo. Sin embargo, no se trata de un curso práctico de demoscene. Enseñar a hacer demos sería un objetivo demasiado ambicioso para un libro. Además, demoscene supone una actividad colectiva, ya que combina el arte gráfico, la música y la programación. No es común que una sola persona realice la totalidad de una demo. Cada demo-grupo adquiere su propio estilo y *know-how*, producto de la experimentación.

Actualmente, el cubrimiento de demoscene en la literatura hispana es bastante limitado, a pesar de la existencia de demo-grupos hispanoparlantes, algunos de los cuales son ampliamente reconocidos. Casi la totalidad de la terminología en demoscene carece de traducción oficial. Por lo tanto, el uso de anglicismos en este libro es amplio. Algunos demo-grupos hispanos se refieren a sus obras como *producciones* o *prods* en lugar de demos. También utilizan el término *escena*, aunque no tan ampliamente como demoscene.

Algunos ejemplos en este libro utilizan el lenguaje FreeBASIC, ya que su sintaxis es simple y amigable. Además, existe un gran número de ejemplos de demos para este lenguaje. En realidad, es posible implementar demos en cualquier lenguaje de programación. Los demosceners más experimentados suelen usar lenguajes de bajo nivel como C. El uso de lenguaje ensamblador también es muy común, ya que permite tener total control sobre el código ejecutable y facilita su optimización y compactación. Esto último es particularmente importante para lograr que el tamaño de la demo no exceda los límites preestablecidos. Todas las herramientas de desarrollo que aparecen mencionadas en este libro son abiertas y gratuitas, al menos hasta el momento de la publicación.

Finalmente, me gustaría expresar un agradecimiento especial para el equipo **Hi-Tech** (wasm.ru) por promover la programación Zen, una filosofía que aprecia el valor estético del código de la misma manera como se hace en demoscene. Una prueba de ello es «*La historia de un byte*», publicada originalmente en *FidoNet*, en la cual se relata la frustración que puede ocasionar la optimización del código [Galuscenko]. Los gurú de Hi-Tech son recordados por sus publicaciones, tutoriales, herramientas. También por definir los primeros códigos de ética en la ingeniería inversa. Igualmente ha sido invaluable el apoyo de **Asterix**, quien ha trabajado con música sintetizada para demoscene durante décadas.

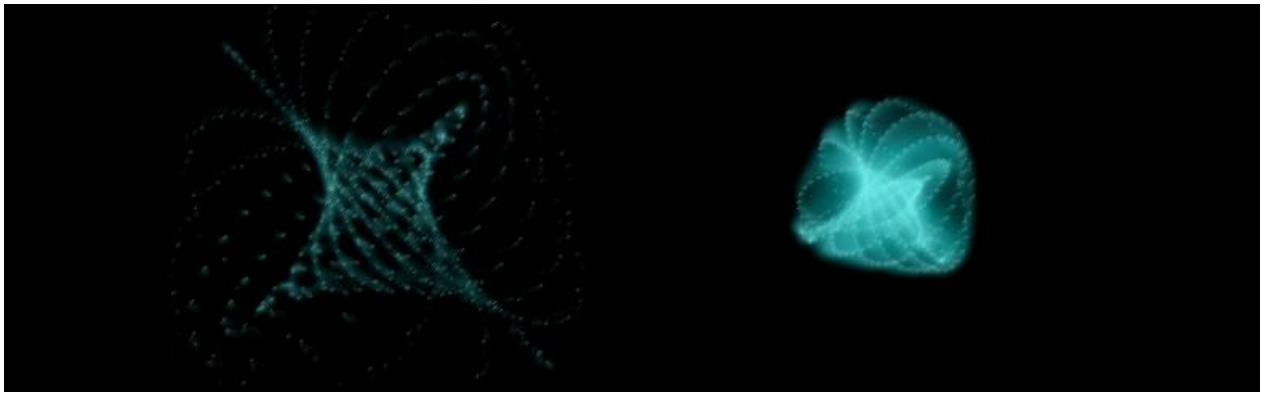
Vladimir Kameñar
Bogotá, Colombia
Septiembre de 2021

¿Qué es Demoscene?

«*El trabajo produce satisfacción estética*»
Sholom Gherman, 1970

Demoscene es un arte que combina la programación con los gráficos, efectos visuales, animación y música. No es lo mismo que los clips de video y las animaciones por computador, ya que en demoscene los contenidos se crean y se combinan de manera manual, utilizando lenguajes de programación y herramientas autóctonas. Además, típicamente se imponen límites de tamaño estrictos para los archivos que generan estas presentaciones. Entonces, se busca incluir la mayor cantidad posible de contenido sin exceder el límite de espacio impuesto. Estos archivos o presentaciones son conocidos como *demo* o *intro*. El término *demo* hace referencia a una demostración de habilidades. El término *intro* se usa cuando se trata de una introducción para presentar un videojuego, álbum, catálogo, etc. Por ejemplo, una de las modalidades comunes en demoscene son las demos de 64K. Esto significa que el tamaño del archivo no debe superar 64 kilobytes. Puede resultar sorprendente que en un archivo tan compacto se incluyan varios minutos y hasta horas de animaciones 3D y música. Es aún más sorprendente que existan modalidades de 32K, 16K, 8K, 4K, etc.

El contenido puede ser tan simple como unas figuras geométricas animadas. A continuación, un ejemplo de toroide para OpenGL, creado por **rel** en lenguaje Basic. Este ejemplo se incluye dentro del paquete de instalación de FreeBASIC[1] y ocupa alrededor de 100 líneas de código.



Toroide de **rel**

Las demos más reconocidas suelen tener el mismo tamaño de ejecutable que el ejemplo de toroide, pero incluyen muchísimos más elementos gráficos, animaciones, efectos de sonido, etc.

[1] *FreeBASIC* https://sourceforge.net/projects/fbc/files/

Por ejemplo, la demo *Paradise* del grupo español **rgba**[2] ocupa tan solo 64Kb e incluye 8 minutos de animación con figuras texturizadas de animales, plantas, paisajes.

Demo *Paradise* de **rgba**

La capacidad gráfica de esta demo, entre muchas otras obras de arte en demoscene, se puede comparar con los videojuegos actuales, aunque *Paradise* fue publicada en 2004. Esta es una de las razones, por las cuales se dice que demoscene influyó en la industria de videojuegos. Por eso, los festivales de demoscene, conocidos como *demo-parties*, suelen ser patrocinados por las compañías que desarrollan videojuegos y los fabricantes de hardware gráfico y de sonido de alta gama. También es común que los artistas y programadores de demoscene, conocidos como *demosceners*, creen sus propias empresas o se vinculen con las compañías existentes en estas industrias. Por ejemplo, Jaakko Lisalo, el creador del popular juego *Angry Birds*, es conocido por sus obras gráficas y musicales en demoscene en los años 90.[3] A finales de la década de los 2000 una gran parte de la industria de videojuegos era conformada por antiguos demosceners.[4]

Sin embargo, la creación de videojuegos actuales no se asemeja a la creación de demos. En demoscene se busca perfeccionar y optimizar el código, minimizar el uso de espacio en disco,

[2] *Demo-grupo rgba* https://www.rgba.org/

[3] *Edge Staff* (2011) http://web.archive.org/web/20121225163908/http://www.edge-online.com/features/meet-man-behind-angry-birds/

[4] *Carlsson, Anders* (2015) https://chipflip.wordpress.com/2015/06/12/famous-people-who-came-from-the-demoscene/

ensayar efectos originales. A veces, no importa que dichos efectos no se visualicen correctamente en otro hardware. La originalidad artística y las habilidades de programación son lo que más se valora. Si los videojuegos se desarrollaran de esta misma manera, entonces tardarían demasiado en ser publicados y no necesariamente funcionarían de manera aceptable en diferentes computadores.

No se debe confundir las competencias de demoscene, también conocidas como *compo*, con las competiciones de programación. En las últimas se busca demostrar la superioridad en la programación al resolver ejercicios específicos dentro de un tiempo límite. En cambio, en las compo no se evalúan las habilidades o conocimientos, sino el producto artístico. Las obras más espectaculares son las que logran captar la atención del público y los jurados. Muchas veces el mismo público hace las veces de jurado. Otra diferencia importante es que la mayoría de los demosceners utilizan pseudónimos. Inclusive pueden utilizar más de una identidad en diferentes eventos. La cultura de anonimato es muy característica en demoscene, ya que no se busca un reconocimiento. Muchos programadores, incluyendo a quien escribe, han disfrutado participar tanto en las compo de demoscene como en las competencias de programación, ya que estos eventos se complementan perfectamente.

Se cree que demoscene también tiene conexión con los orígenes del diseño web [Chris]. Esto se basa en la similitud entre las primeras comunidades de desarrolladores web en los años 90 y las comunidades de demoscene. En ambos casos se promovía la creatividad colectiva, la publicación de código y herramientas por medio de foros, la reutilización anónima de elementos estéticos.

Naturalmente, una mejor manera de conocer el arte consiste en apreciar las obras. La mayoría de las demos para PC pueden ser reproducidas en cualquier computador. Por seguridad, es importante evitar descargar ejecutables de sitios no confiables, ya que pueden contener *malware*. En las plataformas de video, como YouTube, se puede encontrar grabaciones de demos, pero apreciarlas en vivo es más impactante.

Un poco de historia

«*Aprende las reglas como un profesional, para que puedas romperlas como un artista*»
Pablo Picasso

El arte siempre ha hecho parte de la computación, desde los tiempos en que las pantallas eran monocromáticas, solamente permitían desplegar texto y el único dispositivo de sonido era el zumbador o *buzzer*. Las expresiones gráficas y musicales que se originaron en esos tiempos desafiaron los límites de los primeros computadores personales. Dichas expresiones pueden ser consideradas como precursoras de demoscene.

Pseudográficos

Es una forma de simular gráficos usando solamente caracteres de texto. Generalmente se pretende utilizar únicamente los caracteres básicos para evitar distorsión en dispositivos o fuentes diferentes. Por ejemplo:

Oso de peluche formado por caracteres de texto

Estas gráficas surgen junto con las primeras terminales e impresoras, las cuales soportaban únicamente texto. Existe la teoría de que los *pseudográficos* son aún más antiguos. Es posible que se hayan usado en tipografía, mucho antes de que existieran los computadores [Stark]. Hoy en día estas gráficas pueden parecer primitivas, pero no lo eran a finales de los años 70, cuando los computadores aun no disponían de adaptadores gráficos.

Actualmente se utiliza el término *arte ASCII*[5] para hacer referencia a estas formas de expresión artística. Existen catálogos extensos con obras de arte ASCII en línea, como *ASCII Art Archive*.[6] Hoy en día esta forma de arte es usada para personalizar los textos en foros y chats, mensajes de bienvenida en los servicios accedidos por medio de terminales de texto. Otro ejemplo de uso común son los archivos informativos, como LEEME.TXT,[7] los cuales acompañan el software u otros contenidos descargables. Estos archivos suelen incluir arte ASCII.

El famoso *shrug*[8] en una sola línea, entre otras expresiones textuales abreviadas, también es un pseudográfico:

¯_(ツ)_/¯

Las pantallas a color y los primeros adaptadores gráficos, introducidos a comienzos de los años 80, revolucionaron los pseudográficos:

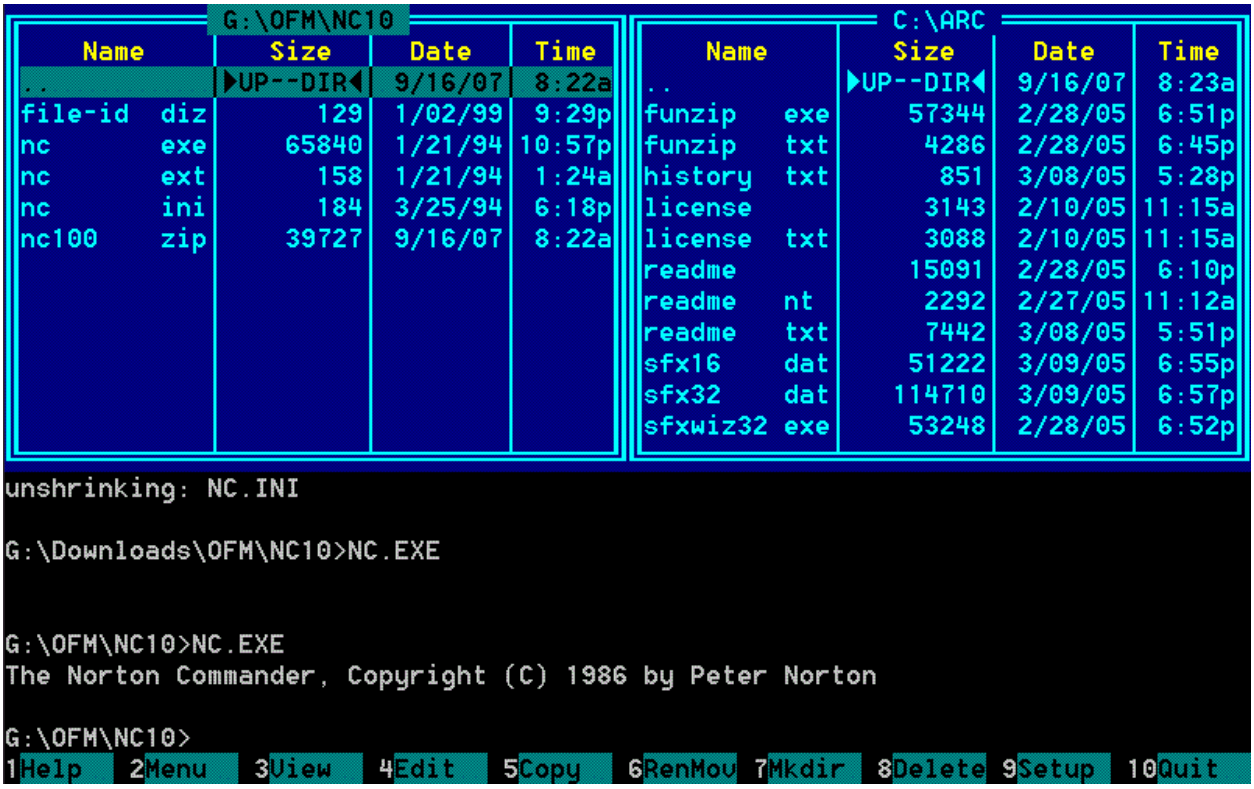
Papá Noel formado por caracteres ASCII a color

Los pseudográficos también fueron utilizados en las primeras interfaces de usuario. El estándar Unicode incluye un bloque de 128 caracteres llamados *Box Drawing*.[9] Estos símbolos hicieron parte del conjunto de caracteres de IBM PC:

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
B				┆	┤	╡	╢	╖	╕	╣	║	╗	╝	╜	╛	┐
C	└	┴	┬	├	─	┼	╞	╟	╚	╔	╩	╦	╠	═	╬	╧
D	╨	╤	╥	╙	╘	╒	╓	╫	╪	┘	┌					

[5] Es común el uso de los anglicismos *ASCII art* o *ASCII artwork*.
[6] https://www.asciiart.eu/
[7] El nombre del archivo puede ser cualquiera, como README.TXT, README.NFO. etc.
[8] Simboliza a una persona con los hombros encogidos en señal de desconocimiento.
[9] *Unicode 13.0*, http://unicode.org/charts/PDF/U2500.pdf

Estos caracteres se utilizan para dibujar líneas horizontales y verticales, polígonos rectangulares. Por ejemplo, el administrador de archivos **Norton Commander** para MS-DOS utilizaba estos mismos caracteres para generar los bordes:



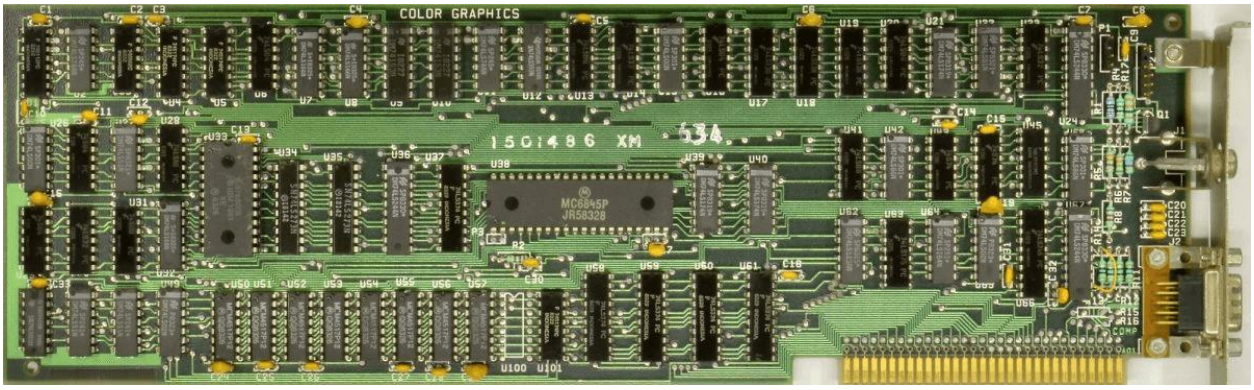
Interfaz de usuario de **Norton Commander** en MS-DOS

Actualmente las interfaces de usuario basadas en texto siguen siendo ampliamente utilizadas en equipos de comunicaciones, sistemas operativos como Linux y Unix. El uso de pseudográficos en este tipo de sistemas sigue siendo aplicable.

Pixel art

El *adaptador gráfico*, también conocido como *tarjeta gráfica* o *tarjeta de video*, es un dispositivo de hardware que se encarga de generar las imágenes que luego son desplegadas en el monitor, proyector, etc. Los adaptadores gráficos en color convierten el espacio de la pantalla es una matriz bidimensional, en la cual cada casilla, conocida como *pixel*, tiene un color específico.

El primer adaptador gráficos a color para PC aparece en 1981. Este adaptador introdujo el estándar **CGA** (*Color Graphics Adapter*) de hasta 16 colores con una resolución de 160x100 pixeles. La resolución se podía ampliar hasta 320x200, pero con una paleta de 4 colores, o hasta 640x200 pixeles, pero en modo monocromático.



Adaptador CGA original de IBM

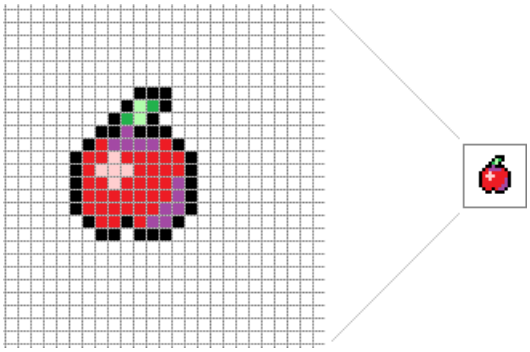
Sin embargo, los usuarios preferían comprar los PC con adaptador monocromático, llamado **MDA** (*Monochrome Display Adapter*), por su menor costo. Además, el adaptador monocromático generaba textos más nítidos. Los adaptadores a color conquistaron el mercado solamente hasta finales de los años 80. El adaptador CGA fue reemplazado por **EGA** (*Enhanced Graphics Adapter*) en 1984. Luego, en 1987, EGA fue reemplazado por **VGA** (*Video Graphics Array*). En la sigla VGA no aparece el término adaptador, ya que el nuevo estándar para gráficos, a diferencia de sus predecesores, era fabricado como un único chip, no como una tarjeta independiente con diferentes circuitos integrados. Por lo tanto, VGA podía ser una tarjeta de expansión o un componente integrado en la tarjeta madre. También existieron otros adaptadores gráficos para PC, como SVGA y XGA.

El juego *SimCity* en VGA en 1993

Las gráficas se popularizaron primero en los computadores Amiga y Macintosh, aunque fueron introducidas antes en los PC de IBM. Por ejemplo, el popular juego *SimCity*, creado por Maxis, inicialmente se desarrollaba para Commodore 64. Los primeros computadores Amiga, fabricados por Commodore, soportaban miles de colores e incluían tarjeta de sonido. Por eso, el origen de demoscene tiene que ver con Amiga, mucho más que con PC.

La limitada resolución y profundidad de color en los primeros adaptadores gráficos dio origen a una nueva forma de arte: **pixel art**.

El propósito de pixel art para las gráficas de baja resolución es equivalente al de pseudográficas para las pantallas de texto. Al seleccionar cuidadosamente el color de cada pixel se busca suavizar la grilla impuesta por el adaptador gráfico. Por ejemplo:



Inicialmente esta forma de arte era completamente manual, similar a la antigua técnica de bordado a punta de cruz.

A medida que crecía la resolución y la profundidad de color de los adaptadores gráficos la importancia de dibujar manualmente cada pixel se ha ido reduciendo. Hoy en día aún se puede apreciar pixel art en los videojuegos hechos al estilo de consola, especialmente en 2D y 2.5D. Por ejemplo, en el portal *Indie Retro News*[10] se publican novedades acerca de estos juegos, la mayoría de los cuales son gratuitos.

Cracktro

Las raíces de demoscene también están en las comunidades de *cracker*, las cuales se dedicaban a eliminar la protección anti-copia en los primeros videojuegos a finales de los años 70 [Collins]. En muchos países aun no existía una regulación en materia de derechos de autor para software. Por lo tanto, la piratería de juegos y otros programas funcionaba de manera abierta, no era vista como algo negativo.

La escases y alto costo de los disquetes[11] en esos tiempos hacían que los usuarios compraran estos insumos de segunda. Muchas veces los disquetes usados incluían juegos y otros programas sin licencia. Por lo tanto, la piratería funcionaba inclusive de manera involuntaria.

Generalmente, luego de liberar un juego de la protección anti-copia el cracker le agregaban una especie de introducción con gráficas y música. Estas introducciones se llamaron **cracktro** o **crack intro** y fueron precursoras de las *intro* de demoscene. A veces, las cracktro tenían mejores

[10] https://www.indieretronews.com/
[11] Los disquetes o discos flexibles eran medios magnéticos removibles utilizados a partir de los años 70.

gráficas que el juego como tal. Las cracktro servían como una firma del grupo o la persona que logró crackear el programa.

Las cracktro evolucionaron y se hicieron independientes del cracking. Entonces, comenzaron a ser distribuidas como obras artísticas. Eventualmente se convirtieron en lo que hoy en día se conoce como demo [Green].

Anders Carlsson de la Universidad de Lund considera que las cracktro son muy similares al grafiti, excepto que se exhiben en el espacio privado y no en el público [Carlsson].

Es un error común asumir que la cultura de anonimato en demoscene se debe a que las cracktro hacían parte de una actividad ilícita. En los años 70 ni siquiera existía el concepto de propiedad intelectual de software. La oficina de patentes de EE.UU. comenzó a registrar patentes de software a partir de 1989. Entonces, en esos tiempos ni siquiera se usaba el término piratería. Por ejemplo, en 1985 la revista **Ahoy!** en EE.UU. presenta la actividad de cracking como un reto intelectual [Ahoy]. Además, la demoscene actual no tiene vínculos con la piratería. Los demosceners prefieren no revelar sus nombres reales por las mismas razones que un escritor usa pseudónimo o un músico usa un nombre artístico. La misma tradición de anonimato existía en los foros temáticos y comunidades en Internet.

Generalmente las cracktro incluyen textos animados. La animación más común es el desplazamiento horizontal en sentido contrario a la lectura. Este efecto se conoce como *ticker* o *zipper*. Por ejemplo, las pantallas digitales tipo LED, entre otras, utilizan esta misma técnica para desplegar textos largos, los cuales de otro modo no caben en pantalla.

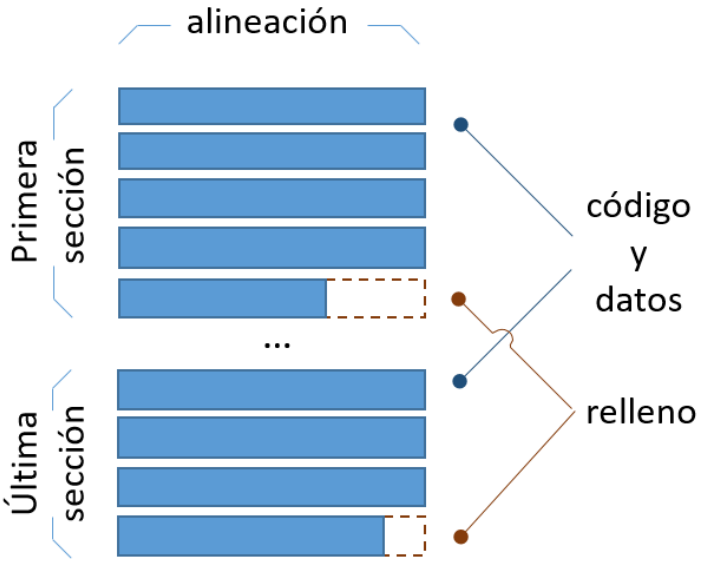
La cracktro hecha por el grupo **Hybrid** para el juego **Prehistorik 2** de **Titus** en 1993[12] incluye un texto animado en la parte superior de la pantalla:

Cracktro por **Hybrid**

[12] https://demozoo.org/productions/159111/

Las animaciones de texto utilizan algoritmos simples, los cuales ocupan muy poco espacio en código máquina. Su uso en las cracktro era común, ya que el espacio disponible para el código solía ser muy limitado.

En algunos casos el tamaño del ejecutable luego de añadir la cracktro debía ser igual que el tamaño original. Entonces, el código de la cracktro debía ser distribuido entre los espacios «vacíos» del ejecutable original. Estos espacios vacíos se usan como relleno para que los segmentos de código o datos dentro del ejecutable tengan un tamaño que sea múltiplo de cierto valor de referencia, conocido como *alineación*. Por ejemplo, en Windows el valor mínimo de alineación es 512 bytes [Wasm]. Los ejecutables suelen tener más de una sección. Además, pueden tener espacios vacíos adicionales. De todos modos, la combinación de todos estos espacios puede sumar un valor muy corto del orden de algunos kilobytes. La siguiente gráfica presenta este concepto de manera simplificada:



Espacios de relleno dentro del código ejecutable

El formato de ejecutable varía entre diferentes sistemas operativos, a saber: en MS-DOS se usaba el formato MZ, en Windows se usa el formato PE-COFF, en Linux el formato más usado es ELF. Todos estos formatos tienen una estructura que se divide en secciones o segmentos con alineación. Por lo tanto, el concepto de relleno es bastante universal.

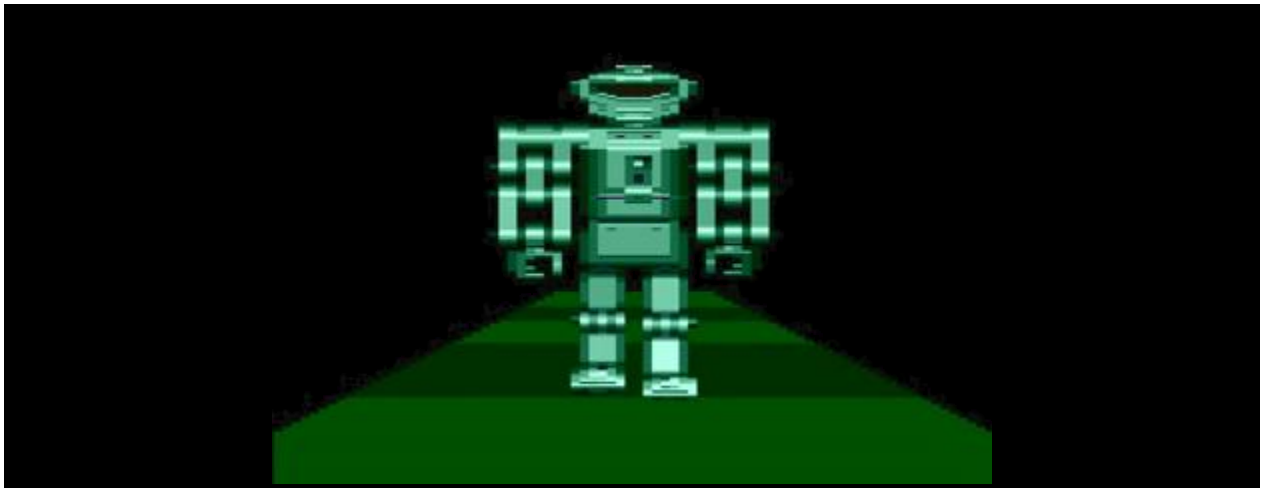
Es común que el código de las cracktro ocupe los espacios de relleno dentro del ejecutable. Sin embargo, también es posible crear o ampliar estos espacios por medio de la manipulación de los datos o el código ejecutable original. Esto resulta ser mucho más complejo, ya que cualquier cambio en el código original puede afectar otros fragmentos de código, producir errores inesperados. Por lo tanto, la primera opción son los espacios de relleno.

La misma publicación en la revista **Ahoy!** menciona que algunos desarrolladores de software dejaban mensajes ocultos para los cracker [Ahoy]. Estos mensajes se encontraban en los espacios

de relleno. También podían ser cadenas de texto que no eran usadas dentro del programa. El contenido de los mensajes podía ser un simple saludo o alguna broma.

Uso comercial

Los fabricantes de hardware, especialmente de audio y video, han utilizado obras de demoscene como elemento publicitario. Por ejemplo, **Atari** presentó una demo de un robot 3D en la feria CES[13] en 1985 para promover su último modelo de computador de ese entonces.[14]



Demo de Atari en CES 1985

El demo-grupo sueco **Triton**,[15] conocido por crear el secuenciador de audio FastTracker II, creó una demo comercial para el fabricante de tarjetas de audio **Gravis Ultrasound**. El demo-grupo finés **Future Crew**[16] ha hecho demos para **Creative Labs**. El demo-grupo alemán **Farbrausch**[17] hizo una demo comercial para la marca **Afri-Cola**[18] [Reunanen].

Las demos también fueron comercializadas al interior de demoscene. Por ejemplo, varios demo-grupos holandeses crearon demos para ser vendidas a otros grupos [Szarafinski].

Seguramente ha habido muchos más ejemplos de uso comercial. Sin embargo, la gran mayoría de las demos no se han hecho por dinero. Demoscene siempre se ha basado en la distribución gratuita de las obras [Borzyskowski]. Esta característica de demoscene difiere de los paradigmas de software libre, ya que los códigos fuente de las demos generalmente no son públicos. Además, en demoscene no existe un concepto de licencia con términos y condiciones. Por lo tanto, el

[13] *Consumer Electronics Show*, una feria de tecnología que se realiza en Estados Unidos desde los años 60.
[14] *Demo de Atari en CES 1985*. https://www.youtube.com/watch?v=pP4YtHW7Gh0
[15] *Demo-grupo Triton*, https://demozoo.org/groups/337/
[16] *Demo-grupo Future Crew*, https://demozoo.org/groups/357/
[17] *Demo-grupo Farbrausch*, https://demozoo.org/groups/211/
[18] *Afri-Cola* es una bebida gaseosa producida en Alemania.

modelo de distribución de las demos no es igual que el software comercial, ni tampoco el de software libre.

Las competencias de demoscene, descritas más adelante, podían ofrecer premios en dinero, pero estos premios no buscan compensar el tiempo y esfuerzo necesarios para crear una demo. La recompensa más grande siempre ha sido la aprobación del público.

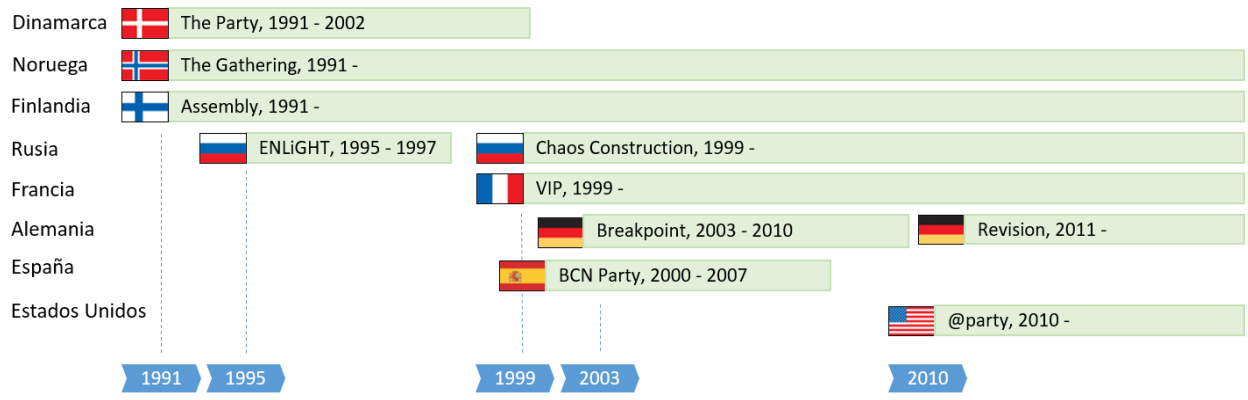
Demo-parties

Una parte integral de demoscene son los festivales donde se exhiben las demos y se organizan competencias. Estos eventos se llaman *demo-parties* o *demo-shows*. Específicamente las competencias se llaman *compos*. El primer objetivo de las demo-parties es socializar, hacer amistades. Estos eventos se originaron cuando la interacción social en Internet aun no existía o se limitaba al intercambio anónimo de mensajes en los foros. Los interlocutores utilizaban alias. Entonces, la posibilidad de reunirse e interactuar cara-a-cara despertaba interés genuino. De hecho, las competencias surgieron a mediados de los años 90. Hasta ese entonces el propósito de las reuniones se limitaba a la socialización e intercambio de conocimientos y herramientas.

Varios países se disputan el título de origen de demoscene. Dinamarca, Finlandia y Noruega son los países donde se fundaron las demo-parties más grandes:

- **The Party** en Dinamarca
- **Assembly** en Finlandia
- **The Gathering** en Noruega

Estas demo-parties surgieron de manera casi simultánea entre 1991 y 1992. Es posible que eventos afines o de menor escala se hayan realizado previamente, inclusive en otros países. La siguiente gráfica presenta la cronología de algunas de las demo-parties más importantes a nivel mundial:



Las demo-parties se han realizado y se siguen realizando en la mayoría de los países de Europa. También se realizan en algunos países de América, África y Asia. Igualmente existen demo-parties en Australia. En 2020 muchos de estos eventos hicieron transición a un formato virtual debido a la pandemia de COVID19.

No todos los eventos se realizan de manera regular. En algunos casos los eventos no son exclusivos de demoscene, sino que hacen parte de un festival de arte digital o computación en general. Por ejemplo, las conferencias de computación gráfica **SigGraph** en América se realizan desde 1974 hasta la actualidad y ya incluyen varios países en Asia. Sin embargo, las categorías de demoscene fueron incorporadas en SigGraph solamente hasta comienzos de los años 2000.

Por lo tanto, no hay consenso en cuanto a cuál es la demo-party más importante o más antigua o más tradicional, etc.

En América Latina se destacan las siguientes demo-parties:

- **Flashparty** en Argentina, desde 1998, con interrupciones[19]
- **Art Engine** en Brasil, 2012[20]

Varios artistas de otros países, como Colombia, también contribuyen contenidos para demoscene [González]. Estos contenidos son principalmente musicales y se conocen como *chiptune* o música de 8 bits.

Prácticamente en todo el mundo existen comunidades de artistas que producen gráficas o música sintetizada. La combinación y compactación de estos elementos para crear una demo se ha convertido en un fenómeno masivo en los países donde ha existido una tradición de hacking [Svensson].

[19] *Flashparty*, https://flashparty.rebelion.digital/index.php?lang=es
[20] *Art Engine Demoparty*, http://web.archive.org/web/20130810084234/artengine.com.br/pt/main/

Gráficas

«*La esencia del dibujo es la línea que explora el espacio*»
Andy Goldsworthy

Las demos son una forma de arte audio-visual, por lo que las gráficas constituyen una parte esencial de demoscene. Por eso, cada demo-grupo debe tener al menos un integrante experto en este tema. A veces, para hacer referencia a las gráficas y efectos visuales se usa la abreviatura **gfx**.

La parte gráfica de una demo se compone no solamente de imágenes estáticas. Las animaciones también son importantes. Generalmente las animaciones se sincronizan con el audio. Se puede decir que la demo tiene una parte gráfica y una parte musical, pero las partes deben estar entrelazadas. Por ejemplo, el ritmo de la animación acelera si aumenta el pulso musical. Sin embargo, no se debe confundir las demos con las visualizaciones musicales. Las gráficas en una demo son como la letra de una canción.

Las gráficas digitales son mapas de bits. Este concepto fue introducido en los primeros adaptadores gráficos o tarjetas de video, como se presentó en el capítulo anterior, y no ha cambiado hasta la actualidad. La imagen que vemos en pantalla es una rejilla, en la cual cada casilla es un *pixel* con un color específico. La densidad de estos puntos, conocida como **resolución gráfica**, y la profundidad de color han ido creciendo con la evolución del hardware gráfico, pero el concepto sigue siendo el mismo. Por lo tanto, la forma más primitiva de dibujo consiste en describir individualmente cada pixel.

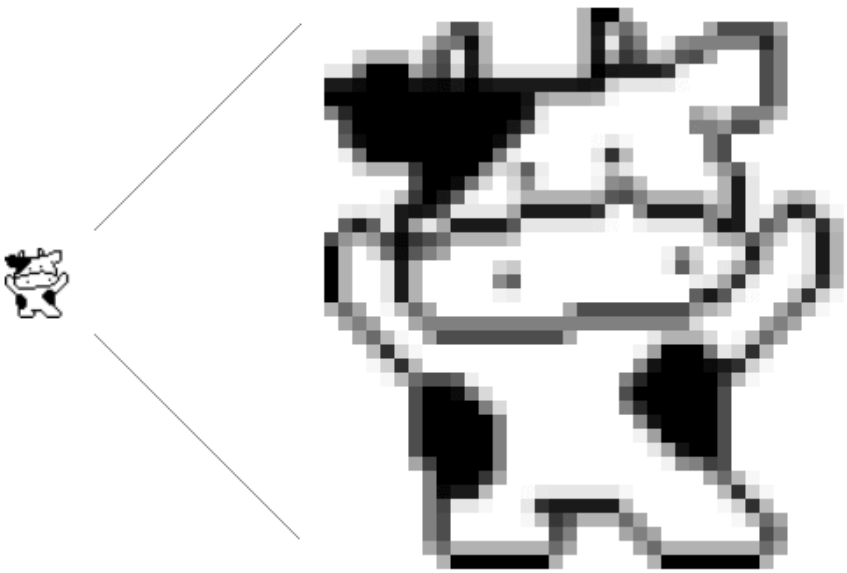


Imagen ampliada al 800% para visualizar la pixelación.

El componente de bajo nivel que permite generar gráficas primitivas en Windows se llama **GDI** (*Graphics Device Interface*). Este componente permite dibujar líneas, curvas, textos, rellenar regiones. Las capacidades de GDI son similares a las del aplicativo *MS Paint*. El equivalente de GDI en Linux es **X.Org**. En cambio, en MS-DOS las aplicaciones generaban gráficas por medio de BIOS[21] o a través de interrupciones de hardware. La introducción de un componente genérico como GDI en Windows fue un avance importante. De hecho, actualmente muchas aplicaciones gráficas y juegos simples siguen utilizando GDI como único componente gráfico.

En cambio, para generar gráficas más complejas, especialmente de tipo vectorial y 3D, se utilizan componentes como **DirectDraw** y **Direct3D** (ambos son parte de **Direct X**), **OpenGL**, **Vulkan**, etc. Estos componentes no sólo simplifican la *renderización*[22] de los modelos gráficos. También permiten aprovechar las capacidades de aceleración por hardware.[23]

Todos estos componentes son usados ampliamente en demoscene.

PCG

Las gráficas generadas de manera procedimental o **PCG** (*Procedural Generation*) emplean algoritmos para generar las imágenes. El empleo de algoritmos implica el uso de recursos de máquina. Este uso puede ser alto, hasta excesivo, si el algoritmo es complejo.

Un ejemplo simple es el árbol. Podemos comenzar por dibujar el tronco como una línea vertical. Luego se produce una bifurcación para dibujar las ramas. Sobre cada una de las ramas nuevamente se produce una bifurcación y así sucesivamente. Para mayor realismo podemos hacer que el ángulo de la ramificación, la longitud de cada rama y el número de ramas sean valores aleatorios. De hecho, es común el uso de números aleatorios en PCG.

Un árbol generado mediante PCG (izquierda) vs. una fotografía real.

[21] BIOS es el *firmware* que maneja la interfaz de entrada salida básica en PC. La interacción directa con BIOS se considera una tarea de bajo nivel. Actualmente se utiliza la interfaz UEFI en lugar de BIOS.
[22] Renderización es el proceso para sintetizar una imagen a partir de su modelo.
[23] La aceleración por hardware hace uso de la unidad de procesamiento gráfico (GPU).

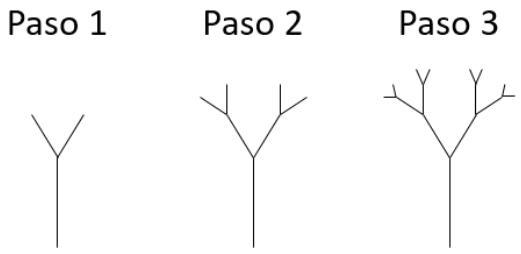
El resultado puede ser muy similar al de una fotografía real. Las diferencias contra una imagen real se hacen menos evidentes conforme disminuye el tamaño de la imagen o el tiempo de exposición (en el caso de las animaciones). Podemos variar el grosor de los trazos y el color, agregar un fondo adecuado para mejorar la imagen aún más.

Para mayor simplicidad, si omitimos los números aleatorios y asumimos que cada bifurcación tiene 2 ramas, podemos resumir el algoritmo para dibujar el árbol con tan solo 10 líneas en FreeBASIC:

```
Sub tree(x0 As ULong, y0 As ULong, sz As ULong, angle As Double, depth As ULong)
    Dim As ULong x1, y1
    x1 = x0 + sz * Cos(angle)
    y1 = y0 + sz * Sin(angle)
    Line (x0, y0) - (x1, y1), 0
    If depth > 0 Then
        tree(x1, y1, sz * 0.7, angle - 0.44, depth - 1)
        tree(x1, y1, sz * 0.7, angle + 0.44, depth - 1)
    End If
End Sub
```

Los coeficientes usados en el código anterior fueron tomados arbitrariamente.

Cada vez que se ejecuta la función `tree()` se dibuja una línea de color negro y se invoca la misma función de manera recursiva 2 veces más para dibujar las ramas. Para limitar el número de iteraciones recursivas utilizamos el contador `depth`, al cual le restamos 1 en cada iteración. Cuando el valor del contador llega a 0 el llamado recursivo finaliza. Si el valor inicial de `depth` es 3 el dibujo se realizaría en 3 pasos como sigue:



Esto no significa que la función se ejecuta 3 veces. En realidad, la función se ejecuta de manera recursiva tantas veces como el número de líneas que vemos en la imagen final (15 veces en este caso), ya que en cada ejecución se dibuja exactamente una línea. El número de ejecuciones en función del grado de recursión `depth` se puede calcular con la siguiente fórmula:

$$N(depth) = 2^{(depth+1)} - 1$$

En análisis de algoritmos esto se conoce como orden de complejidad[24] y se puede expresar como $O(2^n)$. Esto significa que la complejidad de nuestra función crece de manera exponencial. Es decir, si incrementamos en una unidad el valor de `depth` para mejorar el nivel de detalle de la imagen, el número de ejecuciones se duplica. No todos los algoritmos de PCG son recursivos. Inclusive los algoritmos recursivos como el de este ejemplo, pueden ser traducidos de manera no recursiva. Sin embargo, la complejidad seguirá siendo exponencial. Por eso es importante recordar que los algoritmos de PCG pueden generar un alto uso de recursos. Por ejemplo, si intentamos dibujar de esta manera miles de árboles con un alto grado de detalle, eventualmente se pueden producir demoras o alto uso de CPU.

Prácticamente el mismo método de PCG se puede usar para dibujar texturas, partículas, etc. Si en la demo vemos un fondo con estrellas, cristales de nieve, fuegos artificiales, es probable que estemos apreciando el resultado de PCG, como en esta obra del demo-grupo sueco **FairLight**, publicada en 2010:

Demo **Agenda Circling Forth**[25] de **FairLight**

Los algoritmos PCG suelen ser mucho más compactos en comparación con el tamaño de la imagen generada como mapa de bits. El ejemplo de árbol en FreeBASIC presentado más arriba puede ocupar menos de 200 bytes en código máquina. En cambio, el mapa de bits generado con el mismo ejemplo ocupa más de 4Kb, aun en modo monocromático y con compresión PNG. Los algoritmos PCG son ampliamente utilizados en demoscene debido a su capacidad de compactación prácticamente ilimitada.

[24] Esta notación se conoce como *O-grande* y sirve para clasificar de manera abstracta la complejidad de los algoritmos.
[25] https://www.pouet.net/prod.php?which=54603

Fractales

El estudio de los fractales tiene un significado más amplio que su aplicación práctica en las gráficas. La propiedad fundamental de los fractales es la **autosimilitud**, también conocida como **simetría expansiva** o **simetría evolutiva**. Una fórmula o un conjunto de datos, no necesariamente gráficos, que tenga esta propiedad puede ser un fractal. En términos prácticos, *autosimilitud* significa que la forma de un todo coincide, al menos parcialmente, con la forma de las partes que lo componen. En el caso de las imágenes esto implica que a cualquier escala se distingue el mismo patrón.

El estudio teórico de los fractales, junto con la primera definición del término *fractal*, comenzó en los años 70 con los trabajos del matemático Benoit Mandelbrot. Los sistemas autosimilares, en los cuales se basa la teoría de fractales, eran conocidos desde hace varios siglos. De hecho, los fractales son omnipresentes en la naturaleza. Además, durante muchos años se ha creído que el Universo completo podría ser un fractal. Estudios recientes han concluido que el Universo no es un fractal, aunque incluye patrones de simetría propios de los fractales [Sutter].

Ejemplos de fractales en la naturaleza y el Universo

PCG es un caso particular de fractales. Entonces, el ejemplo de árbol presentado más arriba también es un fractal. Más exactamente, el árbol es un ejemplo de **sistema-L** o **sistema de Lindenmayer**.[26]

Es apenas lógico que los fractales sean ampliamente usados en demoscene, ya que permiten sintetizar patrones simétricos con un grado de detalle ilimitado a partir de un código de tamaño limitado. Este código puede ser muy compacto. Por ejemplo, el ganador en la categoría de 4K en la demo-party **Assembly 2012** fue esta obra de arte llamada **Hartverdrahtet**[27] cuyo autor es **Demoscene Passivist**:

Demo **Hartverdrahtet**, tamaño 4K, 2012

Esta demo contiene un universo animado de más de 3 minutos y ocupa tan solo 4Kb. Todo el contenido gráfico son fractales. De ese modo se logra que un contenido visual tan extenso pueda ser generado a partir de un código diminuto. La compresión del código también fue esencial para compactar el tamaño a 4K. El autor comentó que esta obra le tomó aproximadamente 2 meses de trabajo.

Además, los fractales son aplicados en comunicaciones para representar patrones de tráfico. También se aplican en medicina, biología, arqueología, música, arquitectura, diseño de antenas, entre muchas otras áreas. El uso de fractales, específicamente PCG, en los videojuegos tiene gran similitud con demoscene.

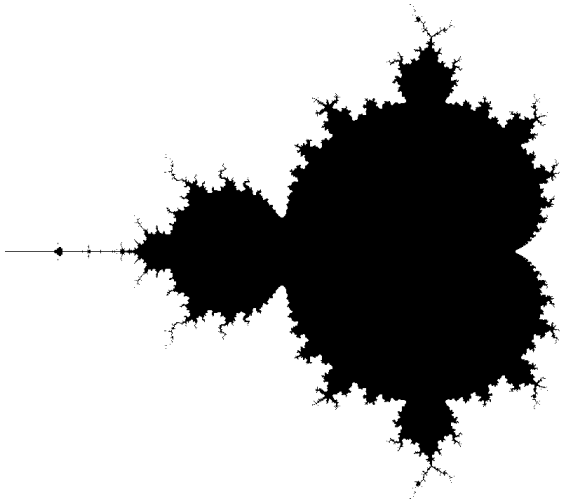
[26] Estos sistemas fueron introducidos por Aristid Lindenmayer en 1968 para describir modelos de crecimiento de las plantas. Los sistemas-L también se usan para describir fractales.
[27] https://www.pouet.net/prod.php?which=59086

El ejemplo de fractales más estudiado es el **conjunto de Mandelbrot**, nombrado en honor a Benoit Mandelbrot. Se define como el conjunto de los números complejos *c*, para los cuales la siguiente función recursiva converge a un valor finito:

$$z_o = c$$

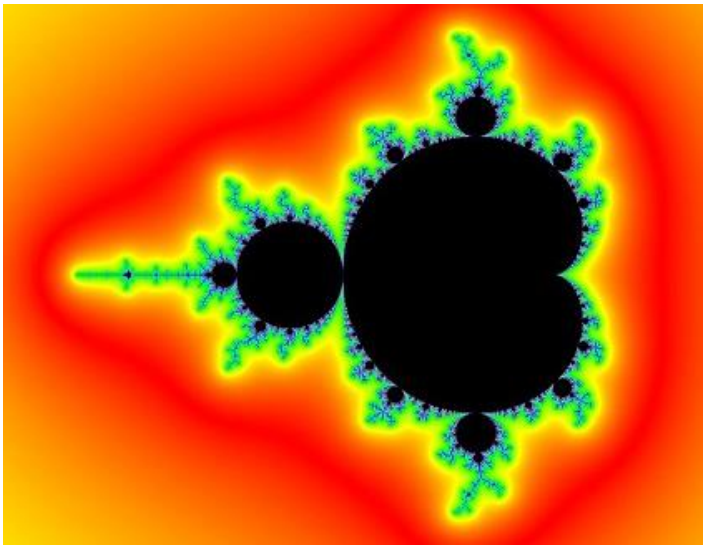
$$z_n = (z_{n-1})^2 + c$$

Gráficamente este conjunto tiene la siguiente forma:



Representación gráfica del conjunto de Mandelbrot

Al igual que en el ejemplo del árbol, el grado de detalle depende del número de iteraciones. Al agregar información de color, ya sea en el exterior del conjunto o interior o ambos, se obtiene un elemento gráfico, con el cual se puede generar texturas, rellenos, etc.



Colorización externa del conjunto de Mandelbrot con base en la distancia

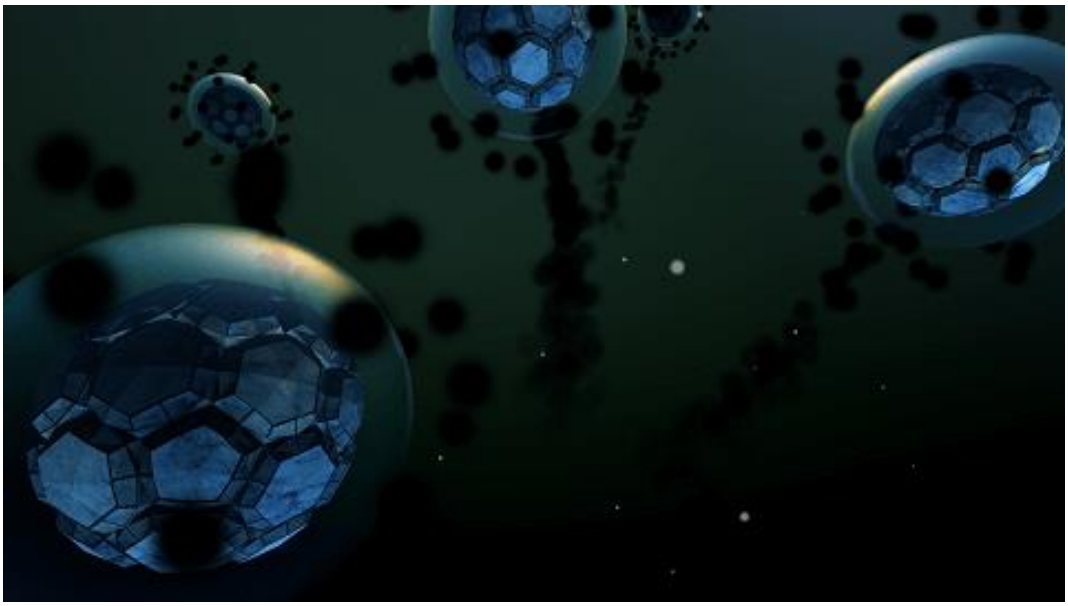
Al variar la escala se obtienen patrones gráficos nuevos gracias a la autosimilitud del fractal. Además, existen otros conjuntos de fractales, como los de Gastón Julia. La implementación de los conjuntos de Mandelbrot y Julia en FreeBASIC está disponible en las publicaciones de Jean Debord [Debord].

WebGL

WebGL es una tecnología gráfica similar a OpenGL, Direct3D, entre otras, pero funciona dentro del navegador web y utiliza API de *JavaScript*.[28] Entonces, WebGL sirve para generar gráficas en 2D y 3D dentro de una página web. Esta tecnología comenzó a ser utilizada en demoscene prácticamente a partir de su publicación a comienzos de los años 2010.

Una ventaja de las demos hechas para la web es su portabilidad. Además, no existe un código ejecutable nativo. Entonces, no hay riesgo de seguridad en la descarga y visualización de estos contenidos.

Algunos demosceners que se desenvuelven en la modalidad web tienen experiencia en el desarrollo de demos nativas tradicionales. Por ejemplo, **Fernando Serrano**, también conocido como **KILE**, hizo parte del demo-grupo **Stravaganza** en los años 2000. Posteriormente, en 2014, KILE portó[29] la demo **This way** a WebGL:



Demo **thisway.js**[30] para WebGL, 2014

[28] JavaScript es un lenguaje interpretado por el navegador. Se usa para implementar contenidos interactivos dentro de una página web.
[29] Portar significa traducir o adaptar un software para que funcione en un entorno o plataforma distinto al original.
[30] https://fernandojsg.com/lab/thiswayjs/

La tecnología web elimina todas las limitaciones que podían tener las demos nativas tradicionales, a saber: tamaño del ejecutable, configuración de resolución de pantalla, compatibilidad con diferentes GPU. Algunos demosceners tradicionales consideran que al eliminar las restricciones desaparece el desafío técnico que caracteriza a las demos. Sin embargo, algunas competencias de demos en formato web imponen un límite de tamaño para JavaScript.

Una gran colección de modelos 3D, efectos y animaciones en WebGL de código abierto se puede encontrar en el sitio *E-Notes*[31] de Evgeny Demidov.

WebGL no es la única tecnología gráfica para visualización en el navegador. Otras tecnologías, como **Flash**, **Shockwave** y **Java Applets**, también han sido utilizadas en demoscene, antes de WebGL. Hoy en día aún se utiliza el término *flashtro* o *flash-demo* para referirse a las demos hechas para la web, aunque no estén hechas en Flash.

Una limitación de las tecnologías web es su menor desempeño en comparación con el código nativo. Esto se debe a que el código JavaScript es interpretado. Además, generalmente el navegador ejecuta JavaScript con baja prioridad. Esta limitación es particularmente evidente en demoscene, ya que muchos de los efectos gráficos y animaciones se basan en algoritmos recursivos (como los fractales), los cuales se deben ejecutar de manera intensiva para generar las imágenes con un alto nivel de detalle. Una parte de estas funciones es trasladada a la GPU, pero el resto del código es interpretado por el navegador. Por lo tanto, el rendimiento puede no ser óptimo.

[31] https://www.ibiblio.org/e-notes/webgl/webgl.htm

Música

«*Ninguna explicación, ninguna combinación de palabras o música o recuerdos puede rozar esa sensación de saber que tú estabas allí y vivo en aquel rincón del tiempo y del mundo*»
Hunter S. Thompson

La música es parte fundamental de demoscene. Casi siempre la música en las demos es sintetizada, no grabada. Los algoritmos que se usan para sintetizar sonido también se basan en PCG, al igual que las gráficas. La música sintetizada no se limita a un conjunto estándar de instrumentos musicales. El compositor tiene la libertad de crear sus propios instrumentos para producir el sonido que desee.

Una característica particular de la música en demoscene es que las obras musicales tienen un valor artístico propio. Existen comunidades de melómanos apasionados por la música sintetizada. Muchos artistas reconocidos de música electrónica, como Bret Autrey (*Blue Stahli*), Sean Tyas, Erez Eisen,[32] Shiva Shidapu, entre otros, han iniciado sus carreras componiendo música *tracker*.

Originalmente los *tracker* o *secuenciadores* eran los nombres de las herramientas de software para componer y reproducir música sintetizada. Posteriormente el mismo nombre comenzó a ser aplicado a la música compuesta con estas herramientas. Entre las herramientas más reconocidas se destacan *Scream Tracker*, *FastTracker*, *Impulse Tracker*. Las primeras implementaciones de tracker fueron hechas para sistemas operativos como AmigaOS y MS-DOS, pero hoy en día existen tracker para todas las plataformas, inclusive para los teléfonos móviles.

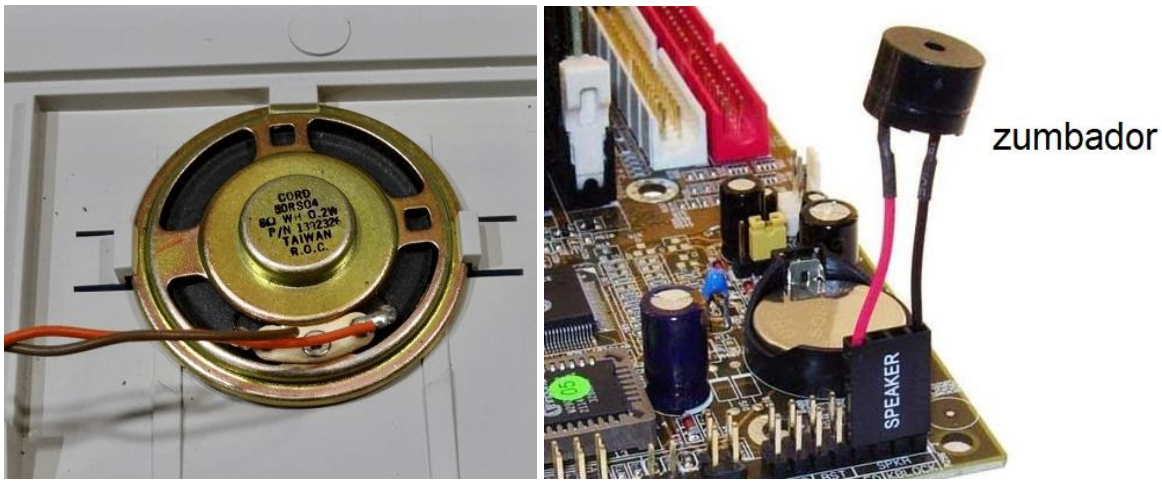
Actualmente las herramientas que se utilizan para componer y reproducir música de demoscene permiten generar obras de calidad profesional. Por eso, la comunidad de artistas musicales en demoscene es ampliamente reconocida. Una prueba de ello es la popularidad que tienen las colecciones de esta clase de música, como Mod Archive[33], donde se puede descargar centenares de miles de obras musicales gratuitas.

El tamaño de archivo de las obras musicales sintetizadas suele ser muy pequeño en comparación con las obras grabadas. El tamaño típico puede ser desde menos de 1Kb hasta algunas decenas de kilobytes. Por lo tanto, el uso de música tracker es común no sólo en demoscene, sino en otras aplicaciones donde el tamaño de archivo debe ser compacto. Por ejemplo, los juegos para plataformas móviles pueden utilizar este tipo de música. De hecho, el uso de música tracker también es común en los videojuegos para PC.

[32] Fundador del grupo israelí *Infected Mushroom*.
[33] https://modarchive.org

Música de 1 bit

Los primeros computadores no tenían tarjeta de sonido. Solamente incluían un dispositivo llamado *buzzer*[34] o zumbador. Este dispositivo emite un pitido con una frecuencia fija entre 100 y 2000 Hz para generar alarmas audibles. Aproximadamente hasta la década de 2000 el zumbador fue un componente estándar del PC. Los computadores actuales, especialmente portátiles, ya no incluyen este dispositivo. Por razones de compatibilidad, en algunos casos los pitidos del zumbador son emulados por medio del sistema de audio integrado.



Zumbador compacto conectado a la tarjeta madre (derecha) vs. el zumbador original de IBM.

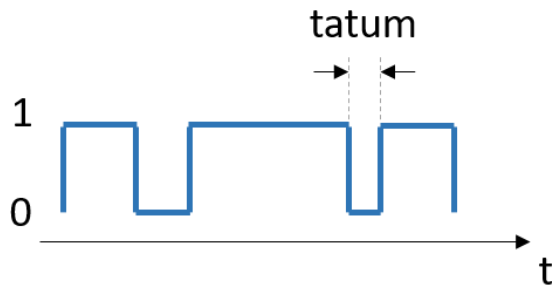
El sonido producido por el zumbador tiene una tonalidad uniforme, generalmente aguda. Sin embargo, se puede variar la duración del pitido. Entonces, al combinar pitidos cortos y largos se produce un sonido modulado. Esta forma de modulación por ancho de pulsos se llama **PWM** (por la sigla en inglés de *Pulse-Width Modulation*).

Los artistas *retro* aun componen música para el zumbador. Por ejemplo, Alex Semenov, mejor conocido como *Shiru*,[35] ha publicado varios álbumes usando únicamente el zumbador del PC. Este género musical se denomina **música de 1 bit**, ya que se trata de una secuencia de encendido y apagado del zumbador, como un código binario. Victor Adán, programador y Doctor en Artes Musicales, califica los dispositivos que reproducen música de 1 bit como los instrumentos musicales más limitados concebibles [Adan].

Dado que solamente se puede encender y apagar el zumbador, la amplitud del sonido no se puede variar. El único portador de información en este tipo de codificación es el tiempo (la duración). Además, los pitidos no pueden tener una separación y duración arbitrariamente cortas. Existe un tiempo mínimo, llamado *tatum*. El tiempo entre 2 pitidos consecutivos y la duración de cada pitido deben ser múltiplos enteros de este valor.

[34] También se conoce como *PC speaker* o *beeper*. En broma se le ha llamado *PC squeaker*.
[35] *Shiru's Stuff* https://shiru.untergrund.net/aboutme.shtml

La codificación de música de 1 bit se reduce a un tren de pulsos:



Tren de pulsos PWM

Este tren de pulsos activa y desactiva el zumbador eléctrico. De esta manera los 1 y 0 se convierten en variaciones de presión de aire, las cuales son percibidas como sonido.

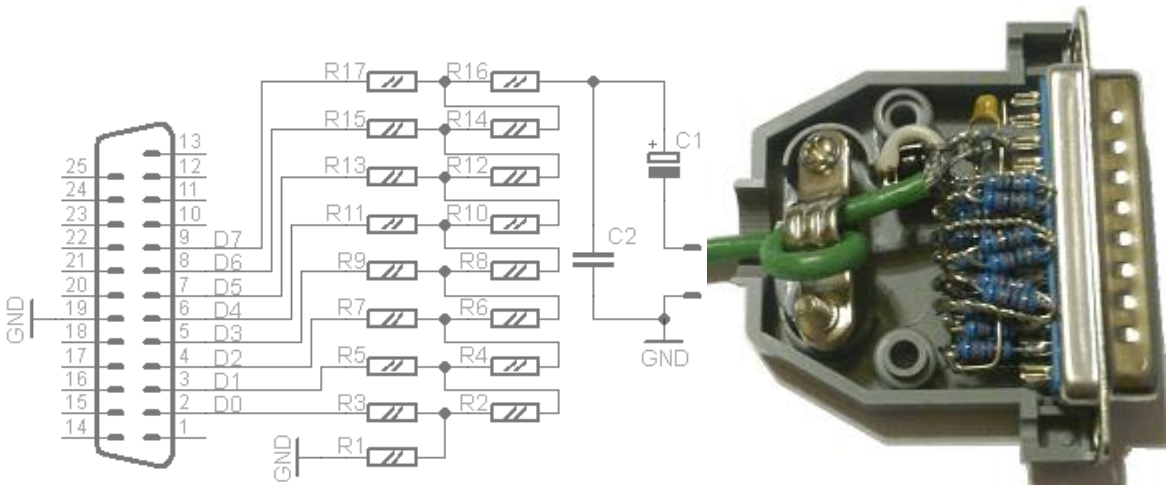
El sonido producido por un tren de pulsos de amplitud constante es intermitente, lo cual puede ser desagradable para el oído. Para «suavizar» estas intermitencias la música de 1 bit suele incluir sucesiones rápidas, en las cuales la frecuencia de los pulsos varía de manera uniforme (generalmente en sentido creciente). Este efecto musical se llama *arpegio*. El sonido que se produce de esta manera tiene afinidad con los instrumentos de cuerda. Por eso, el término *arpegio* se deriva del italiano *arpeggiare*: «tocar el arpa». El uso de arpegio en la música tracker actual sigue siendo común, aunque ya no tiene el propósito de suavizar las intermitencias propias de la música de 1 bit.

Música de 8 bits

Las primeras tarjetas de sonido eran básicamente conversores de digital a análogo (*DAC* por su sigla en inglés). Posteriormente se incluyeron funciones de filtrado, mezclador, entre otras, pero originalmente la única función de la tarjeta de sonido era la conversión de una señal digital, como el tren de pulsos presentado más arriba, en una señal analógica. Esta última podía ser amplificada y dirigida a un parlante. La diferencia esencial en comparación con el zumbador era la variación de la amplitud de los pulsos.

Covox fue uno de los primeros ejemplos de dispositivo de audio que hacía esta conversión. Fue creado a finales de los años 80. El nombre completo era *Covox Speech Thing*. Se trataba de un dispositivo externo, el cual se conectaba al puerto paralelo[36] (LPT). En el interior de Covox se encontraba un conversor de tipo **R-2R**, también conocido como escalera de resistencias. Debido a que su diseño era muy simple y el costo del dispositivo original era muy alto,[37] los aficionados a la electrónica comenzaron a fabricar clones artesanales de Covox.

[36] Este puerto se usaba principalmente para conectar la impresora.
[37] El costo de los primeros Covox llegaba hasta 80 USD. En cambio, las partes podían costar menos de 10 USD.



El diseño era tan simple que en los Covox artesanales los componentes aparecían soldados directamente sobre los pines del conector, sin utilizar circuito impreso.

La calidad del sonido dependía en gran medida de la precisión de los valores de resistencia en Ω. Además, la velocidad del puerto LPT no permitía producir pulsos de alta frecuencia para cubrir todo el espectro audible.[38] Por lo tanto, el sonido que se podía producir por medio de Covox era muy limitado en espectro y en calidad. Sin embargo, el sonido digital comenzó a hacer parte de demoscene gracias a la simplicidad de los primeros dispositivos de audio, como los clones de Covox. Una prueba de ello es que los primeros tracker para MS-DOS soportaban Covox. La popularidad de este dispositivo se mantuvo durante los años 90. En cambio, las tarjetas de sonido originales tardaron muchos años en conquistar el mercado debido a su alto costo.

Hoy en día se sigue usando el mismo diseño de Covox en diferentes mini-computadores y sistemas embebidos. Actualmente la disponibilidad de un puerto paralelo no es común. Por lo tanto, se usan conversores para puerto serie. El resto del dispositivo sigue siendo una escalera de resistencias.

También existieron versiones más avanzadas de Covox, en las cuales se podía reproducir sonido estéreo. Inicialmente eran 2 circuitos idénticos utilizando 2 puertos paralelos, uno para cada canal, pero posteriormente se implementó un multiplexor para compartir un único puerto. También han existido versiones con entrada de audio por micrófono y versiones internas, las cuales se instalaban en una de las ranuras ISA[39] de la tarjeta madre.

En el circuito ilustrado más arriba se puede apreciar que la entrada digital del conversor DAC utiliza 8 pines. Esto significa que la amplitud de cada muestra se codifica usando 8 bits. Por eso se dice que Covox y otros dispositivos similares producen **música de 8 bits**. Hoy en día se usa el mismo término para referirse a la música de tipo tracker, lo cual no siempre es correcto, ya que

[38] El puerto paralelo no limita la velocidad de transmisión, por lo que en teoría Covox podía funcionar para frecuencias más altas, pero la capacidad de los computadores de ese entonces no lo permitía.
[39] ISA era un bus de 8 o 16 bits disponible en las tarjetas madre de PC a partir de los años 80.

algunos secuenciadores como *FrastTracker II* pueden utilizar codificación de 16 bits. En cambio, el sonido de Covox, Game Boy y las primeras consolas, como NES y Sega, realmente era de 8 bits.

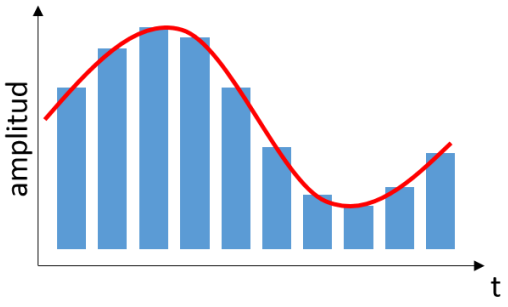
En la música de 8 bits los valores de amplitud varían entre 0 y 255 para un total de $2^8 = 256$ valores posibles. La codificación discreta de la amplitud puede tener diferentes formatos, entre los cuales el más utilizado es PCM.

PCM

La posibilidad de variar la amplitud es la base de la modulación **PAM** (por la sigla en inglés de *Pulse-Amplitude Modulation*). La implementación digital de la modulación PAM con muestreo se conoce como **PCM** (por la sigla en inglés de *Pulse-Code Modulation*).

PCM es el formato más usado hoy en día para codificar el audio digital. Los datos que recibe la tarjeta de sonido están en formato PCM. Este concepto no se debe confundir con el formato de archivo. El formato de archivo, como MP3 o Vorbis, se usa para almacenar o transmitir el audio. Las aplicaciones que reproducen estos archivos realizan internamente la conversión al formato PCM, el cual es entendido por la tarjeta de sonido. Por ejemplo, el formato de archivo WAV contiene un encabezado seguido por los datos PCM sin compresión. El formato PCM también es usado en los CD, DVD, Blu-ray, etc.

Las primeras tarjetas de sonido de uso masivo soportaban codificación PCM de 8 bits.[40] En el estándar de CD se usa el formato PCM de 16 bits. En cambio, el audio de DVD y Blu-ray puede ser de hasta 24 bits. A mayor número de bits[41] mayor es el número de niveles discretos de amplitud que pueden ser representados en este formato. Por lo tanto, la calidad del sonido mejora. Con 16 bits se puede representar $2^{16} = 65536$ niveles de amplitud, lo cual es 256 veces más que con 8 bits. De hecho, la modulación de 1 bit presentada más arriba también puede ser vista como un caso particular de PCM con 2 valores de amplitud únicamente [Adan].



PCM

[40] También existieron dispositivos de 4 bits, pero su uso no fue tan amplio.
[41] En digitalización, el número de bits de codificación también se conoce como la *resolución*.

Las barras verticales en la gráfica anterior representan las muestras (valores discretos) de amplitud. A diferencia de PWM, los espacios entre las muestras no son silencios. Estos espacios se incluyen solamente para diferenciar más claramente el muestreo.

El muestreo es discreto tanto en el sentido vertical (los valores de amplitud) como en el sentido horizontal (el tiempo). La calidad de la digitalización depende del número de bits, pero también de la frecuencia de muestreo. Por ejemplo, la frecuencia de muestreo estándar en CD es de 44.1 KHz, lo cual permite cubrir todo el rango audible entre 20 Hz y 20 KHz[42]. Esto significa que un segundo de audio corresponde a 44100 muestras. Si se usa el formato PCM de 16 bits con un solo canal (mono) cada muestra ocuparía 2 bytes. Entonces, tan solo un segundo de audio ocuparía 88200 bytes. Inclusive si reducimos el formato a 8 bits y utilizamos una frecuencia de muestreo menor, el tamaño seguirá siendo demasiado grande y un segundo sería demasiado corto para su uso práctico en demoscene. No olvidemos que las demos pueden ser de 64K, 32K, etc. Además, la mayor parte de ese tamaño es usado por las gráficas.

Por lo tanto, es evidente que el formato PCM no puede ser usado directamente en demoscene. Si comprimimos las muestras usando un formato como MP3 el tamaño se reduciría aproximadamente 10 veces, pero aun así seguiría siendo demasiado grande. Por eso, desde los inicios de demoscene se han utilizado secuenciadores o tracker, los cuales permiten generar audio de forma procedimental.

Tracker

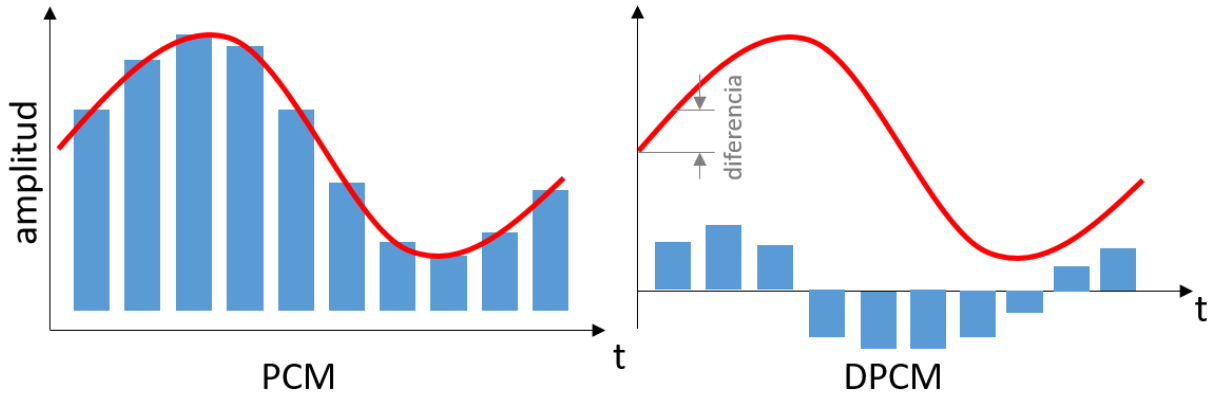
El secuanciador o tracker crea un conjunto virtual de instrumentos para interpretar una obra musical. Las obras de tipo tracker no son digitalizaciones de sonido grabado, como en el caso de WAV, MP3, etc. El concepto de tracker es más similar a MIDI, ya que su contenido son las partituras musicales. Sin embargo, en el caso de MIDI se utilizan instrumentos estándar soportados por el sintetizador, el cual generalmente hace parte de la tarjeta de sonido [Phillips]. Entre estos instrumentos se encuentran el piano, la guitarra acústica, etc. En cambio, los tracker utilizan sus propios instrumentos, los cuales no necesariamente corresponden con instrumentos musicales reales.

El nombre tracker se deriva de *track*, que significa pista en inglés. Los instrumentos se reproducen sobre múltiples pistas o canales de audio. Los instrumentos se componen de muestras o *samples*. Por ejemplo, el sonido de percusión de una batería o el claxon de un auto pueden ser samples. Estos samples son editados y combinados por medio del sintetizador integrado en el tracker.

Debido a que el formato PCM, presentado mas arriba, no es compacto, los samples se almacenan en un formato derivado para reducción de tamaño. Un formato comúnmente utilizado es conocido como **PCM Diferencial** o *DPCM*. En este formato se codifican las diferencias entre las

[42] La frecuencia de muestreo debe ser mayor que el doble de la frecuencia máxima de la señal analógica.

amplitudes consecutivas en lugar de las amplitudes mismas, como se muestra en la siguiente gráfica:



PCM vs. DPCM

Generalmente la magnitud de las variaciones de amplitud es mucho menor que las amplitudes, especialmente si la frecuencia de muestreo es alta. Por lo tanto, la codificación diferencial puede ser representada con menos bits. Por ejemplo, una muestra PCM de 16 bits podría ser codificada en formato DPCM de 8 bits, lo cual reduce a la mitad el tamaño de la muestra en bytes. Si la magnitud de las diferencias excede la resolución diferencial, entonces se produce una pérdida de calidad.

Para reconstruir el valor real de amplitud se realiza una suma aritmética de las diferencias. Entonces, el decodificador de DPCM a PCM es una simple operación de suma, como se puede ver en el siguiente ejemplo:

```
Dim dpcm(3) As Byte = { 10, 2, 3, -11 }  ' 8-bit DPCM
Dim pcm(3) As Short                      ' 16-bit PCM
Dim acc As Short = 0

For index As Integer = 0 To UBound(dpcm)
    acc = acc + dpcm(index)
    pcm(index) = acc
Next
```

Al ejecutar este código, los valores DPCM `{10, 2, 3, -11}` se convierten en PCM `{10, 12, 15, 4}`. En este ejemplo los valores DPCM son de 8 bits y los valores PCM son de 16 bits, pero en los diferentes formatos de música tracker el número de bits puede variar.

Otro formato comúnmente utilizado para compactar los samples es **DPCM Adaptativo** o *ADPCM*. Existen diferentes implementaciones de ADPCM, algunas de las cuales se usan en telefonía y se encuentran estandarizadas por ITU-T.[43] El formato usado en la música tracker consiste de una

[43] Estándar *G.722* para codificación de audio: https://www.itu.int/rec/T-REC-G.722

tabla con valores diferenciales fijos seguida por una secuencia DPCM, en la cual los valores codificados son índices hacia dicha tabla. El mismo principio es usado en los archivos de imágenes de mapa de bits, como BMP, en los cuales cada pixel es codificado como un índice hacia una paleta de colores. Entre más compacta sea la paleta de colores menor es el número de bits necesarios para codificar los índices. Por ejemplo, si la paleta contiene 16 colores, cada índice puede ser representado con 4 bits, ya que $2^4 = 16$. Lo mismo se aplica en ADPCM en música tracker. El siguiente ejemplo hace la conversión de ADPCM de 4 bits a PCM de 16 bits:

```
Dim deltas(15) As Byte =_
 { 0, 1, 2, 4, 8, 16, 32, 64, -1, -2, -4, -8, -16, -32, -48, -64 }
Dim adpcm(3) As Byte = { &hF0, &h2A, &h35, &hF7 } ' 4-bit ADPCM
Dim pcm(7) As Short                               ' 16-bit PCM
Dim acc As Short = 0, delta As UByte

For index As Integer = 0 To UBound(adpcm)
    delta = adpcm(index)
    acc = acc + deltas(delta And &hF)
    pcm(index Shr 1) = acc
    acc = acc + deltas(delta Shr 4)
    pcm((index Shr 1) + 1) = acc
Next
```

El contenido del arreglo `deltas` es un ejemplo típico de tabla con los 16 valores diferenciales más comunes, tanto positivos como negativos. Si se utiliza esta tabla estándar, su contenido puede ser omitido dentro del archivo para mayor ahorro de espacio. Dado que los índices hacia la tabla diferencial en este ejemplo son de 4 bits, pero el tipo de dato mínimo es el byte de 8 bits, es necesario utilizar operadores lógicos como `And` y `Shr` para separar un valor de 8 bits en 2 valores de 4 bits.

Aunque DPCM y ADPCM ofrecen un alto grado de compactación, algunos tracker soportan compresión MP3 o Vorbis para mayor reducción del tamaño de archivo. Por ejemplo, **OXM**, también conocido como *oggmod*, es un formato de archivo tracker, en el cual los samples utilizan compresión Vorbis.

El sonido que reproduce el tracker no es el sample PCM, sino una mezcla de diferentes samples, los cuales se utilizan para interpretar una partitura. La partitura es una secuencia de códigos, los cuales representan la notación musical: do, re, mi, fa, sol, la, si. Estos códigos también incluyen las notas sostenidas y se repiten para cada octava. Además, cada uno de estos códigos puede incluir un efecto musical específico y un control de volumen. El control de volumen sirve para resaltar o atenuar los instrumentos.

Entonces, el sonido de cada pista o canal en todo momento se calcula teniendo en cuenta nota, sample, volumen y efecto. Además, el tracker también debe mezclar las pistas, ya que las primeras herramientas de este tipo surgieron cuando las tarjetas de sonido aun no incluían mezclador.

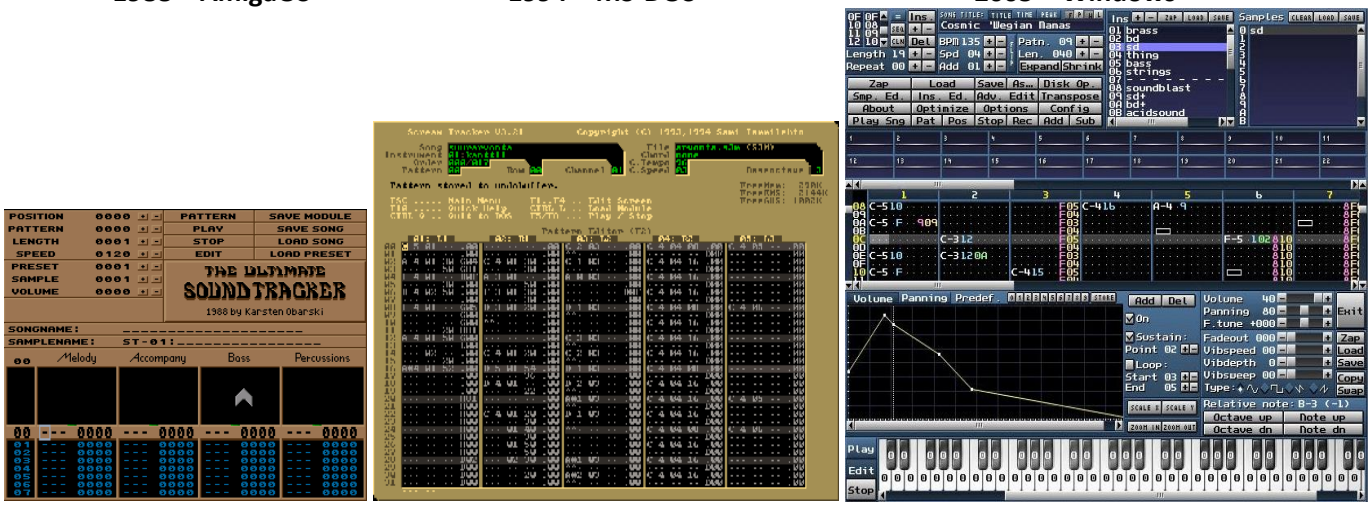
Módulos

Los módulos son una familia de archivos de audio de tipo tracker. Su nombre coincide con el formato original de esta familia, creado para los computadores Amiga entre 1987 y 1988. Los archivos en AmigaOS no utilizaban extensiones para identificar el tipo de archivo, pero al portar esta tecnología a MS-DOS y otros sistemas operativos, se asignó la extensión **MOD**. Este formato, el cual dio origen a toda la familia de módulos en la música tracker, fue creado por el compositor alemán Karsten «Obi» Obarski para su **Ultimate Soundtracker**. Este tracker, junto con su formato MOD, se popularizaron rápidamente, ya que permitían crear música de manera simple y en condiciones caceras.

Originalmente MOD soportaba 4 canales y hasta 15 instrumentos. Los samples se almacenaban sin compresión en formato PCM de 8 bits, el mismo formato usado por el hardware de Amiga. Esto se hizo para que la tarea de síntesis de audio generara una carga de procesamiento mínima, ya que los primeros computadores no tenían mayor capacidad de procesador. Posteriormente el formato se amplió para incluir un mayor número de canales e instrumentos.

Hoy en día el formato MOD se sigue utilizando en demoscene y videojuegos. La gran mayoría de los reproductores de música tracker soportan este formato, ya que es el más básico. Además, los tracker más recientes implementan los mismos conceptos fundamentales que tenía el antiguo Ultimate Soundtracker de Karsten Obarski. Inclusive la interfaz de usuario de todos los tracker es similar.

1988 – AmigaOS **1994 – MS-DOS** **2005 – Windows**



Evolución de las Interfaces de usuario: *Ultimate Soundtracker* (1988) vs. *Scream Tracker* (1994) vs. *MilkyTracker* (2005)

En 1994 surgen 2 nuevos formatos basados en MOD, a saber: S3M y XM. El formato **S3M** fue creado por el demo-grupo finés **Future Crew** para el **ScreamTracker 3**. Una novedad en este

formato es la posibilidad de mezclar música sintetizada y pregrabada. Además, soporta hasta 99 instrumentos y permite especificar la distribución de *paneo*[44] por defecto.

XM significa *e**X**tended **M**odule*. Este formato fue introducido por el demo-grupo **Triton** para el **FastTracker II**. El formato XM soporta samples de 16 bits, DPCM, repeticiones, multi-muestreo. La funcionalidad de multi-muestreo permite incluir múltiples samples de un mismo instrumento, cubriendo notas distintas. Esto es particularmente útil cuando la obra contiene secuencias altas y bajas (agudos y graves). En algunos instrumentos como el piano el sonido varía no sólo dependiendo de la nota, sino también dependiendo de la intensidad de la pulsación de las teclas. Entonces, los samples pueden representar pulsaciones fuertes, medias, ligeras, etc. Los sintetizadores actuales emplean esta misma técnica de multi-muestreo. Además, en XM se ampliaron los efectos. Posteriormente se introdujo soporte para ADPCM.

Otro formato de módulo comúnmente utilizado es **IT**, el cual fue creado en 1995 para el **Impulse Tracker**. Originalmente este tracker para MS-DOS era gratuito, aunque de código fuente cerrado, pero en 2014 el código fuente fue liberado por su autor, Jeffrey Lim, bajo una licencia BSD. El formato IT puede ser convertido sin pérdidas a la mayoría de otros formatos, como XM, ya que sus funcionalidades son equivalentes.

También se puede destacar el formato de módulo **V2M**, como uno de los más avanzados. El formato V2M fue creado por el demo-grupo **Farbrausch** a comienzos de los años 2000. Aunque este formato incluye más funcionalidades y efectos que sus predecesores, no ha logrado desplazar a los formatos más tradicionales como XM.

Además, existen muchos otros formatos de módulos tracker: AHX/THX, AMF, AMS, DBM, DIGI, DMF, DSM, FAR, GDM, HVL, IMF, J2B, MDL, MED, MPTM, MO3, MT2, MTM, OCT, OKT, PLM, PSM, PTM, STM, ULT, STP, etc. Inclusive algunos de estos formatos tienen sub-formatos y versiones no oficiales. Esta gran variedad tiene que ver con la costumbre de adaptar las herramientas para uso interno, la cual es común en los demo-grupos.

Algunos demo-grupos crean su propio tracker con su propio formato de módulo. De hecho, frecuentemente la base de estos desarrollos es la ingeniería inversa. Inclusive el formato original MOD no era un formato abierto, pero fue *hackeado* en 1988 para conocer su estructura y replicarla en los formatos nuevos y mejorados. Lo más sorprendente es que fueron varios demo-grupos los que *hackearon* el formato original de manera independiente. El resultado de este *hackeo* masivo fue la aparición de múltiples clones de MOD, los cuales eran muy similares, pero incompatibles entre sí. Esta es otra de las razones, por las que demoscene se popularizó primero en los países con una tradición de hacking [Svensson].

No sería práctico incluir un tracker completo dentro de una demo para reproducir un único módulo. Existen librerías o codecs que sirven específicamente para reproducir módulos de la

[44] Paneo o *panning* es la distribución del sonido por medio de 2 o más canales (estéreo o multi-canal) de manera controlada. Por ejemplo, el oído puede percibir que el sonido proviene principalmente de un canal que de otro.

misma manera como lo hacen los tracker, pero sin la opción de editar el contenido. Estas librerías son usadas en demoscene, videojuegos, reproductores multimedia, etc. A continuación se enumeran algunas de estas librerías, incluyendo únicamente referencias gratuitas y de código abierto:

Módulos	Librería	Plataformas
V2M	**libV2**[45] de Farbrausch, el mismo demo-grupo que creó el formato V2M. Existen adaptaciones de libV2 para diferentes lenguajes de programación y otras plataformas. Originalmente esta librería no era de código fuente abierto, pero actualmente lo es.	Windows
MOD, XM, S3M	**IBXM** y **micromod**[46] de Martin Cameron. IBXM es una librería Java, una de las más antiguas, mientras que micromod es la versión nativa y web. La versión Java es compatible inclusive con plataformas antiguas como J2ME. La versión nativa incluye ejemplos para lenguajes C y Pascal.	multiplataforma, web, Java
MOD, XM, S3M, IT	**OZMod**[47] de Igor Kravtchenko funciona en una gran variedad de dispositivos que soportan Java, incluyendo Android.	Java
XM	**µFMOD**[48] o **uFMOD**, una librería multiplataforma en lenguaje ensamblador. Incluye ejemplos para diferentes lenguajes de programación. Según el portal Democoder.ru, µFMOD es la librería para reproducir música tracker más compacta que existe.[49]	Windows, Linux, BSD, KolibriOS
	libxm[50] de Romain «Artefact2» Dalmaso es una librería compacta en lenguaje C. El código fuente se puede compilar para diferentes plataformas, como Windows y Linux. Incluye versión web en JavaScript, ejemplos de uso y documentación detallada.	multiplataforma, web

[45] *Repositorio de libV2*. https://github.com/farbrausch/fr_public/tree/master/altona_wz4/wz4/wz4player/libv2
[46] *Repositorio de IBXM y micromod*. https://github.com/martincameron/micromod
[47] *Sitio oficial de OZMod*. http://web.archive.org/web/20160314034847/http://www.tsarevitch.org/ozmod/
[48] *Sitio oficial de µFMOD*. https://ufmod.sourceforge.io/ind_es.htm
[49] *Librerías en el portal Democoder.ru*. https://web.archive.org/web/20140214154448/http://democoder.ru/libs/
[50] *Repositorio de libxm*. https://github.com/Artefact2/libxm

Además, existen librerías universales como **libopenmpt**[51] y **libMIkMod**,[52] las cuales se usan en videojuegos y otras aplicaciones. En cambio, su uso en demoscene no es común debido a su gran tamaño. También existen librerías comerciales.

Generalmente las librerías más compactas reproducen un único formato de módulo. Sin embargo, existen herramientas de conversión de un formato de módulo a otro. Dado que los diferentes formatos de módulo tienen funcionalidades distintas, esta conversión de formato puede producir una pérdida de calidad e introducir errores. Por ejemplo, Open ModPlug Tracker[53] es un tracker gratuito para Windows, el cual soporta múltiples formatos de módulos y permite hacer la conversión de un formato a otro. La librería libopenmpt hace parte de este mismo tracker.

[51] https://lib.openmpt.org/libopenmpt/
[52] http://mikmod.sourceforge.net/
[53] *Sitio oficial del tracker OpenMPT*. https://openmpt.org/

Optimización y compactación

«*Compactibilidad es la capacidad de poner un contenido extenso en cosas de pequeño volumen*»
Antón Chejov

Una de las características más representativas de demoscene es el reducido tamaño en disco de las obras. Para compactar al máximo el ejecutable se utiliza una combinación de técnicas manuales y herramientas de compresión.

Muchos de los primeros PC se usaban sin disco duro interno. Tanto el sistema operativo como los programas se almacenaban en diskettes magnéticos. Los diskettes más comunes eran de 5¼ pulgadas de 360 Kb y los de 3½ pulgadas de 1.44 Mb.

Diskettes de 3½ pulgadas de 1.44 Mb.

Debido a la escasez de estos insumos y su alto costo, al menos hasta los años 90, era común que en un mismo diskette se almacenara la mayor cantidad posible de programas y archivos. Los aficionados a la computación tenían al menos un diskette «universal», en el cual se encontraba el sistema operativo MS-DOS, herramientas de diagnóstico de hardware, antivirus y otras herramientas y utilerías. Si aún sobraba espacio, entonces también se almacenaban juegos. Esta idea de aprovechar al máximo el espacio disponible también fue aplicada por las compañías de software, quienes distribuían sus productos en diskettes. El espacio disponible se rellenaba con versiones de prueba de otros productos, manuales, etc.

Hoy en día la necesidad de optimizar y compactar los contenidos aún existe en desarrollo web. Por ejemplo, si una página web tiene un peso excesivo, puede tardar mucho en desplegarse, lo cual afecta la experiencia de usuario y la visibilidad de la página en los motores de búsqueda. La optimización de tamaño también es común en desarrollo de apps para plataformas móviles, ya que los dispositivos móviles, especialmente de gama baja, pueden tener recursos de hardware muy limitado. Además, la reducción de tamaño ahorra el uso de tráfico de datos en la descarga del contenido.

Seguramente los límites de tamaño en demoscene se basan en la costumbre de aprovechar al máximo el espacio disponible. También existen razones más específicas. Por ejemplo, el direccionamiento de memoria en MS-DOS se hacía de manera segmentada: se usaba un selector de segmento combinado con un puntero de 16 bits. Esto se conoce como modelo de memoria y no se debe confundir con la arquitectura de CPU [Chen]. El formato ejecutable COM estaba limitado a un único segmento. Por lo tanto, su tamaño máximo era 2^{16} = 64Kb. En realidad, el tamaño máximo era un poco menor que 64Kb, ya que el rango de memoria desde 0x0000 hasta 0x00FF era reservado para uso del sistema. Entonces, el tamaño máximo del ejecutable COM era 64Kb menos este rango reservado de 256 bytes, lo cual nos da como resultado 65280 bytes. Por lo tanto, la categoría más grande en las competencias de demoscene en los tiempos de MS-DOS era de 64K. Los formatos de ejecutable más actuales, como MS-COFF en Windows, ya no tienen estos límites de tamaño, pero los límites se mantienen en demoscene para que las competencias sean aún más desafiantes.

Compresión

Aun con el uso de PCG para generar las gráficas, animaciones y música, el tamaño de la demo puede ser demasiado grande. La manera más directa de reducirlo es la compresión. Si la demo es un archivo ejecutable, entonces la descompresión debe ser hecha automáticamente por el mismo ejecutable. Esto significa que el contenido comprimido, junto con el código necesario para descomprimirlo, se incluyen dentro del mismo archivo ejecutable. Al ejecutar este archivo, la demo se auto-descomprime en memoria y comienza a proyectar los contenidos audiovisuales.

A veces, se usan múltiples niveles de compresión. Por ejemplo, se puede hacer la analogía con las herramientas de compresión como WinRAR y 7zip. A veces, al comprimir un archivo con WinRAR y luego volver a comprimirlo con 7zip, se logra un tamaño menor que al comprimirlo directamente con 7zip. En demoscene también se usan combinaciones de algoritmos de compresión para reducir el tamaño al mínimo posible.

La compresión de cualquier información, incluyendo los datos y código de una demo, es un proceso que busca optimizar el tamaño. Este proceso se puede explicar con basa en la *entropía*. Supongamos que los datos de entrada son esta cadena de texto:

```
magia
```

Para calcular la entropía contemos la ocurrencia de cada símbolo:

```
m: 1
a: 2
g: 1
i: 1
```

La longitud de la cadena son 5 símbolos. Entonces, la frecuencia de ocurrencia para cada símbolo se calcula como sigue:

```
m: 1/5 = 0.2
a: 2/5 = 0.4
g: 1/5 = 0.2
i: 1/5 = 0.2
```

La entropía se calcula con el valor absoluto del logaritmo en base 2 de la frecuencia:

$$m: |\log_2(1/5)| = 2.322$$
$$a: |\log_2(2/5)| = 1.322$$
$$g: |\log_2(1/5)| = 2.322$$
$$i: |\log_2(1/5)| = 2.322$$

Al sumar los valores de entropía obtenemos:

2.322 + 1.322 + 2.322 + 2.322 = 8.288 bits

La suma es el número de bits que teóricamente serían suficientes para almacenar los datos de entrada. En este caso serían 9 bits, ya que el número debe ser entero. Si utilizamos codificación ASCII cada símbolo ocupa 8 bits. Entonces, la cadena de texto completa ocupa $8 \cdot 5 = 40$ bits. La relación entre el tamaño real y el tamaño teórico sería 40 / 9 = 4.44. Este valor es una medida de efectividad (o inefectividad) en el almacenamiento. El valor de 4.44 indica que un algoritmo de compresión ideal podría compactar los datos al menos 4 veces.

El cálculo de entropía también se usa para determinar si los datos están comprimidos o cifrados [Wasm].

Con base en el ejemplo anterior podemos concluir que la compactibilidad de los datos mejora si aumenta la frecuencia de ocurrencia de los símbolos. Por eso, una cadena de bytes con repeticiones de los mismos valores se comprime mejor que una cadena con valores únicos. Este hecho es usado ampliamente en demoscene. Por eso los demosceners prefieren usar lenguajes de bajo nivel como ensamblador. Esto les permite tener control sobre el código máquina generado. Entonces, el código se puede reorganizar y adaptar no sólo para que sea más compacto, sino también para que sea más compactible durante la compresión subsiguiente.

Existen muchas herramientas de compresión y cifrado para ejecutables, mejor conocidas como empaquetadores, algunas de las cuales son gratuitas y de código fuente abierto. Por ejemplo, *UPX*[54] es uno de los empaquetadores más antiguos y mejor conocidos. Este empaquetador es multiplataforma y completamente abierto y gratuito. Los empaquetadores se usan no solamente

[54] https://upx.github.io/

para reducir el tamaño del ejecutable. También sirven para proteger el código contra la ingeniería inversa [Wasm].

Los demo-grupos suelen desarrollar sus propios empaquetadores en lugar de usar los que ya existen. De esta manera pueden tener mayor control sobre la compresión de las demos para lograr un tamaño menor aún. Por ejemplo, el demo-grupo **Farbrausch** creó su propio empaquetador basado en UPX y combinado con el algoritmo de compresión *aPACK*[55] [Jagdmann]. Este empaquetador evolucionó en un producto gratuito y abierto: *kkrunchy*.[56] El mismo demo-grupo creó un motor gráfico y un generador de modelos 3D. El uso de herramientas endógenas (creadas por el mismo demo-grupo) es una práctica común en demoscene. Esto se debe a la necesidad de contar con funcionalidades originales que no existen en las herramientas disponibles.

Código no utilizado

Los compiladores modernos tienen la capacidad de identificar fragmentos de código, variables, librerías u otras entidades del programa que nunca se utilizan. Esto se conoce como **código muerto** o **código no utilizado**. El compilador puede omitir incluirlo en el código ejecutable final para efectos de optimización.

Sin embargo, la identificación de código no utilizado no siempre es posible de forma automática. Por ejemplo, supongamos que la demo incluye un módulo de síntesis de voz. Este módulo se usa para reproducir los textos embebidos dentro de un archivo de música tracker incluido en la misma demo. Supongamos que el archivo de música finalmente incluido en la demo no tiene ningún texto que deba ser reproducido. El compilador no tendía la capacidad de analizar el contenido del archivo tracker para identificar que el módulo de síntesis de voz nunca se va a utilizar, ya que el compilador no conoce el formato de ese archivo. En cambio, el autor de la demo, quien sí conoce el formato, puede hacer este análisis y determinar que una parte del código puede ser eliminada. Por lo tanto, la optimización manual del código puede ser más efectiva. Además, si todo o parte del código está escrito en lenguaje ensamblador, la optimización es tarea del programador, no del compilador.

Un ejemplo de optimización por eliminación de código no utilizado se puede encontrar en la herramienta *Eff*.[57] Esta herramienta se incluye junto con la librería *µFMOD* para reproducir música tracker en formato XM. La herramienta *Eff* sirve para optimizar el código de la librería *µFMOD* para reproducir un archivo XM específico. Para esto, la herramienta analiza el contenido del archivo XM para identificar los efectos musicales utilizados, a saber: *vibrato*, *arpeggio*, *tremolo*, *portamento*, etc. La gran mayoría de las composiciones tracker utilizan solamente

[55] https://www.ibsensoftware.com/products_aPACK.html
[56] http://www.farbrausch.de/~fg/kkrunchy/
[57] https://ufmod.sourceforge.io/Win32/es.htm#22

algunos efectos, no todos. Entonces, *Eff* genera un *archivo de cabecera*,[58] el cual se utiliza para recompilar la librería *µFMOD* eliminando las implementaciones de los efectos musicales y funcionalidades que no son necesarias para reproducir el archivo XM específico. De esta manera el tamaño del código compilado se reduce, pero no se garantiza que el mismo código pueda reproducir correctamente una composición XM distinta. Esto puede ser considerado como una optimización agresiva[59] y contraria a las buenas prácticas de programación, ya que el correcto funcionamiento del programa se mantiene solamente bajo ciertas condiciones de uso.

Sin embargo, en demoscene todas las formas de optimización son bienvenidas. El uso de métodos poco ortodoxos es parte de la costumbre de romper las reglas. Lo más importante es que la demo funcione correctamente durante su presentación ante el público y el tamaño de archivo no supere el límite predefinido.

Micro-optimización

La optimización y la compresión del código son procesos independientes. Con base en el cálculo de entropía podemos estimar qué tan compactible es el código ejecutable. Entonces, el tamaño comprimido converge a un valor que se puede estimar matemáticamente. En cambio, la optimización no tiene un límite estimable. Por eso, la optimización manual se puede convertir en una obsesión. Los últimos días antes de la demo-party son una «carrera contrarreloj» para exprimir hasta el último byte y lograr que el tamaño de la demo quede dentro del límite contemplado.

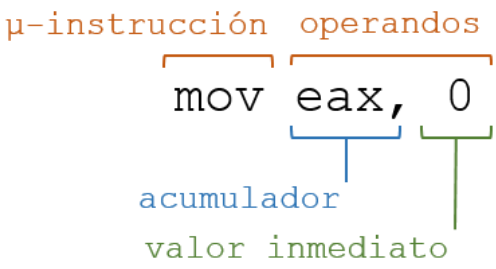
La micro-optimización es como la preparación de los boxeadores para el pesaje. La última semana de preparación está llena de sacrificios enormes para los atletas, quienes reducen drásticamente el consumo de alimentos y se someten a un desgaste físico extremo con tal de bajar hasta el último gramo de peso. En demoscene este proceso es más intelectual que físico, aunque la pérdida de calorías puede ser comparable en ambos escenarios.

Micro-optimización es el proceso de reorganización o sustitución de código, conservando la funcionalidad original, en busca de pequeñas mejoras en términos de tamaño y/o desempeño. En algunos casos el objeto de la optimización puede ser una única micro-instrucción[60] o un conjunto de micro-instrucciones en código máquina.

[58] Un archivo de cabecera o archivo de inclusión contiene código fuente, el cual se incluye dentro de otro archivo de código fuente. Esta inclusión se puede utilizar para compilar el código de manera condicionada o personalizada.
[59] La optimización es calificada como agresiva si el resultado de la misma puede producir errores en el funcionamiento del programa.
[60] Una micro-instrucción es la operación básica que realiza el procesador durante uno o más ciclos de ejecución.

Inicialización de un registro en cero

A continuación se presenta un ejemplo de micro-optimización de una única micro-instrucción de CPU. Supongamos que deseamos cargar el valor cero en el registro acumulador.[61] Muchos algoritmos en código máquina incluyen este tipo de inicialización. Entonces, cualquier ejemplo de demo suele incluir un código como este. El mnemónico[62] de esta operación en la arquitectura x86[63] con la sintaxis de Intel es el siguiente:



En código máquina de esta instrucción son los siguientes 5 bytes:

```
B8 00 00 00 00
```

`B8` es el *opcode*[64] de la operación MOV. Los 4 bytes subsiguientes contienen el cero como un valor inmediato de 32 bits.

Una alternativa más compacta para cargar el valor cero en el acumulador es la operación lógica OR-exclusivo (XOR):

```
xor eax, eax
```

El resultado siempre es cero, sin importar el valor original que tiene el acumulador, ya que para todos los bits del registro se aplica la siguiente tabla de verdad, donde $\oplus$ representa la operación XOR:

$0 \oplus 0 = 0$
$1 \oplus 1 = 0$

El equivalente en código máquina ocupa tan solo 2 bytes:

```
33 C0
```

---

[61] El registro acumulador se usa para almacenar los operandos y resultados de las operaciones aritméticas y lógicas. También puede ser usado como un registro de propósito general.
[62] Mnemónico es una representación simbólica de una micro-instrucción en lenguaje ensamblador. Los mnemónicos facilitan la comprensión del código máquina al representarlo con palabras en lugar de valores binarios. El ensamblador convierte los mnemónicos en código máquina.
[63] x86 es una familia de procesadores de 16 y 32 bits creada por Intel. Generalmente este término hace referencia a los procesadores de 32 bits, como la familia Intel Pentium, AMD Athlon, entre otros.
[64] Opcode es una parte de la micro-instrucción en lenguaje máquina, la cual indica el código de operación.

En este caso no se incluye un valor inmediato, ya que la operación XOR se aplica al valor actual del registro acumulador contra el mismo valor. Por eso el código máquina es más corto. Entonces, al reemplazar la micro-instrucción MOV por la micro-instrucción XOR el tamaño del código se reduce de 5 a 2 bytes.

Sin embargo, las operaciones de estas micro-instrucciones MOV y XOR no son exactamente idénticas. XOR es una micro-instrucción de tipo aritmético-lógico. Por lo tanto, esta micro-instrucción altera el registro de estado.[65] Entonces, si la carga del valor cero en el acumulador se encuentra en medio de la evaluación de un salto condicional, este salto dejaría de ser condicional. Los saltos condicionales son la base de la lógica de tipo if/else en los lenguajes de alto nivel. Una micro-instrucción aritmético-lógica como XOR afecta el funcionamiento de estas condicionales en caso de que se encuentre entre la evaluación de la condición y el salto respectivo. Este ejemplo demuestra que la micro-optimización, inclusive en el nivel más bajo posible, puede ocasionar errores.

Punto medio

Un ejemplo típico de cálculo aritmético, el cual se usa ampliamente en las gráficas PCG, es el punto medio entre 2 puntos. El cálculo del punto medio sirve para centrar una gráfica dentro de otra, calcular promedios, aplicar efectos simétricos, etc.

Supongamos que tenemos 2 puntos con coordenadas cartesianas (x_0, y_0) y (x_1, y_1). El punto medio o equidistante tendría las coordenadas (x_m, y_m) que se calculan como sigue:

$$x_m = \frac{x_o + x_1}{2} \qquad y_m = \frac{y_o + y_1}{2}$$

Entonces, cada coordenada se calcula por medio de una suma seguida por una división entre 2. En FreeBASIC este código sería como sigue, utilizando aritmética de enteros:

```
Function med(p0 As Integer, p1 As Integer) As Integer
    return (p0 + p1) \ 2
End Function
```

Al compilar esta función obtenemos el siguiente código máquina de tamaño 14 bytes:

```
8B 45 0C                mov     eax, [p1]
03 45 08                add     eax, [p0]
B9 02 00 00 00          mov     ecx, 2
99                      cdq
F7 F9                   idiv    ecx
```

[65] El registro de estado o registro de banderas (EFLAGS en x86) contiene los bits de estado de la última operación aritmético-lógica y el estado actual de CPU. Sirve para identificar si un valor es igual a cero, el signo, paridad, etc.

Para mayor simplicidad se omite el código de inicio y fin de la función, dejando únicamente el cálculo aritmético. La primera micro-instrucción carga el valor de la coordenada p_1 en el registro acumulador. La micro-instrucción siguiente le suma el valor de la coordenada p_0. Luego se inicializa el registro contador con el valor 2. Luego el valor en el registro acumulador, el cual contiene la suma de las coordenadas, se amplía a 64 bits con la micro-instrucción CDQ y se divide entre el valor del registro contador con la micro-instrucción IDIV.

La división entera entre 2^n equivale al corrimiento a la derecha de *n* bits. La multiplicación entera por 2^n equivale al corrimiento a la izquierda de *n* bits. Entonces, la división entre 2 puede ser sustituida por el corrimiento a la derecha de 1 bit, como sigue:

```
Function med(p0 As Integer, p1 As Integer) As Integer
    return (p0 + p1) Shr 1
End Function
```

Al compilar esta función obtenemos un código máquina de 8 bytes:

```
8B 45 0C                mov     eax, [p1]
03 45 08                add     eax, [p0]
D1 F8                   sar     eax, 1
```

La versión con el corrimiento de bits es más compacta, ocupa únicamente el registro acumulador, tiene 3 micro-instrucciones en lugar de 5. De hecho, algunos compiladores de lenguajes de alto nivel realizan esta micro-optimización de manera automática.

Sin embargo, al igual que en el ejemplo anterior, en este caso también existe una diferencia, la cual puede ocasionar errores. Esta diferencia se presenta solamente con los números negativos. Las coordenadas para el cálculo del punto medio podrían ser negativas, especialmente si son relativas. En este caso los resultados que arrojan los operadores `\` y `Shr` pueden ser diferentes. El operador `Shr` redondea el resultado hacia el mayor valor entero que sea igual o menor que el resultado real. El operador `\` hace lo mismo para los valores positivos. En cambio, para los negativos, el operador `\` redondea el resultado hacia el mayor valor entero que sea igual o mayor que el resultado real. Por ejemplo:

```
 5 \ 2   = 2
 5 Shr 1 = 2
-5 \ 2   = -2
-5 Shr 1 = -3
```

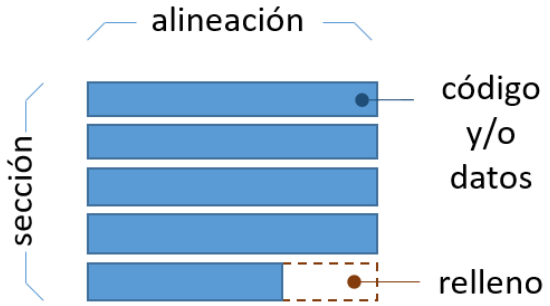
En un caso el valor real -2.5 se aproxima a -2, mientras que en otro caso el mismo valor se aproxima a -3. Si esta operación se realiza sobre las coordenadas de pixeles de una imagen la diferencia de una unidad puede ser imperceptible. Sin embargo, no olvidemos que los algoritmos PCG suelen realizar muchas iteraciones. Si el error se replica en cada iteración el resultado puede ser completamente distinto al esperado.

Alineación vs. tamaño del ejecutable

Reducir el tamaño del código ejecutable en 1 byte no hace que el tamaño del archivo se reduzca en 1 byte. Esto tiene que ver con la estructura del ejecutable, compuesta de secciones, cuyo tamaño es múltiplo de un valor fijo conocido como alineación. El espacio de relleno, producto de la alineación, ocupa espacio de almacenamiento. El espacio de relleno se reserva al final de la sección para que el tamaño de la misma sea un múltiplo exacto de la alineación:

$$(T + R)\ mod\ A = 0$$

Donde *T* es el tamaño utilizado, *R* es el relleno, *A* es la alineación y *mod* es la operación módulo, la cual calcula el residuo de la división. Gráficamente:



Espacio de relleno al final de la sección del ejecutable

Una forma de calcular el relleno *R* en función del tamaño *T* y la alineación *A*, asumiendo que el valor de *A* es una potencia de 2:

$$R = \Big((T + A - 1)\ and\ \big(not\ (A - 1)\big)\Big) - T$$

Si se utiliza el formato **complemento a dos**[66] para representar números negativos, entonces *not(A – 1)* es lo mismo que *–A* y la fórmula se puede simplificar como sigue [Warren]:

$$R = \big((T + A - 1)\ and\ (-A)\big) - T$$

El valor de *A – 1* se suma al tamaño *T* para que el redondeo con la operación lógica AND arroje un resultado igual o mayor que el valor de *T*.

El valor de alineación siempre es una potencia de 2, tanto en Windows como en Linux. Esto facilita los cálculos necesarios para localizar una dirección en memoria. Por eso la fórmula anterior contiene solamente operaciones simples como AND, NOT, suma y resta. Si el valor de alineación

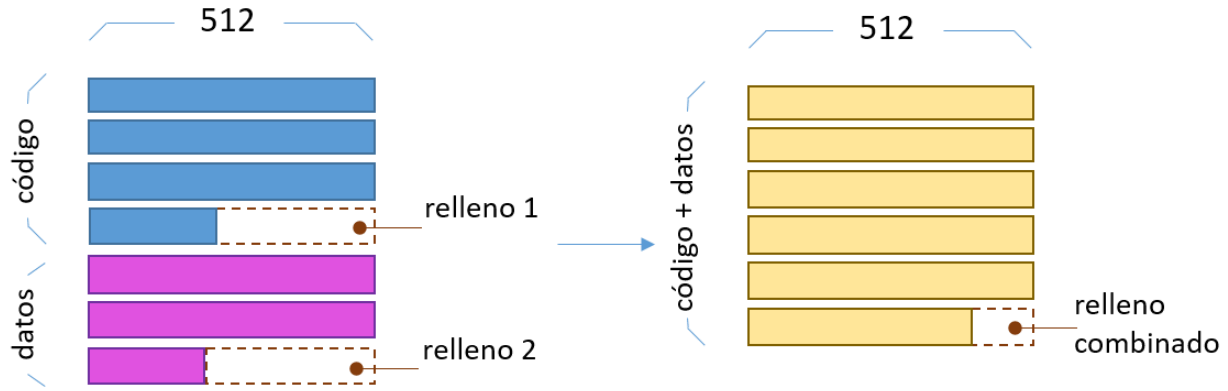
[66] En el formato *complemento a dos* un número negativo se representa restando 1 del valor sin signo e invirtiendo todos los bits del resultado. Este formato es usado en la gran mayoría de procesadores actuales.

no fuera una potencia de 2, sería necesario utilizar operaciones como división y módulo. Generalmente estas operaciones son más costosas para la CPU [Warren].

El formato de ejecutable en Windows se llama PE-COFF. El equivalente en Linux es el formato ELF. Ambos formatos tienen una estructura de secciones con alineación. En el caso de Windows el valor mínimo de alineación son 512 bytes, como se había mencionado anteriormente [Wasm]. Entonces, para que el tamaño del ejecutable se reduzca puede ser necesario eliminar centenares de bytes de código, ya que el espacio de relleno puede ser de hasta *A – 1* bytes. Esta es otra de las razones, por las cuales la micro-optimización no produce resultados proporcionales al esfuerzo aplicado.

Una técnica para reducir el tamaño del ejecutable y el espacio de relleno consiste en combinar secciones. Por ejemplo, en lugar de almacenar el código y los datos en secciones separadas, como hacen los compiladores de lenguajes de alto nivel, se puede almacenar todo en una única sección. Al reducir el número de secciones también se puede reducir el espacio no utilizado.

Supongamos que el valor de alineación es 512 y tenemos 2 secciones: una sección de código de tamaño 1720 bytes y otra sección de datos de tamaño 1200 bytes. Utilizando la fórmula anterior, calculamos que el espacio de relleno para la primera sección sería 328 bytes. Este es el valor que se debe sumar a 1720 para completar un múltiplo de 512. De la misma manera calculamos que el espacio de relleno para la segunda sección sería 336 bytes. La suma de ambos espacios de relleno sería 328 + 336 = 664 bytes. Si la suma de los espacios de relleno es mayor o igual que el valor de alineación, como en este caso, entonces la combinación de las secciones genera una reducción en el espacio de relleno. Al combinar ambas secciones el tamaño utilizado sería 1720 + 1200 = 2920 bytes. El espacio de relleno combinado, calculado con la misma fórmula, se reduce a 152 bytes. El tamaño de la sección combinada resulta ser menor que la suma de las secciones separadas, gracias a la reducción del espacio de relleno:



Combinación de secciones para reducir el espacio de relleno

Estamos asumiendo que los contenidos de las secciones pueden ser concatenados uno tras otro, lo cual no siempre es cierto. A veces, se debe agregar un espacio mínimo de relleno, pero este espacio puede ser mucho menor que la alineación entre secciones.

Además, hay que considerar que las secciones tienen permisos y estos permisos también deben ser combinados. Por ejemplo, la sección que contiene el código ejecutable tiene permiso de ejecución, pero normalmente no tiene permiso para escribir. En cambio, la sección de datos normalmente tiene permiso para escribir, pero no tiene permiso de ejecución. Entonces, la sección combinada debe tener ambos permisos. Por lo tanto, la técnica de combinación de secciones se debe aplicar con cuidado. Aunque esta tarea es hecha por el enlazador,[67] es responsabilidad del programador analizar los impactos.

Las secciones incluyen encabezados donde se especifican los permisos, el nombre, entre otros metadatos. Por lo tanto, al combinar secciones no sólo se reduce el espacio de relleno. También se reduce el espacio ocupado por los encabezados.

Generalmente, en PE-COFF el código ejecutable es almacenado en una sección de nombre «`.text`»[68] y los datos de sólo lectura se almacenan en una sección de nombre «`.rdata`». Los nombres de las secciones pueden variar dependiendo del compilador. El *cargador de programas*[69] no utiliza estos nombres. Los contenidos de las secciones son procesados por el cargador con base en los permisos y otros metadatos. Para combinar estas 2 secciones al momento de generar el archivo ejecutable utilizando el enlazador LINK.EXE[70] o POLINK.EXE[71] se agrega el siguiente argumento en la línea de comandos:

```
/MERGE:.rdata=.text
```

En la mayoría de los casos es seguro combinar al menos estas 2 secciones, ya que los permisos son compatibles.

En cambio, el formato ELF, el cual se utiliza en Linux y otros sistemas operativos, puede tener valores de alineación muy cortos. Un ejecutable ELF completo puede ocupar menos de 100 bytes [Raiter].

No solamente los tamaños de las secciones de código y/o datos deben estar alineados. El formato de archivo ejecutable incluye otras estructuras, las cuales también requieren alineación. Por ejemplo, la tabla de importación se usa para invocar funciones externas, las cuales se encuentran en las librerías dinámicas DLL. Si la demo utiliza OpenGL para generar las gráficas, entonces la tabla de importación puede contener los nombres de las funciones de OpenGL. La tabla de importación también utiliza alineación en su estructura.

---

[67] El enlazador o *linker* es una herramienta que combina el código objeto generado por el compilador para producir el archivo ejecutable.
[68] Los nombres de las secciones provienen de la sintaxis del lenguaje ensamblador.
[69] El cargador o *loader* es parte del sistema operativo; se encarga de cargar en memoria los archivos ejecutables.
[70] LINK.EXE es el enlazador estándar de Microsoft, utilizado en Visual C++.
[71] POLINK.EXE es el enlazador de Pelles C, un compilador gratuito de C para Windows: http://www.smorgasbordet.com/pellesc/

El contenido de la tabla de importación se puede combinar y traslapar con otras estructuras y secciones del ejecutable. Por ejemplo, la herramienta *ImpLib SDK*[72] permite generar *librerías de importación*[73] personalizadas con tablas de importación más compactas. Esto contribuye a minimizar el tamaño del ejecutable.

No todos los compiladores utilizan enlazador. Por ejemplo, el esamblador **Flat Assembler**[74] (FASM) permite generar archivos ejecutables directamente. Entonces, el programador puede definir manualmente todas las estructuras del formato ejecutable y los valores de alineación. De esta manera se tiene control sobre cada byte del archivo ejecutable, lo cual hace posible realizar cualquier tipo de optimización, pero el riesgo de producir errores de formato aumenta. Tanto el compilador *FreeBASIC* como *ImpLib SDK*, entre muchas otras herramientas de desarrollo, utilizan FASM.

En general, las técnicas que consisten en combinar o acortar las estructuras internas del archivo ejecutable pueden afectar la compatibilidad con versiones específicas de sistema operativo. El hecho de que el ejecutable funcione correctamente en una versión no garantiza que funcione en otras. Las versiones nuevas pueden requerir valores de alineación más grandes o generar errores si diferentes estructuras se encuentran combinadas o con un tamaño menor que el esperado. Por lo tanto, el uso de estas técnicas no es aconsejable en ambientes no controlados. Inclusive en demoscene estas técnicas se aplican como última opción durante la fase final de desarrollo, luego de haber optimizado y comprimido el código al máximo posible.

[72] *ImpLib SDK*. https://implib.sourceforge.io/ind_es.htm
[73] La librería de importación o *biblioteca de importación* es un archivo de código objeto, el cual es usado por el enlazador para generar la tabla de importación del ejecutable.
[74] *FASM*. https://flatassembler.net/

Epílogo

«Ars longa, vita brevis»[75]
Hipócrates, 400 a. C.

Seguramente aún faltan muchos temas específicos de demoscene por cubrir. Las tecnologías gráficas, la música sintetizada y las técnicas de optimización y compactación descritas en este libro son tan solo «la punta del iceberg». Además, demoscene evoluciona y se actualiza constantemente. Una prueba de ello son las demos para la web y plataformas móviles. Esta evolución es parte de la cultura digital. Quizás, algún día podremos apreciar demos hechas con enjambres de drones, hologramas y tecnologías que aún no existen.

Si esta obra logró despertar el interés en demoscene el siguiente paso puede ser la participación en cualquiera de las demo-parties que se realizan periódicamente alrededor del mundo. Estos eventos incluyen actividades informativas y recreativas para el público en general. La base de demoscene es la diversión, más que la competencia. Actualmente muchos de estos eventos se realizan de manera virtual.

Mucho antes de que existiera demoscene, Sholom Gherman, un reconocido filósofo y estudioso de las artes, decía que el trabajo produce satisfacción estética. Demoscene ilustra perfectamente este concepto, ya que las demos son producto de un esfuerzo colectivo intenso. Los autores de las demos son quienes disfrutan más el resultado de su trabajo, pero también disfrutan el trabajo mismo. Generalmente no ganan dinero ni fama. Entonces, lo hacen porque sienten la necesidad de demostrar sus habilidades creando obras que desafían la imaginación.

[75] Se puede traducir del latín como: «Perfeccionar un arte toma mucho tiempo, pero la vida es breve».

Bibliografía

[Adan]

Adan, Victor (2010). «*Discrete Time 1-bit Music: foundations and models*». En inglés. Universidad de Columbia. Tesis Doctorado. https://victoradan.github.io/pdfs/va_phdthesis.pdf

[Ahoy]

Kevelson, Morton (1985). «*ICEPIC*». En inglés. Ahoy! Vol. 22. ISSN 8750-4383. https://archive.org/details/Ahoy_Issue_22_1985-10_Ion_International_US/page/n71

[Borzyskowski]

Borzyskowski, George (1997). «*The Hacker Demo Scene and its Cultural Artifacts*». En inglés. Universidad de Curtin. Perth. Archivado: http://web.archive.org/web/19970629185951/curtin.edu.au/conference/cybermind/papers/borzysko.html

[Bush]

Bush, Randy (1992). «*FidoNet: Technology, Use, Tools, and History*». En inglés. https://www.fidonet.org/inet92_Randy_Bush.txt

[Carlsson]

Carlsson, Anders (2009). «*The Forgotten Pioneers of Creative Hacking and Social Networking – Introducing the Demoscene*». En inglés. p.17. ISBN 978-0-9807186-3-8 https://www.mat.ucsb.edu/Publications/burbano_MAH2009.pdf

[Chen]

Chen, Raymond (2020). «*A look back at memory models in 16-bit MS-DOS*». En inglés. Microsoft. https://devblogs.microsoft.com/oldnewthing/20200728-00/?p=104012

[Chris]

Chris, Cynthia; Gerstner, David A. (2013). «*Media Authorship*». En inglés. Routledge. p.160. ISBN 978-0-203-13601-0

[Collins]

Collins, Karen (2017). «*From Pac-Man to Pop Music: Interactive Audio in Games and New Media*». En inglés. Routledge. ISBN 978-1-351-21772-9

[Debord]

Debord, Jean (2012). «*Mandelbrot and Julia sets with PANORAMIC and FreeBASIC*». En inglés. Back2Basic. Vol. 6. http://back2basic.phatcode.net/?Issue-%236

[Galuscenko]

Galuscenko, Dmitry (2003). «*История одного байта*». En ruso. Archivado: https://web.archive.org/web/20120511031802/wasm.ru/article.php?article=onebyte

[Gherman]

Gherman, Sh. M.; Skatershchikov, V. K. (1970). «*Беседы об эстетике*». En ruso. Editorial «Znanie». Moscú.

[González]

González, Lorena (2014). «*Chiptune: música de 8 bits*». Cartel Urbano. ISSN 1900-6462 https://cartelurbano.com/musica/chiptune-musica-de-8-bits

[Green]

Green, Dave (1995). «*Demo or Die!*». En inglés. Wired Magazine. Vol. 3-07. https://www.wired.com/1995/07/democoders/

[Hartmann]

Hartmann, Doreen (2017). «*Digital Art Natives: Praktiken, Artefakte und Strukturen der Computer-Demoszene*». En alemán. Kulturverlag Kadmos. ISBN 978-3-86599-343-4

[Jagdmann]

Jagdmann, Dirk (2008). «*The Farbrausch way to make demos*». En inglés. TNG Big Tech Day. https://llg.cubic.org/docs/farbrauschDemos/

[Kopka]

Kopka, Tobias (2021). «*Demoscene accepted as UNESCO cultural heritage in Germany*». En inglés. http://demoscene-the-art-of-coding.net/2021/03/20/ demoscene-accepted-as-unesco-cultural-heritage-in-germany/

[Phillips]

Phillips, Dave (2000). «*About the Mod: Part One*». En inglés. Linux Journal. ISSN 1075-3583 https://www.linuxjournal.com/article/4349

[Raiter]

Raiter, Brian (1999). «*A Whirlwind Tutorial on Creating Really Teensy ELF Executables for Linux*». En inglés. http://www.muppetlabs.com/~breadbox/software/tiny/teensy.html

[Reunanen]

Reunanen, Markku; Silvast, Antti (2009). «*Demoscene Platforms: A Case Study on the Adoption of Home Computers*». En inglés. Springer, Berlín. DOI 10.1007/978-3-642-03757-3_30 https://link.springer.com/content/pdf/10.1007%2F978-3-642-03757-3_30.pdf

[Stark]

Stark, Joan G. (2001). «*The History of ASCII (Text) Art»*. En inglés. Archivado: https://web.archive.org/web/20091026141759/geocities.com/SoHo/7373/history.htm

[Sutter]

Sutter, Paul (2021). «*Is there a pattern to the universe?*». En inglés. Space. https://www.space.com/universe-pattern-fractals-cosmic-web

[Svensson]

Svensson, Carl (2021). «*Pondering the Scene: Why are Demos European?*». En inglés. https://datagubbe.se/sceneorig/

[Szarafinski]

Szarafinski, Stephan (1995). «*Here comes my story of Tilburg...*». En inglés. http://fms.komkon.org/MSX/Docs/Tilburg1995.txt

[Warren]

Warren, Henry S. (2013) «*Hacker's delight*». En inglés. Addison-Wesley. 2ª Ed. p. 68. ISBN 978-0-321-84268-8

[Wasm]

Volodya; NEOx (2003). «*Об упаковщиках в последний раз*». En ruso. Hi-Tech/UinC. Archivado:
Parte I: http://web.archive.org/web/20121124070139/wasm.ru/article.php?article=packlast01
Parte II: http://web.archive.org/web/20121124054820/wasm.ru/article.php?article=packers2

Índice