Encyclopedia of Computer and ASCII Arts

Marylynn Lyman

Brennan Wofford

First Edition, 2012

ISBN 978-81-323-1271-0

Published by:
College Publishing House
4735/22 Prakashdeep Bldg,
Ansari Road, Darya Ganj,
Delhi - 110002
Email: info@wtbooks.com

Table of Contents

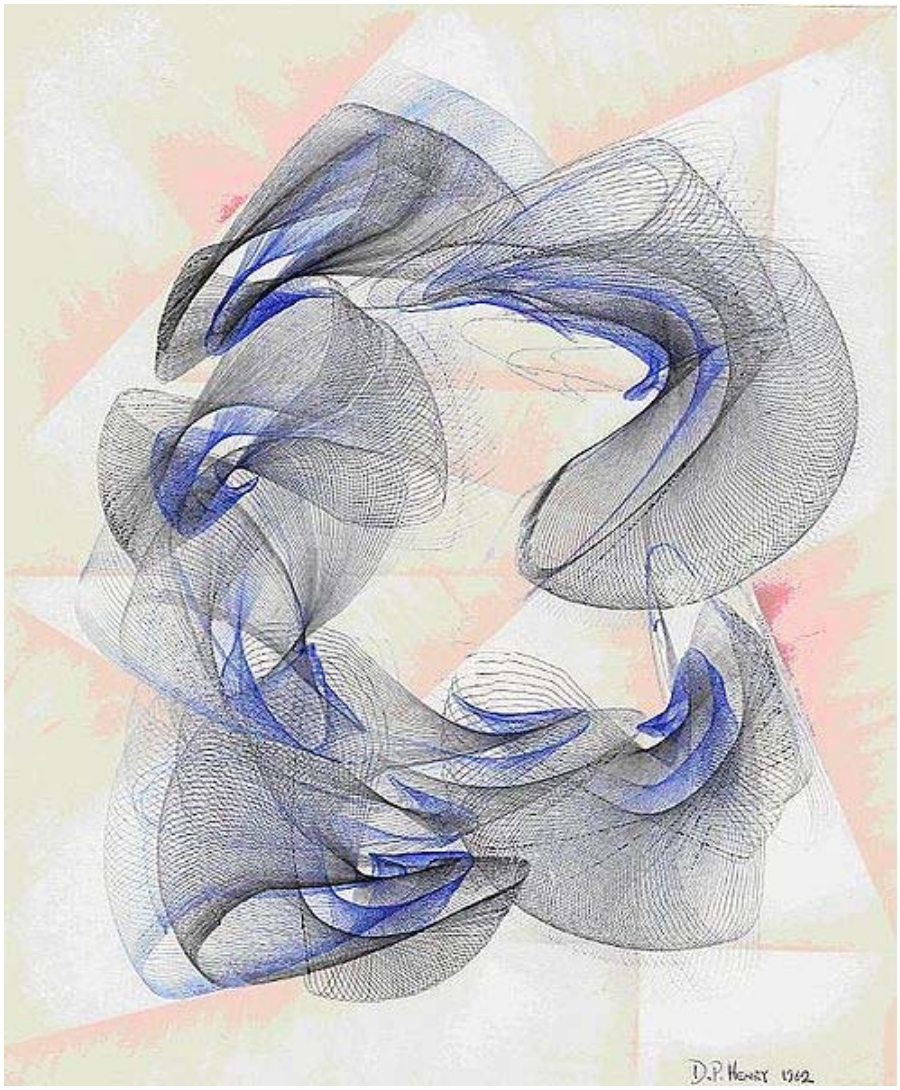
Chapter 1

Computer Art

Arambilet: *Dots on the I's*, D-ART 2009 Online Digital Art Gallery, exhibited at IV09 and CG09 computer Graphics conferences, at Pompeu Fabra University, Barcelona; Tianjin University, China; Permanent Exhibition at the London South Bank University

Computer art is any art in which computers play a role in production or display of the artwork. Such art can be an image, sound, animation, video, CD-ROM, DVD-ROM, videogame, web site, algorithm, performance or gallery installation. Many traditional disciplines are now integrating digital technologies and, as a result, the lines between traditional works of art and new media works created using computers has been blurred. For instance, an artist may combine traditional painting with algorithm art and other digital techniques. As a result, defining computer art by its end product can thus be difficult. Computer art is by its nature evolutionary since changes in technology and software directly affect what is possible. Notable artists in this vein include James Faure Walker, Manfred Mohr, Ronald Davis, Joseph Nechvatal, Matthias Groebel, George Grie, Olga Kisseleva, John Lansdown and Perry Welman.

History



Picture by drawing machine 1, Desmond Paul Henry, c.1960s

By the mid-1960s, most individuals involved in the creation of computer art were in fact engineers and scientists because they had access to the only computing resources available at university scientific research labs. Many artists tentatively began to explore the emerging computing technology for use as a creative tool. In the summer of 1962, Dr. A. Michael Noll programmed a digital computer at Bell Telephone Laboratories in Murray Hill, New Jersey to generate visual patterns solely for artistic purposes . His later computer-generated patterns simulated paintings by Piet Mondrian and Bridget Riley and become classics. Noll also used the patterns to investigate aesthetic preferences in the mid 1960s.

Computer art dates back to at least 1960, with the invention of the Henry Drawing Machine by Desmond Paul Henry. His work was shown at the Reid Gallery in London in 1962, after his machine-generated art won him the privilege of a one-man exhibition. In

1963 Joan Shogren of San Jose State University wrote a computer program based on artistic principles, resulting in an early public showing of computer art in San Jose, California on May 6, 1963.

The first two exhibitions of computer art were held in 1965- Computer-Generated Pictures, April 1965, at the Howard Wise Gallery in New York, and Generative Computergrafik, February 1965, at the Technische Hochschule in Stuttgart, Germany. The Stuttgart exhibit featured work by Georg Nees; the New York exhibit featured work by Bela Julesz and A. Michael Noll. Note the names of these expositions, not mentioning the word 'art', because these 'generated pictures' were not yet seen as such. A third exhibition was put up in November 1965 at Galerie Wendelin Niedlich in Stuttgart, Germany, showing works by Frieder Nake and Georg Nees.

In 1968, the Institute of Contemporary Arts (ICA) in London hosted one of the most influential early exhibitions of computer art- Cybernetic Serendipity. The exhibition included many of whom often regarded as the first true digital artists, Nam June Paik, Frieder Nake, Leslie Mezei, Georg Nees, A. Michael Noll, John Whitney, and Charles Csuri. One year later, the Computer Arts Society was founded, also in London.

At the time of the opening of Cybernetic Serendipity, in August 1968, a symposium was held in Zagreb, Yugoslavia, under the title "Computers and visual research". It took up the European artists movement of New Tendencies that had led to three exhibitions (in 1961, 63, and 65) in Zagreb of concrete, kinetic, and constructive art as well as op art and conceptual art. New Tendencies changed its name to "Tendencies" and continued with more symposia, exhibitions, a competition, and an international journal (bit international) until 1973.

Katherine Nash and Richard Williams published *Computer Program for Artists: ART 1* in 1970.

Xerox Corporation’s Palo Alto Research Center (PARC) designed the first Graphical User Interface (GUI) in the 1970s. The first Macintosh computer is released in 1984, since then the GUI became popular. Many graphic designers quickly accepted its capacity as a creative tool.

Output devices

Formerly, technology restricted output and print results: early machines used pen-and-ink plotters to produce basic hard copy. In the 1970s, the dot matrix printer (which was much like a typewriter) was used to reproduce varied fonts and arbitrary graphics. The first animations were created by plotting all still frames sequentially on a stack of paper, with motion transfer to 16-mm film for projection. During the 1970s and 1980s, dot matrix printers were used to produce most visual output while microfilm plotters were used for most early animation.

In 1976, the inkjet printer was invented with the increase in use of personal computers. The inkjet printer is now the cheapest and most versatile option for everyday digital color output. RasterImage Processing (RIP) is typically built into the printer or supplied as a software package for the computer; it is required to achieve the highest quality output. Basic inkjet devices do not feature RIP. Instead, they rely on graphic software to rasterize images. The laser printer, though more expensive than the inkjet, is another affordable output device available today.

Graphic software

Adobe Systems, founded in 1982, developed the PostScript language and digital fonts, making drawing painting and image manipulation software popular. Adobe Illustrator, a vector drawing program based on the Bézier curve introduced in 1987 and Adobe Photoshop, written by brothers Thomas and John Knoll in 1990 were developed for use on MacIntosh computers. and compiled for DOS/Windows platforms by 1993.

Chapter 2

Computer Graphics

The development of computer graphics, has made computers easier to interact with, and better for understanding and interpreting many types of data. Developments in computer graphics have had a profound impact on many types of media and have revolutionized animation, movies and the video game industry.

Overview

The term computer graphics has been used in a broad sense to describe "almost everything on computers that is not text or sound". Typically, the term *computer graphics* refers to several different things:

- the representation and manipulation of image data by a computer
- the various technologies used to create and manipulate images
- the images so produced, and
- the sub-field of computer science which studies methods for digitally synthesizing and manipulating visual content

Today, computers and computer-generated images touch many aspects of daily life. Computer imagery is found on television, in newspapers, for example in weather reports, or for example in all kinds of medical investigation and surgical procedures. A well-constructed graph can present complex statistics in a form that is easier to understand and interpret. In the media "such graphs are used to illustrate papers, reports, thesis", and other presentation material.

Many powerful tools have been developed to visualize data. Computer generated imagery can be categorized into several different types: 2D, 3D, 5D, and animated graphics. As technology has improved, 3D computer graphics have become more common, but 2D computer graphics are still widely used. Computer graphics has emerged as a sub-field of computer science which studies methods for digitally synthesizing and manipulating visual content. Over the past decade, other specialized fields have been developed like information visualization, and scientific visualization more concerned with "the visualization of three dimensional phenomena (architectural, meteorological, medical, biological, etc.), where the emphasis is on realistic renderings of volumes, surfaces, illumination sources, and so forth, perhaps with a dynamic (time) component".

Initial Development of Computer Graphics

The advance in computer graphics was to come from one MIT student, Ivan Sutherland. In 1961 Sutherland created another computer drawing program called Sketchpad. Using a light pen, Sketchpad allowed one to draw simple shapes on the computer screen, save them and even recall them later. The light pen itself had a small photoelectric cell in its tip. This cell emitted an electronic pulse whenever it was placed in front of a computer screen and the screen's electron gun fired directly at it. By simply timing the electronic pulse with the current location of the electron gun, it was easy to pinpoint exactly where the pen was on the screen at any given moment. Once that was determined, the computer could then draw a cursor at that location.

Sutherland seemed to find the perfect solution for many of the graphics problems he faced. Even today, many standards of computer graphics interfaces got their start with this early Sketchpad program. One example of this is in drawing constraints. If one wants to draw a square for example, s/he doesn't have to worry about drawing four lines perfectly to form the edges of the box. One can simply specify that s/he wants to draw a box, and then specify the location and size of the box. The software will then construct a perfect box, with the right dimensions and at the right location. Another example is that Sutherland's software modeled objects - not just a picture of objects. In other words, with a model of a car, one could change the size of the tires without affecting the rest of the car. It could stretch the body of the car without deforming the tires.

These early computer graphics were Vector graphics, composed of thin lines whereas modern day graphics are Raster based using pixels. The difference between vector graphics and raster graphics can be illustrated with a shipwrecked sailor. He creates an SOS sign in the sand by arranging rocks in the shape of the letters "SOS." He also has some brightly colored rope, with which he makes a second "SOS" sign by arranging the rope in the shapes of the letters. The rock SOS sign is similar to raster graphics. Every pixel has to be individually accounted for. The rope SOS sign is equivalent to vector graphics. The computer simply sets the starting point and ending point for the line and perhaps bend it a little between the two end points. The disadvantages to vector files are that they cannot represent continuous tone images and they are limited in the number of colors available. Raster formats on the other hand work well for continuous tone images and can reproduce as many colors as needed.

Also in 1961 another student at MIT, Steve Russell, created the first video game, Spacewar. Written for the DEC PDP-1, Spacewar was an instant success and copies started flowing to other PDP-1 owners and eventually even DEC got a copy. The engineers at DEC used it as a diagnostic program on every new PDP-1 before shipping it. The sales force picked up on this quickly enough and when installing new units, would run the world's first video game for their new customers.

E. E. Zajac, a scientist at Bell Telephone Laboratory (BTL), created a film called "Simulation of a two-giro gravity attitude control system" in 1963. In this computer generated film, Zajac showed how the attitude of a satellite could be altered as it orbits

the Earth. He created the animation on an IBM 7090 mainframe computer. Also at BTL, Ken Knowlton, Frank Sindon and Michael Noll started working in the computer graphics field. Sindon created a film called Force, Mass and Motion illustrating Newton's laws of motion in operation. Around the same time, other scientists were creating computer graphics to illustrate their research. At Lawrence Radiation Laboratory, Nelson Max created the films, "Flow of a Viscous Fluid" and "Propagation of Shock Waves in a Solid Form." Boeing Aircraft created a film called "Vibration of an Aircraft."

It wasn't long before major corporations started taking an interest in computer graphics. TRW, Lockheed-Georgia, General Electric and Sperry Rand are among the many companies that were getting started in computer graphics by the mid 1960's. IBM was quick to respond to this interest by releasing the IBM 2250 graphics terminal, the first commercially available graphics computer.

Ralph Baer, a supervising engineer at Sanders Associates, came up with a home video game in 1966 that was later licensed to Magnavox and called the Odyssey. While very simplistic, and requiring fairly inexpensive electronic parts, it allowed the player to move points of light around on a screen. It was the first consumer computer graphics product.

Also in 1966, Sutherland at MIT invented the first computer controlled head-mounted display (HMD). Called the Sword of Damocles because of the hardware required for support, it displayed two separate wireframe images, one for each eye. This allowed the viewer to see the computer scene in stereoscopic 3D. After receiving his Ph.D. from MIT, Sutherland became Director of Information Processing at ARPA (Advanced Research Projects Agency), and later became a professor at Harvard.

Dave Evans was director of engineering at Bendix Corporation's computer division from 1953 to 1962, after which he worked for the next five years as a visiting professor at Berkeley. There he continued his interest in computers and how they interfaced with people. In 1968 the University of Utah recruited Evans to form a computer science program, and computer graphics quickly became his primary interest. This new department would become the world's primary research center for computer graphics.

In 1967 Sutherland was recruited by Evans to join the computer science program at the University of Utah. There he perfected his HMD. Twenty years later, NASA would re-discover his techniques in their virtual reality research. At Utah, Sutherland and Evans were highly sought after consultants by large companies but they were frustrated at the lack of graphics hardware available at the time so they started formulating a plan to start their own company.

A student by the name of Edwin Catmull started at the University of Utah in 1970 and signed up for Sutherland's computer graphics class. Catmull had just come from The Boeing Company and had been working on his degree in physics. Growing up on Disney, Catmull loved animation yet quickly discovered that he didn't have the talent for drawing. Now Catmull (along with many others) saw computers as the natural progression of animation and they wanted to be part of the revolution. The first animation that Catmull

saw was his own. He created an animation of his hand opening and closing. It became one of his goals to produce a feature length motion picture using computer graphics. In the same class, Fred Parke created an animation of his wife's face. Because of Evan's and Sutherland's presence, UU was gaining quite a reputation as the place to be for computer graphics research so Catmull went there to learn 3D animation.

As the UU computer graphics laboratory was attracting people from all over, John Warnock was one of those early pioneers; he would later found Adobe Systems and create a revolution in the publishing world with his PostScript page description language. Tom Stockham led the image processing group at UU which worked closely with the computer graphics lab. Jim Clark was also there; he would later found Silicon Graphics, Inc.

The first major advance in 3D computer graphics was created at UU by these early pioneers, the hidden-surface algorithm. In order to draw a representation of a 3D object on the screen, the computer must determine which surfaces are "behind" the object from the viewer's perspective, and thus should be "hidden" when the computer creates (or renders) the image.

Image types

2D computer graphics

Raster graphic sprites (left) and masks (right)

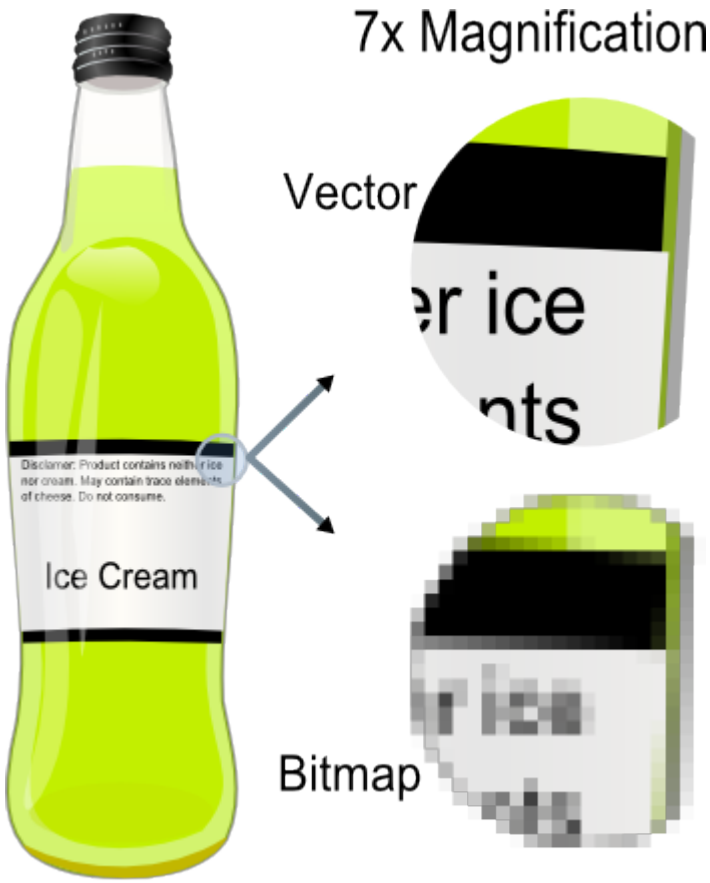
2D computer graphics are the computer-based generation of digital images—mostly from two-dimensional models, such as 2D geometric models, text, and digital images, and by techniques specific to them.

2D computer graphics are mainly used in applications that were originally developed upon traditional printing and drawing technologies, such as typography, cartography, technical drawing, advertising, etc.. In those applications, the two-dimensional image is not just a representation of a real-world object, but an independent artifact with added semantic value; two-dimensional models are therefore preferred, because they give more direct control of the image than 3D computer graphics, whose approach is more akin to photography than to typography.

Pixel art

Pixel art is a form of digital art, created through the use of raster graphics software, where images are edited on the pixel level. Graphics in most old (or relatively limited) computer and video games, graphing calculator games, and many mobile phone games are mostly pixel art.

Vector graphics



Example showing effect of vector graphics versus raster (bitmap) graphics

Vector graphics formats are complementary to raster graphics, which is the representation of images as an array of pixels, as it is typically used for the representation of photographic images. There are instances when working with vector tools and formats is best practice, and instances when working with raster tools and formats is best practice. There are times when both formats come together. An understanding of the advantages and limitations of each technology and the relationship between them is most likely to result in efficient and effective use of tools.

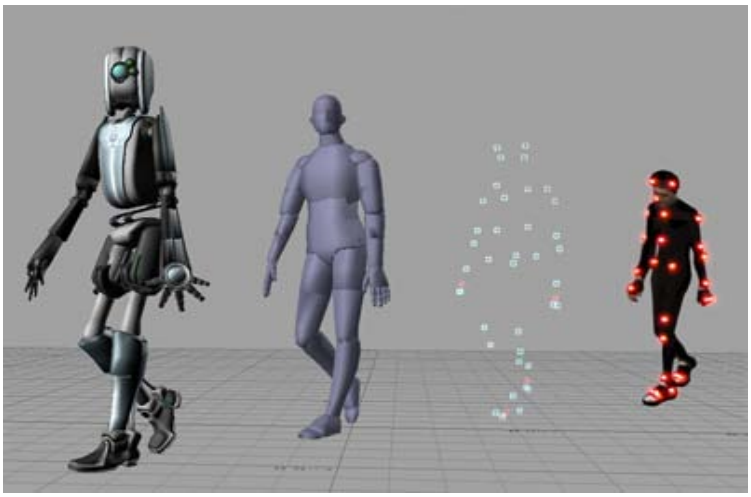
3D computer graphics

3D computer graphics in contrast to 2D computer graphics are graphics that use a three-dimensional representation of geometric data that is stored in the computer for the purposes of performing calculations and rendering 2D images. Such images may be for later display or for real-time viewing.

Despite these differences, 3D computer graphics rely on many of the same algorithms as 2D computer vector graphics in the wire frame model and 2D computer raster graphics in the final rendered display. In computer graphics software, the distinction between 2D and 3D is occasionally blurred; 2D applications may use 3D techniques to achieve effects such as lighting, and primarily 3D may use 2D rendering techniques.

3D computer graphics are often referred to as 3D models. Apart from the rendered graphic, the model is contained within the graphical data file. However, there are differences. A 3D model is the mathematical representation of any three-dimensional object. A model is not technically a graphic until it is visually displayed. Due to 3D printing, 3D models are not confined to virtual space. A model can be displayed visually as a two-dimensional image through a process called *3D rendering,* or used in non-graphical computer simulations and calculations. There are some 3D computer graphics software for users to create 3D images.

Computer animation



An example of Computer animation produced using Motion capture

Computer animation is the art of creating moving images via the use of computers. It is a subfield of computer graphics and animation. Increasingly it is created by means of 3D computer graphics, though 2D computer graphics are still widely used for stylistic, low bandwidth, and faster real-time rendering needs. Sometimes the target of the animation is the computer itself, but sometimes the target is another medium, such as film. It is also referred to as CGI (Computer-generated imagery or computer-generated imaging), especially when used in films.

Virtual entities may contain and be controlled by assorted attributes, such as transform values (location, orientation, and scale) stored in an object's transformation matrix. Animation is the change of an attribute over time. Multiple methods of achieving animation exist; the rudimentary form is based on the creation and editing of keyframes, each storing a value at a given time, per attribute to be animated. The 2D/3D graphics software will interpolate between keyframes, creating an editable curve of a value mapped over time, resulting in animation. Other methods of animation include procedural and expression-based techniques: the former consolidates related elements of animated entities into sets of attributes, useful for creating particle effects and crowd simulations; the latter allows an evaluated result returned from a user-defined logical expression, coupled with mathematics, to automate animation in a predictable way (convenient for controlling bone behavior beyond what a hierarchy offers in skeletal system set up).

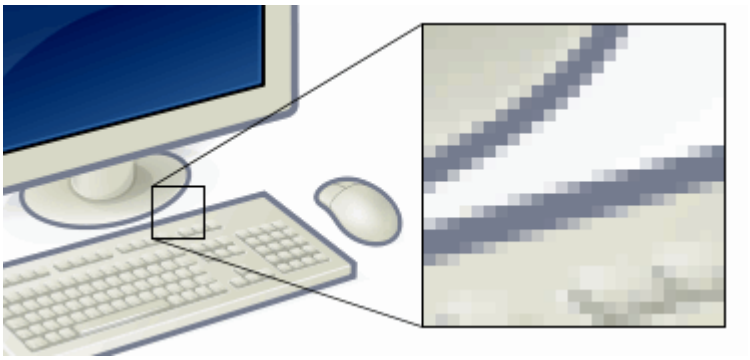
To create the illusion of movement, an image is displayed on the computer screen then quickly replaced by a new image that is similar to the previous image, but shifted slightly. This technique is identical to the illusion of movement in television and motion pictures.

Concepts and principles

Images are typically produced by optical devices;such as cameras, mirrors, lenses, telescopes, microscopes, etc. and natural objects and phenomena, such as the human eye or water surfaces.

A digital image is a representation of a two-dimensional image in binary format as a sequence of ones and zeros. Digital images include both vector images and raster images, but raster images are more commonly used.

Pixel



In the enlarged portion of the image individual pixels are rendered as squares and can be easily seen.

In digital imaging, a pixel (or picture element) is a single point in a raster image. Pixels are normally arranged in a regular 2-dimensional grid, and are often represented using dots or squares. Each pixel is a sample of an original image, where more samples

typically provide a more accurate representation of the original. The intensity of each pixel is variable; in color systems, each pixel has typically three components such as red, green, and blue.

Graphics

Graphics are visual presentations on some surface, such as a wall, canvas, computer screen, paper, or stone to brand, inform, illustrate, or entertain. Examples are photographs, drawings, line art, graphs, diagrams, typography, numbers, symbols, geometric designs, maps, engineering drawings, or other images. Graphics often combine text, illustration, and color. Graphic design may consist of the deliberate selection, creation, or arrangement of typography alone, as in a brochure, flier, poster, web site, or book without any other element. Clarity or effective communication may be the objective, association with other cultural elements may be sought, or merely, the creation of a distinctive style.

Arambilet: *Dots on the I's*, D-ART 2009 Online Digital Art Gallery, exhibited at IV09 and CG09 computer Graphics conferences, at Pompeu Fabra University, Barcelona; Tianjin University, China; Permanent Exhibition at the London South Bank University

Rendering

Rendering is the process of generating an image from a model (or models in what collectively could be called a *scene* file), by means of computer programs. A scene file contains objects in a strictly defined language or data structure; it would contain geometry, viewpoint, texture, lighting, and shading information as a description of the virtual scene. The data contained in the scene file is then passed to a rendering program to be processed and output to a digital image or raster graphics image file. The rendering program is usually built into the computer graphics software, though others are available as plug-ins or entirely separate programs. The term "rendering" may be by analogy with an "artist's rendering" of a scene. Though the technical details of rendering methods vary, the general challenges to overcome in producing a 2D image from a 3D representation stored in a scene file are outlined as the graphics pipeline along a rendering device, such as a GPU. A GPU is a purpose-built device able to assist a CPU in performing complex rendering calculations. If a scene is to look relatively realistic and predictable under virtual lighting, the rendering software should solve the rendering equation. The rendering equation doesn't account for all lighting phenomena, but is a general lighting

model for computer-generated imagery. 'Rendering' is also used to describe the process of calculating effects in a video editing file to produce final video output.

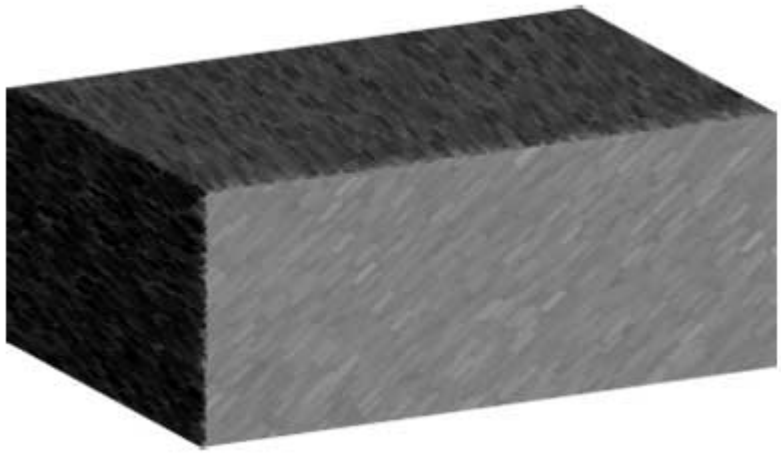
3D projection

3D projection is a method of mapping three dimensional points to a two dimensional plane. As most current methods for displaying graphical data are based on planar two dimensional media, the use of this type of projection is widespread, especially in computer graphics, engineering and drafting.

Ray tracing

Ray tracing is a technique for generating an image by tracing the path of light through pixels in an image plane. The technique is capable of producing a very high degree of photorealism; usually higher than that of typical scanline rendering methods, but at a greater computational cost.

Shading



Example of shading

Shading refers to depicting depth in 3D models or illustrations by varying levels of darkness. It is a process used in drawing for depicting levels of darkness on paper by applying media more densely or with a darker shade for darker areas, and less densely or with a lighter shade for lighter areas. There are various techniques of shading including cross hatching where perpendicular lines of varying closeness are drawn in a grid pattern to shade an area. The closer the lines are together, the darker the area appears. Likewise, the farther apart the lines are, the lighter the area appears. The term has been recently generalized to mean that shaders are applied.

Texture mapping

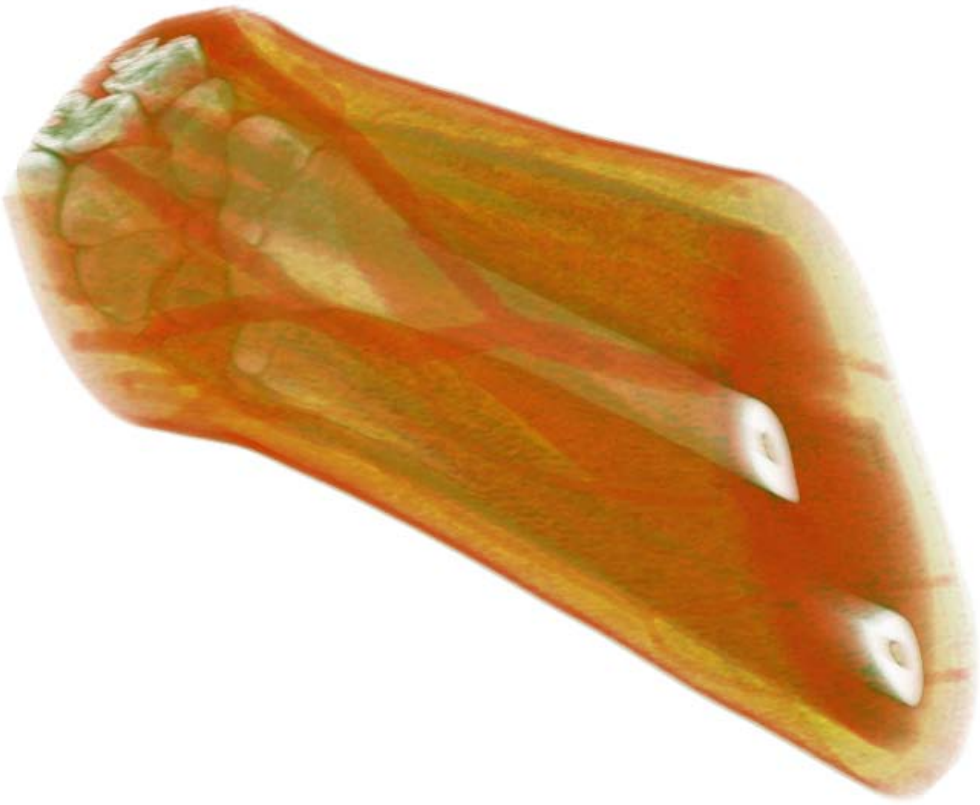
Texture mapping is a method for adding detail, surface texture, or colour to a computer-generated graphic or 3D model. Its application to 3D graphics was pioneered by Dr Edwin Catmull in 1974. A texture map is applied (mapped) to the surface of a shape, or polygon. This process is akin to applying patterned paper to a plain white box. Multitexturing is the use of more than one texture at a time on a polygon. Procedural textures (created from adjusting parameters of an underlying algorithm that produces an output texture), and bitmap textures

(created in an image editing application) are, generally speaking, common methods of implementing texture definition from a 3D animation program, while intended placement of textures onto a model's surface often requires a technique known as UV mapping.

Anti-aliasing

Rendering resolution-independent entities (such as 3D models) for viewing on a raster (pixel-based) device such as a LCD display or CRT television inevitably causes aliasing artifacts mostly along geometric edges and the boundaries of texture details; these artifacts are informally called "jaggies". Anti-aliasing methods rectify such problems, resulting in imagery more pleasing to the viewer, but can be somewhat computationally expensive. Various anti-aliasing algorithms (such as supersampling) are able to be employed, then customized for the most efficient rendering performance versus quality of the resultant imagery; a graphics artist should consider this trade-off if anti-aliasing methods are to be used. A pre-anti-aliased bitmap texture being displayed on a screen (or screen location) at a resolution different than the resolution of the texture itself (such as a textured model in the distance from the virtual camera) will exhibit aliasing artifacts, while any procedurally-defined texture will always show aliasing artifacts as they are resolution-independent; techniques such as mipmapping and texture filtering help to solve texture-related aliasing problems.

Volume rendering



Volume rendered CT scan of a forearm with different colour schemes for muscle, fat, bone, and blood.

Volume rendering is a technique used to display a 2D projection of a 3D discretely sampled data set. A typical 3D data set is a group of 2D slice images acquired by a CT or MRI scanner.

Usually these are acquired in a regular pattern (e.g., one slice every millimeter) and usually have a regular number of image pixels in a regular pattern. This is an example of a regular volumetric grid, with each volume element, or voxel represented by a single value that is obtained by sampling the immediate area surrounding the voxel.

3D modeling

3D modeling is the process of developing a mathematical, wireframe representation of any three-dimensional object, called a "3D model", via specialized software. Models may be created automatically or manually; the manual modeling process of preparing geometric data for 3D computer graphics is similar to plastic arts such as sculpting. 3D models may be created using multiple approaches: use of NURBS curves to generate

accurate and smooth surface patches, polygonal mesh modeling (manipulation of faceted geometry), or polygonal mesh subdivision (advanced tessellation of polygons, resulting in smooth surfaces similar to NURBS models). A 3D model can be displayed as a two-dimensional image through a process called *3D rendering*, used in a computer simulation of physical phenomena, or animated directly for other purposes. The model can also be physically created using 3D Printing devices.

Pioneers in graphic design

Charles Csuri

Charles Csuri is a pioneer in computer animation and digital fine art and created the first computer art in 1964. Csuri was recognized by *Smithsonian* as the father of digital art and computer animation, and as a pioneer of computer animation by the Museum of Modern Art (MoMA) and Association for Computing Machinery-SIGGRAPH.

Donald P. Greenberg

Donald P. Greenberg is a leading innovator in computer graphics. Greenberg has authored hundreds of articles and served as a teacher and mentor to many prominent computer graphic artists, animators, and researchers such as Robert L. Cook, Marc Levoy, and Wayne Lytle. Many of his former students have won Academy Awards for technical achievements and several have won the SIGGRAPH Achievement Award. Greenberg was the founding director of the NSF Center for Computer Graphics and Scientific Visualization.

A. Michael Noll

Noll was one of the first researchers to use a digital computer to create artistic patterns and to formalize the use of random processes in the creation of visual arts. He began creating digital computer art in 1962, making him one of the earliest digital computer artists. In 1965, Noll along with Frieder Nake and Georg Nees were the first to publicly exhibit their computer art. During April 1965, the Howard Wise Gallery exhibited Noll's computer art along with random-dot patterns by Bela Julesz.

A modern render of the Utah teapot, an iconic model in 3D computer graphics created by Martin Newell, 1975.
Other pioneers

- Jim Blinn
- Arambilet
- Benoît B. Mandelbrot
- Henri Gouraud
- Bui Tuong Phong
- Pierre Bézier
- Paul de Casteljau
- Daniel J. Sandin
- Alvy Ray Smith
- Ton Roosendaal
- Ivan Sutherland
- Steve Russell

The study of computer graphics

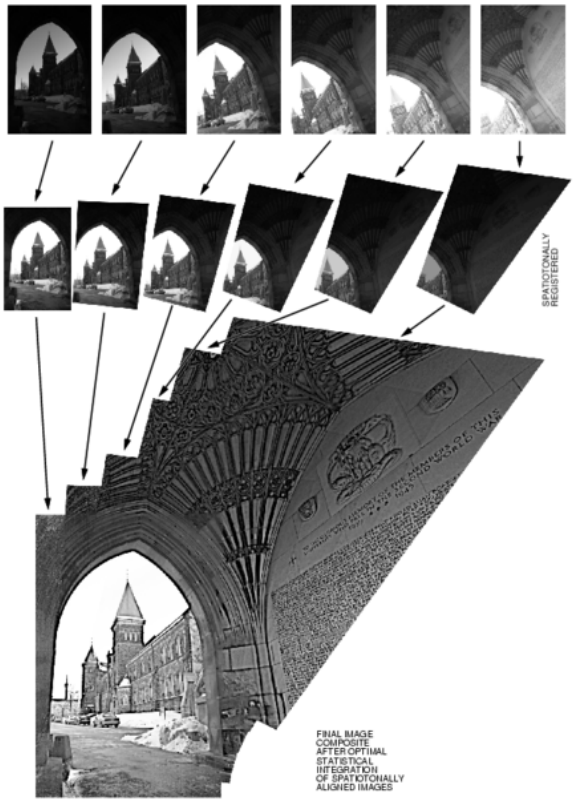
The study of computer graphics is a sub-field of computer science which studies methods for digitally synthesizing and manipulating visual content. Although the term often refers to three-dimensional computer graphics, it also encompasses two-dimensional graphics and image processing.

As an academic discipline, computer graphics studies the manipulation of visual and geometric information using computational techniques. It focuses on the *mathematical* and *computational* foundations of image generation and processing rather than purely aesthetic issues. Computer graphics is often differentiated from the field of visualization, although the two fields have many similarities.

Chapter 3

Computational Photography (Artistic) and Demoscene

Computational photography (artistic)



Computational ("undigital") photography attempts to capture an array of real numbers, rather than integers. Multiple differently exposed pictures of overlapping subject matter are mathematically aggregated into a complete whole.

Computational photography refers to computer image processing in which the image acquisition is influenced by the desire to process multiple pictures of the same subject matter, into an aggregate image or image sequence.

Computational photography is also known in the literature as "Intelligent Image Processing".

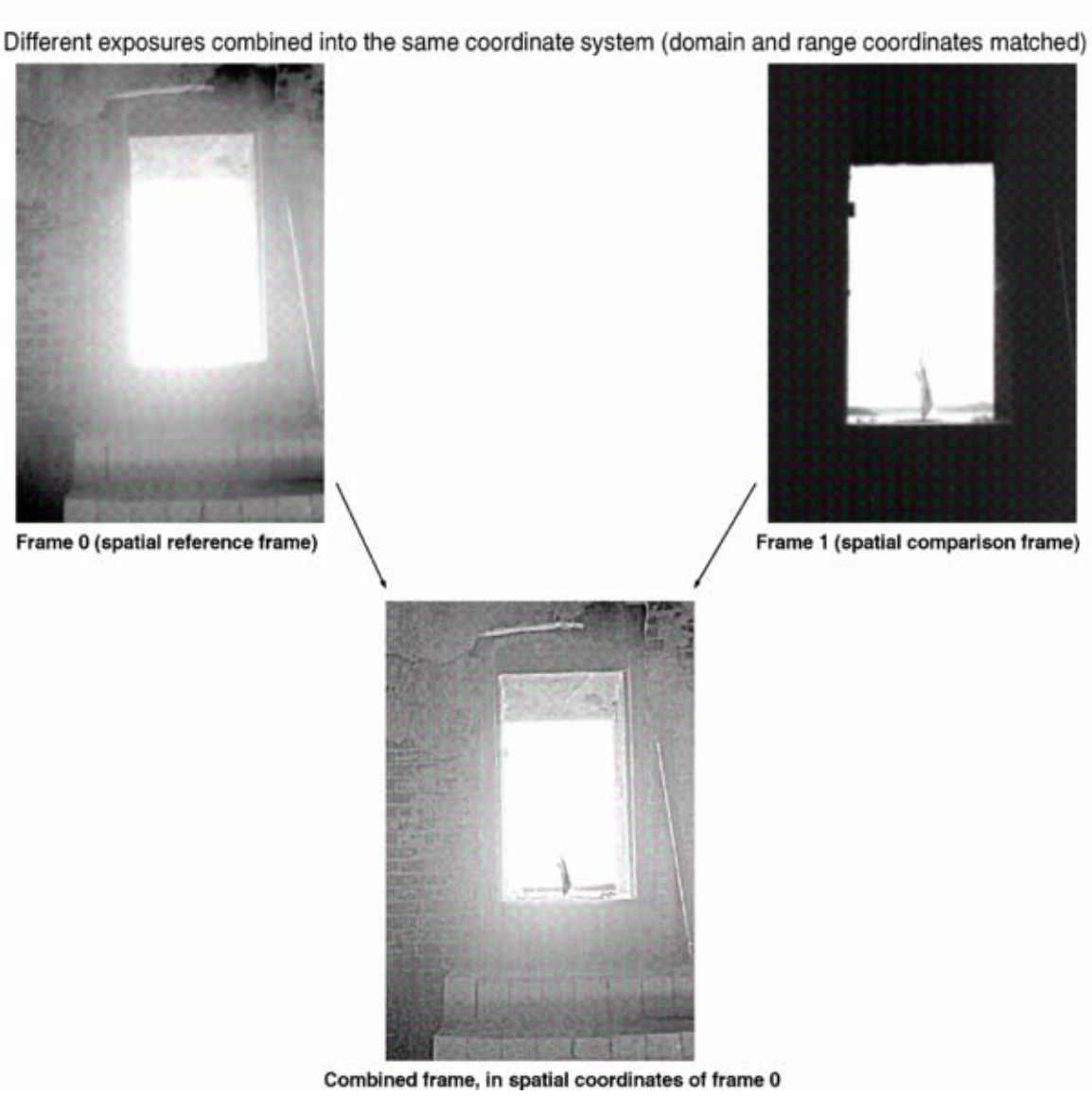
Relation to digital imaging

Digital photography refers to a quantized (finite) word length, i.e. to pixel quantities that are usually integers. Conversely, what is expected of computational photography is pixel quantities that are floating point numbers in acquisition, processing, storage, and display. This is done through file formats that represent floating point values quantimetrically, i.e. on some known (to within a constant scale factor) tone scale. Examples of quantimetric scales include, but are not limited to the following:

- Linear in the quantity of light received, up to a single unknown multiplicative constant for the entire image;
- Logarithmic in the quantity of light received, up to a single unknown additive constant for the entire image.

A recent paper, entitled "Being Undigital" attempted to define the essence of computational photography as an "undigital" capture of images. Ideally, computational photography captures, as best as possible, an array of real numbers rather than integers, while maintaining the ability to store, wirelessly send, process, and display the image on a continuous tone scale.

First simultaneous estimation of projection (homography) and gain change, together with composite of multiple differently exposed pictures to generate a high dynamic range image, 1991:



(Published in a 1993 paper, which was the precursor to the 1995 paper entitled "Being Undigital".)

File formats for computational photography

There are two main file formats for computational photography: one is based on an extension of the PPM (Portable Pixmap), and the other is based on an extension of JPEG.

PDM and PLM

Most computational photography uses double precision arrays, e.g. 64 bits per pixel (64BPP), Portable Double Maps use the headers P7 (greyscale) and P8 (color) in place of P5 (Portable Grey Map) and P6 (Portable Pix Map), respectively. Portable Lightspace

Maps also use double precision data, but quantimetrically, usually either linear or logarithmic. PLM headers are P9 (greyscale) or PA (color).

Both PDM and PLM are double precision; the difference is in whether they are in imagespace or lightspace. PDM is in imagespace, whereas PLM is in lightspace (i.e. quantimetric: either linear, logarithmic, or some other format that's specified in the header).

The most commonly represented quantimetric scale is the linear scale.

A typical 640 pixel wide, 480 pixel high color image in PLM format is represented in the following example:

```
PA
# cementinit pork.ppm 1.000000 1.000000 1.000000 -o pork.plm
640 480
255
1
2.200000
v\254^Zm\201^L\202@\372\357]5\224+x@\304\262vN^F^TR@u\252\340\224
T\204@$\341\3\33\332\214\333
{@_\256\262^C`@X@u\252\340\224   T\204@$\341\333\332\214\333{@_\256\
... (binary data)...
```

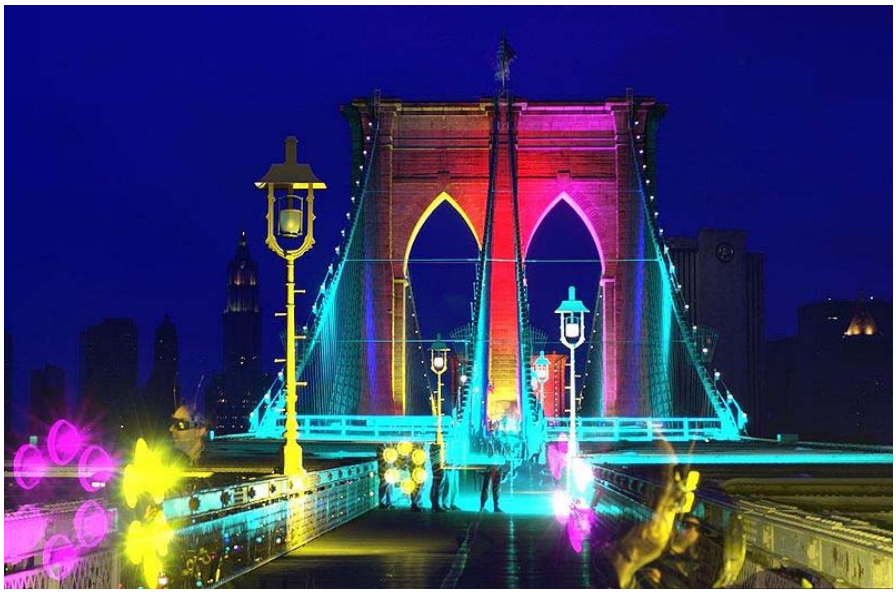
Many utilities such as those available in the Comparametric Toolkit (available freesource, from Sourceforge), preserve commands as comments in the header. Like PPM, a "#" sign at the beginning of a line denotes a comment, while still in the ascii text header. The header is ascii text (human readable and easy to work with) and the data is binary. Images in PLM are denoted with the plm file extension, for example, "test.plm".

The JLM file format is a compressed version of the PLM, that generalizes the JPEG algorithm, to do floating point arrays.

Computational photography as an artistic medium

A 1981 wearable computational photography apparatus

Early 1980s example of wearable computational photography as an art form



Wearable Computational Photography originated in the 1970s and early 1980s, and has evolved into a more recent art form.

Computational photography, as an art form, has been practiced by capture of differently exposed pictures of the same subject matter, and combining them together. This was the insipiration for the development of the wearable computer in the 1970s and early 1980s.

Historical inspiration

Computational photography was inspired by the work of Charles Wyckoff, and thus computational photography datasets (e.g. differently exposed pictures of the same subject matter that are taken in order to make a single composite image) are sometimes referred to as Wyckoff Sets, in his honor.

Early work in this area (joint estimation of image projection and exposure value) was undertaken by Mann and Candoccia.

Charles Wyckoff devoted much of his life to creating special kinds of 3-layer photographic films that captured different exposures of the same subject matter. A picture of a nuclear explosion, taken on Wyckoff's film, appeared on the cover of Life Magazine and showed the dyamic range from dark outer areas to inner core.

Demoscene

The **demoscene** is a computer art subculture that specializes in producing demos, which are non-interactive audio-visual presentations that run in real-time on a computer. The main goal of a demo is to show off programming, artistic, and musical skills.

The demoscene first appeared during the 8-bit era on computers such as the Commodore 64, ZX Spectrum and Amstrad CPC, and came to prominence during the rise of the 16/32-bit home computers (the Amiga and the Atari ST). In the early years, demos had a strong connection with software cracking. When a cracked program was started, the cracker or his team would take credit with a graphical introduction called a "crack intro" (shortened *cracktro*). Later, the making of intros and standalone demos evolved into a new subculture independent of the software (piracy) scene.

Concept

Screen shot from Second Reality, a famous demo by Future Crew.

Prior to the popularity of IBM PC compatibles, most home computers of a given line had relatively little variance in their basic hardware, which made their capabilities practically identical. Therefore, the variations among demos created for one computer line were attributed to programming alone, rather than one computer having better hardware. This created a competitive environment in which demoscene groups would try to outperform each other in creating amazing effects, and often to demonstrate why they felt one machine was better than another (for example Commodore 64 or Amiga versus Atari 800 or ST).

Demo writers went to great lengths to get every last bit of performance out of their target machine. Where games and application writers were concerned with the stability and functionality of their software, the demo writer was typically interested in how many CPU cycles a routine would consume and, more generally, how best to squeeze great activity onto the screen. Writers went so far as to exploit known hardware errors to produce effects that the manufacturer of the computer had not intended. The perception that the demo scene was going to extremes and charting new territory added to its draw.

Recent computer hardware advancements include faster processors, more memory, faster video graphics processors, and hardware 3D acceleration. With many of the past's challenges removed, the focus in making demos has moved from squeezing as much out of the computer as possible to making stylish, beautiful, well-designed real time artwork - a directional shift that many "old school demosceners" seem to disapprove of. This can be explained by the break introduced by the PC world, where the platform varies and most of the programming work that used to be hand-programmed is now done by the graphics card. This gives demo-groups a lot more artistic freedom, but can frustrate some of the old-schoolers for lack of a programming challenge. The old tradition still lives on, though. Demo parties have competitions with varying limitations in program size or platform (different series are called compos). On a modern computer the executable size may be limited to 64 kB or 4 kB. Programs of limited size are usually called intros. In other compos the choice of platform is restricted; only old computers, like the 8-bit Commodore 64, or the 32-bit Amiga or Atari ST, or mobile devices like handheld phones or PDAs are allowed. Such restrictions provide a challenge for coders, musicians and graphics artists and bring back the old motive of making a device do more than it was intended for.

History

Game Music IV on the Commodore 64 by Charles Deenen (also known as "The Mercenary Cracker" (TMC)) was perhaps one of the very first demos ever produced. Though TMC dated all his productions to 1991, this demo is known to have been produced in 1985.

The earliest computer programs that have some resemblance to demos and demo effects can be found among the so-called display hacks. Display hacks predate the demoscene for several decades, with the earliest examples dating back to the early 1950s.

Demos in the demoscene sense began as software crackers' "signatures", that is, crack screens and crack intros attached to software whose copy protection was removed. The first crack screens appeared on the Apple II computers in the late 1970s and early 1980s, and they were often nothing but plain text screens crediting the cracker or his group. Gradually, these static screens evolved into increasingly impressive-looking introductions containing animated effects and music. Eventually, many cracker groups started to release intro-like programs separately, without being attached to pirated software. These programs were initially known by various names, such as *letters* or *messages*, but they later came to be known as *demos*.

Simple demo-like music collections were put together on the C64 in 1985 by Charles Deenen, inspired by crack intros, using music taken from games and adding some homemade color graphics. In the following year the movement now known as the demoscene was born. The Dutch groups 1001 Crew and The Judges, both Commodore 64-based, are often mentioned as the earliest demo groups. Whilst competing with each other in 1986, they both produced pure demos with original graphics and music involving more than just casual work, and used extensive hardware trickery. At the same time demos from others, such as Antony Crowther (Ratt), had started circulating on Compunet in the United Kingdom. On the ZX Spectrum, Castor Cracking Group released their first demo called *Castor Intro* in 1986. The ZX Spectrum demo scene was slow to start, but it started to rise in the late 1980s, most noticeably in Eastern Europe.

Competition

The demoscene is a largely competition-oriented subculture, with groups and individual artists competing against each other in technical and artistic excellence. In the early days, this competition came in the form of setting records, like the number of "bobs" (blitter objects) on the screen per frame, or the number of DYCP (different Y Character position) scrollers on a C64. These days, there are organized competitions, or compos, held at demoparties, although there have been some online competitions as well. It has also been common for diskmags to have voting-based *charts* which provide ranking lists for the best coders, graphicians, musicians, demos and other things. However, the respect for charts has diminished since the 1990s.

Party-based competitions usually require the artist or a group member to be present at the event. The winners are selected by a public voting amongst the visitors and awarded at a prizegiving ceremony at the end of the party. Competitions at a typical demo event include a *demo compo*, an *intro compo* (usually 64K), a *graphics compo* and a *music compo*. Most parties also split some categories by platform, format or style.

There are no criteria or rules the voters should be bound by, and a visitor typically just votes for those entries that made the biggest impression on him or her. In the old demos, the impression was often attempted with programming techniques introducing new effects and breaking performance records in old effects. Over the years, the emphasis has moved from technical excellence to more artistic values such as overall design, audiovisual impact and mood.

The demoscene constitutes the most part of its own audience, with the opinions of the community itself considered the most valid. For example, it is often considered *lame* to win large events with works that appeal to the non-demomaking masses but do not adhere to good demoscene aesthetics. However, most of the demos regarded as the best of all time have appealed both to the demomaking community itself and a larger audience.

In the recent years, an initiative to award demos in an alternative way arose by the name of the Scene.org Awards. The essential concept of the awards was to avoid the subjectivity of mass-voting at parties, and select a well-renowned jury to handle the task of selecting the given year's best productions on several aspects, such as Best Graphics or Best 64k Intro.

Parties

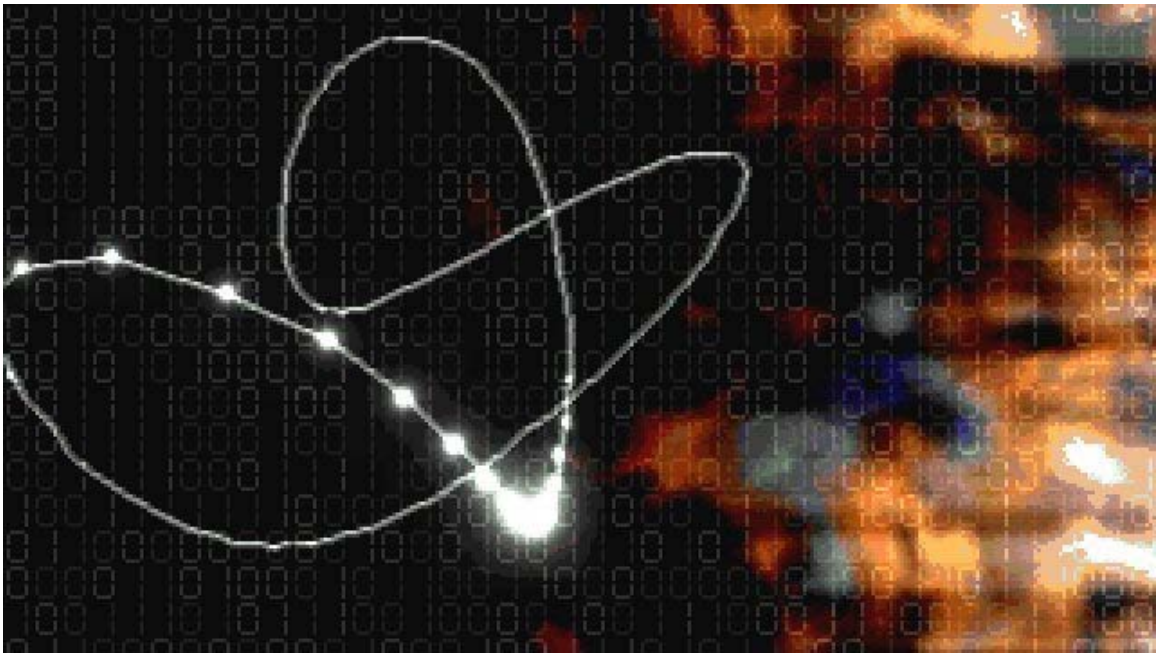
Assembly 2004 - a combination of a demoparty and a LAN party

A demoparty is an event which gathers demomakers and provides them competitions to compete in. A typical demoparty is a non-stop event lasting over a weekend, providing the visitors a lot of time for socializing. The competing works, at least those in the most important competitions, are usually shown at night, using a video projector and big loudspeakers.

Demoparties started to appear in the 1980s in the form of copyparties where software pirates and demomakers gathered to meet each other and share their software. Competitions did not become a major aspect of the events until the beginning of the 1990s.

Demoscene events are most frequent in Europe, with around fifty parties every year. For comparison, there have only been a dozen or so demoparties in the United States and Canada in total (such as Spring Break and NAID). Most events are local, gathering demomakers mostly from a single country, while the largest international parties (such as Breakpoint and Assembly) attract visitors from all over the globe.

Demo types



Screenshot from *Gift* by Potion, winner of the Mekka & Symposium 2000 Amiga 64k intro competition (Youtube)

The demoscene still exists on many platforms, including the PC, C64, MSX, ZX Spectrum, Amstrad CPC, Amiga, Atari, Dreamcast and Game Boy Advance. The large variety of platforms makes their respective demos hard to compare. Some 3D benchmark programs also have a demo or showcase mode, which derives its roots from the days of the 16-bit platforms.

There are several categories demos are informally classified into, the most important being the division between the "full-size" **demos** and the size-restricted **intros**, a difference visible in the competitions of nearly any demo party. The most typical competition categories for intros are the **64K intro** and the **4K intro**, where the size of the executable file is restricted to 65536 and 4096 bytes, respectively.

Groups

PC-Demo: Interceptor by Black Maiden.

A typical demo is created by a **demogroup**, which is a team of demosceners. Although some demogroups boast dozens of members, the number of individuals involved in a single production rarely exceeds ten. Since the demogroup is also a major way of self-identification for demosceners, even individual creations are usually associated with a group.

A demoscener is typically specialized in a certain area of creativity. The traditional division is in **coders**, **graphicians** and **musicians**, who are specialized in programming (often including overall design), still graphics (including 2D art and 3D modelling) and music, respectively. There are also demosceners who have little involvement in the actual demomaking but that do considerable work in areas such as party organizing.

Impact

A demo running on a TI-86 calculator

Although demos are still a more or less obscure form of art even in the traditionally active demoscene countries, the scene has had an impact on areas such as computer games industry and new media art.

A great deal of European game programmers, artists and musicians have come from the demoscene, often cultivating the learned techniques, practices and philosophies in their work. For example, the Finnish company Remedy Entertainment, known for the Max Payne series of games, was founded by the PC group Future Crew, and most of its employees are former or active Finnish demosceners. Sometimes demos even provide direct influence even to game developers that have no demoscene affiliation: for instance, Will Wright names demoscene as a major influence on the Maxis game Spore, which is largely based on procedural content generation.

Certain forms of computer art have a strong affiliation with the demoscene. Tracker music, for example, originated in the Amiga games industry but was soon heavily dominated by demoscene musicians.. Currently, there is a major tracking scene separate from the actual demoscene.

Over the years, desktop computer hardware capabilities have improved by orders of magnitude, and so for most programmers, tight hardware restrictions are no longer a common issue. Nevertheless, demosceners continue to study and experiment with creating impressive effects on limited hardware. Since handheld consoles and cellular phones have comparable processing power or capabilities to the desktop platforms of old (such as low resolution screens which require pixel-art, or very limited storage and memory for music replay), many demosceners have been able to apply their niche skills to develop games for these platforms, and earn a living doing so.

Some attempts have been made to increase the familiarity of demos as an art form. For example, there have been demo shows, demo galleries and demoscene-related books, sometimes even TV programs introducing the subculture and its works.

Sometimes a demoscene-based production may become very famous in technical contexts. For example, the 96-kilobyte FPS game *.kkrieger* by Farbrausch uses procedural content generation algorithms that are quite common on today's 64K intros but largely unknown to the computer games enthusiasts and the US-based game development community.

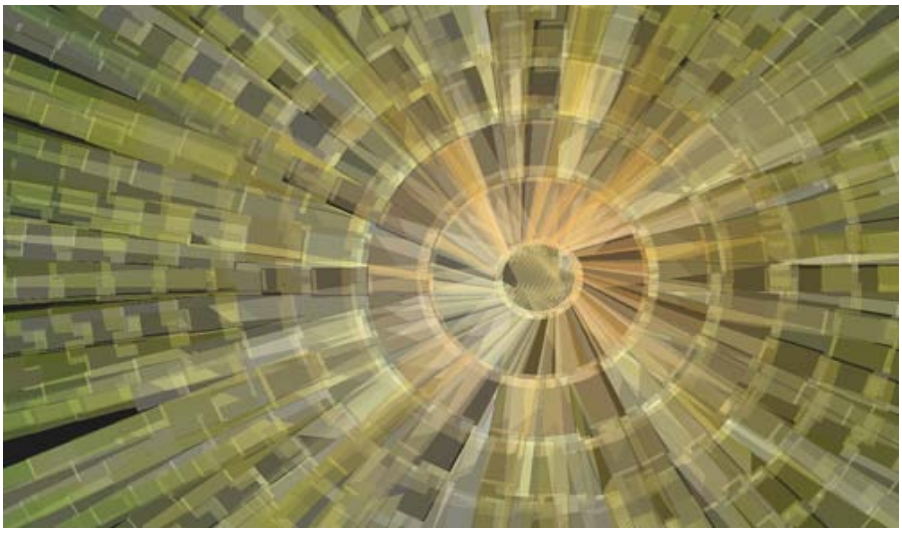
Chapter 4

Algorithmic Art and Fractal Art

Algorithmic art

Algorithmic art, also known as *algorithm art*, is art, mostly visual art, of which the design is generated by an algorithm. Algorithmic artists are sometimes called **algorists**.

Overview



"Octopod" by Mikael Hvidtfeldt Christensen. An example of algorithmic art produced with the software *Structure Synth*.

Algorithmic art is a subset of generative art, and is practically always executed by a computer. If executed by a computer, it is also classified as computer-generated art, but in much computer-generated art the role of the computer is confined to the execution. In contrast, in algorithmic art the creative design is the result of an algorithmic process, usually using a random or pseudo-random process to produce variability. Algorithmic art is also related to systems art.

It is usually digital art, although a number of artists work with plotters. Fractal art is an example of algorithmic art.

History

The earliest known examples of algorithmic art are artworks created by Georg Nees and Frieder Nake in the early 1960s. These works were executed by a plotter controlled by a personal computer, and were therefore computer-generated art but not digital art. The act of creation lay in writing the program, the sequence of actions to be performed by the plotter.

Aside from the ongoing work of Verostko and his fellow algorists, the next known examples are fractal artworks created in the mid to late 1980s. These are important here because they use a different means of execution. Whereas the earliest algorithmic art was "drawn" by a plotter, fractal art simply creates an image in computer memory; it is therefore digital art. The native form of a fractal artwork is an image stored on a computer –this is also true of very nearly all equation art and of most recent algorithmic art in general. However, in a stricter sense "fractal art" is not considered algorithmic art, because the algorithm is not devised by the artist.

The role of the algorithm

For a work of art to be considered algorithmic art, its creation must include a process based on an algorithm devised by the artist. Here, an algorithm is simply a detailed recipe for the design and possibly execution of an artwork, which may include computer code, functions, expressions, or other input which ultimately determines the form the art will take. This input may be mathematical, computational, or generative in nature. Inasmuch as algorithms tend to be deterministic, meaning that their repeated execution would always result in the production of identical artworks, some random factor is usually introduced. If the algorithm is executed by a computer, this can be the use of a pseudo-random number generator. Some artists also work with organically based gestural input which is then modified by an algorithm.

By this definition, algorithmic art is not to be confused with graphical methods such as generating a fractal out of a fractal program; it is necessarily concerned with the human factor (one's own algorithm, and not one that is pre-set in a package). The artist must be concerned with the most appropriate expression for their idea, just as a painter would be most concerned with the best application of colors. By this definition, defaulting to something like a fractal generator (and using it for all or most of your creations) would in essence be letting the computer dictate the form of the final work, and not truly be a creative art. The artist's self-made algorithms are an integral part of the authorship, as well as being a medium through which their ideas are conveyed.

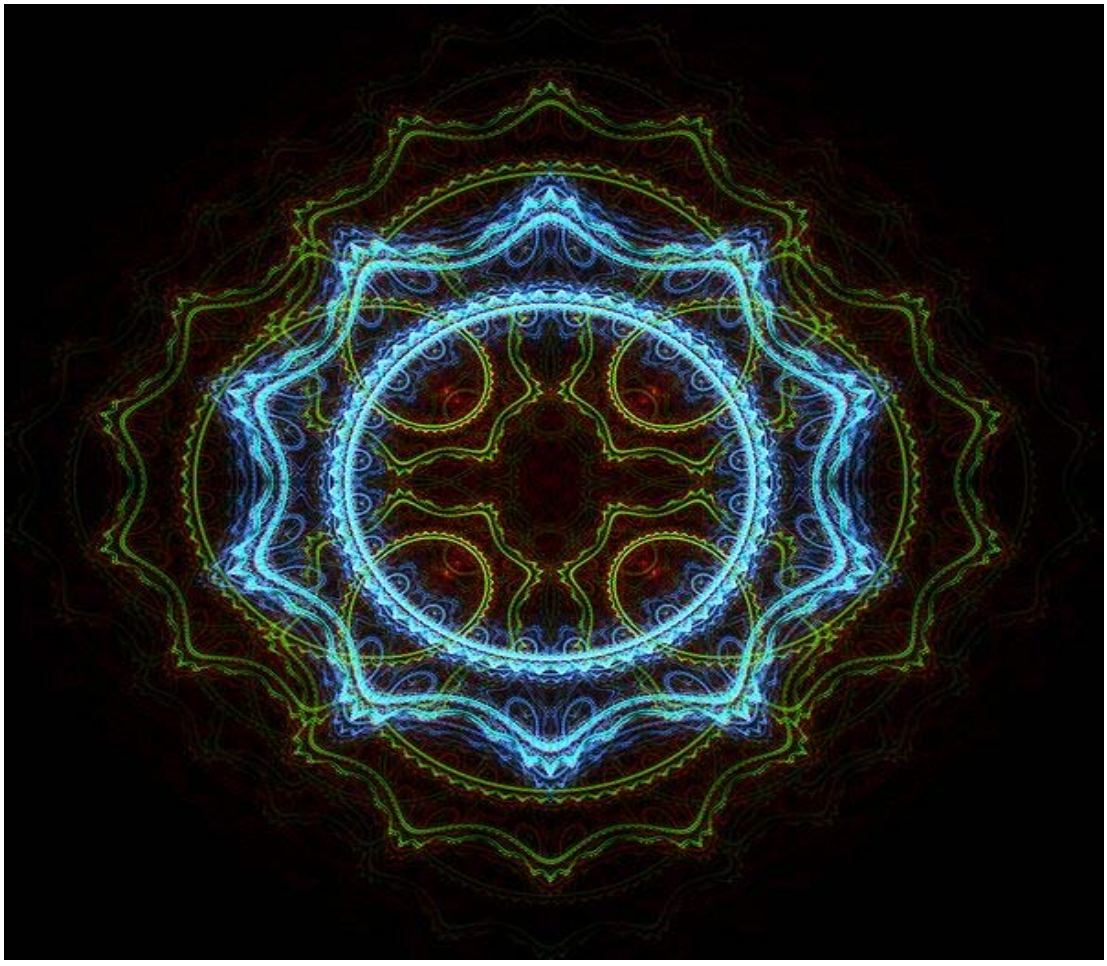
Algorists

"Algorist" is a term used for digital artists who create algorithmic art. One group of algorists is known as *Les Algoristes*.

Algorists formally began correspondence and establishing their identity as artists following a panel titled "Art and Algorithms" at SIGGRAPH in 1995. The co-founders were Roman Verostko and Jean-Pierre Hébert. Hébert is credited with coining the term and its definition, which is quite unsurprisingly, in the form of his own algorithm:

```
if (creation && object of art && algorithm && one's own algorithm) {
     include * an algorist *
} elseif (!creation || !object of art || !algorithm || !one's own
algorithm) {
     exclude * not an algorist *
}
```

Fractal art



A piece of fractal art generated in Apophysis.

Fractal art is a form of algorithmic art created by calculating fractal objects and representing the calculation results as still images, animations, and media. Fractal art is usually created indirectly with the assistance of fractal-generating software, iterating through three phases: setting parameters of appropriate fractal software, executing the possibly lengthy calculation and evaluating the product. In some cases, other graphics programs are used to further modify the images produced. This is called post-processing.

As stated by Kerry Mitchell in The Fractal Art Manifesto, "Fractal Art is a subclass of two-dimensional visual art, and is in many respects similar to photography—another art form that was greeted by skepticism upon its arrival. Fractal images typically are manifested as prints, bringing Fractal Artists into the company of painters, photographers, and printmakers. Fractals exist natively as electronic images. This is a format that traditional visual artists are quickly embracing, bringing them into FA's digital realm. Generating fractals can be an artistic endeavor, a mathematical pursuit, or just a soothing diversion. However, FA is clearly distinguished from other digital activities by what it is, and by what it is not." According to Mitchell, fractal art is not computerized art, lacking in rules, unpredictable, nor something that any person with access to a computer can do well. Instead, fractal art is expressive, creative, and requires input, effort, and intelligence. Most importantly, "fractal art is simply that which is created by Fractal Artists: ART."

Fractal art comes from uses fractal designs. Fractals are any of various extremely irregular curves or shapes for which any suitably chosen part is similar in shape to a given larger or smaller part when magnified or reduced to the same size. There are many different kinds of fractal images and can be subdivided into several groups.

- Fractals derived from standard geometry by using iterative transformations on an initial common figure like a straight line (the Cantor dust or the von Koch curve), a triangle (the Sierpinsky triangle), or a cube (the Menger sponge). The first fractal figures invented near the end of the 19th and early 20th centuries belong to this group.
- IFS (iterate function systems).
- Strange attractors.
- Flame fractals.
- L-system fractals.
- Fractals created by the iteration of complex polynomials: perhaps the most famous fractals.
- Quaternionic and (recently) hypernionic fractals. * Fractal terrains generated by random fractal processes.

Techniques

A 3D landscape generated with Terragen

Fractals of all four kinds have been used as the basis for digital art and animation. Starting with 2-dimensional details of fractals, such as the Mandelbrot Set, fractals have found artistic application in fields as varied as texture generation, plant growth simulation and landscape generation.

Fractals are sometimes combined with human-assisted evolutionary algorithms, either by iteratively choosing good-looking specimens in a set of random variations of a fractal artwork and producing new variations, to avoid dealing cumbersome or unpredictable parameters, or collectively, like in the Electric Sheep project, where people use fractal flames rendered with distributed computing as their screensaver and "rate" the flame they are viewing, influencing the server, which reduces the traits of the undesirables, and increases those of the desirables to produce a computer-generated, community-created piece of art.

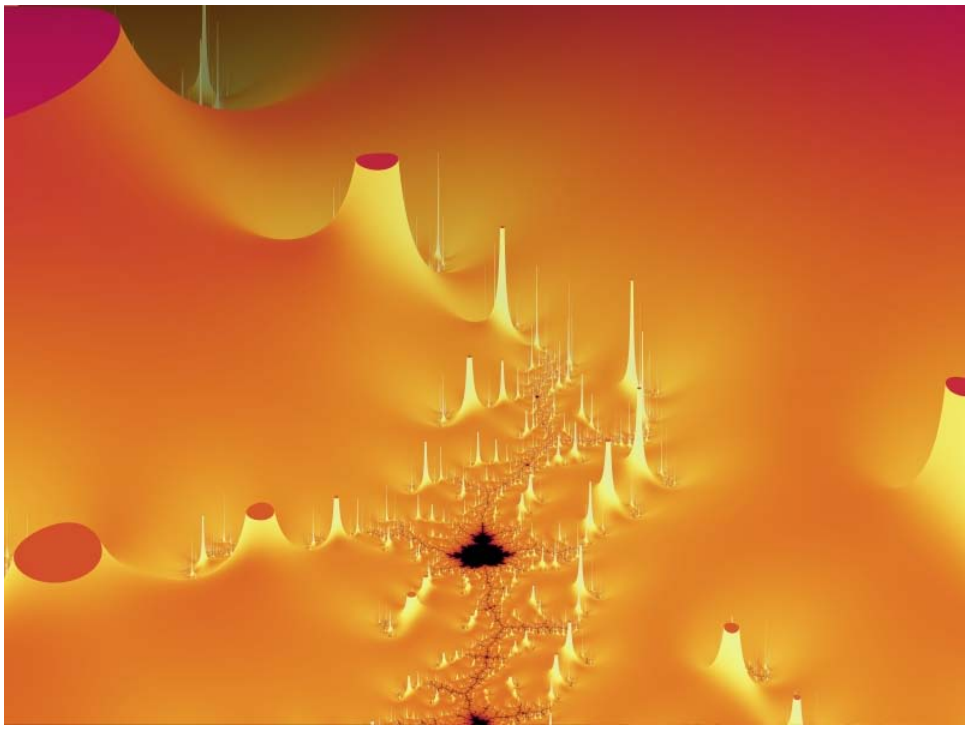
Landscapes

The first fractal image that was intended to be a work of art, seems to be the famous image on the cover of *Scientific American*, August 1985. This image showed a landscape

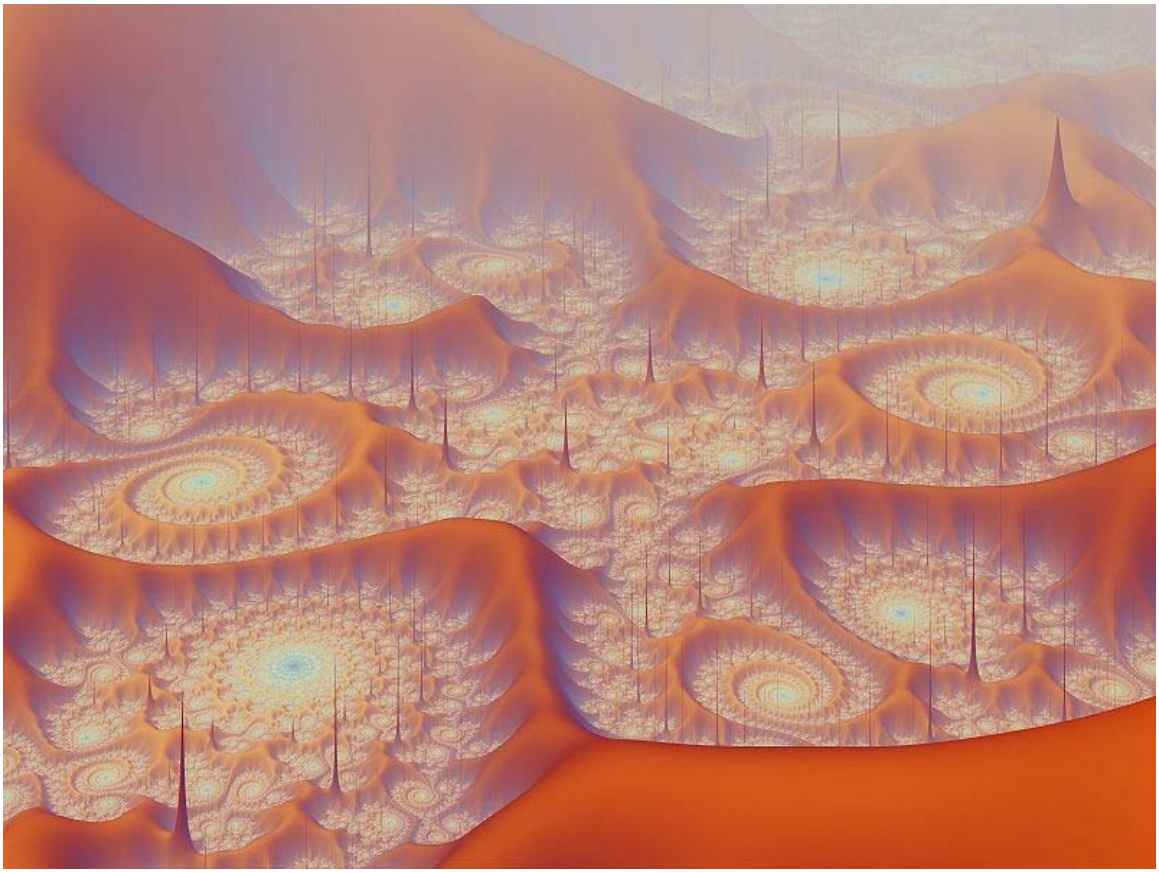
formed from the potential function on the domain outside the (usual) Mandelbrot set. However, as the potential function grows fast near the boundary of the Mandelbrot set, it was necessary for the creator to let the landscape grow downwards, so that it looked as if the Mandelbrot set was a plataeu atop a mountain with steep sides. The same technique was used a year after in some images in Peitgen & Richter: The Beauty of Fractals (1986).

In this book you can find a formula to estimate the distance from a point outside the Mandelbrot set to the boundary of the Mandelbrot set (and a similar formula for the Julia sets), and one can wonder why the creator did not use this function instead of the potential function, because it grows in a more natural way.

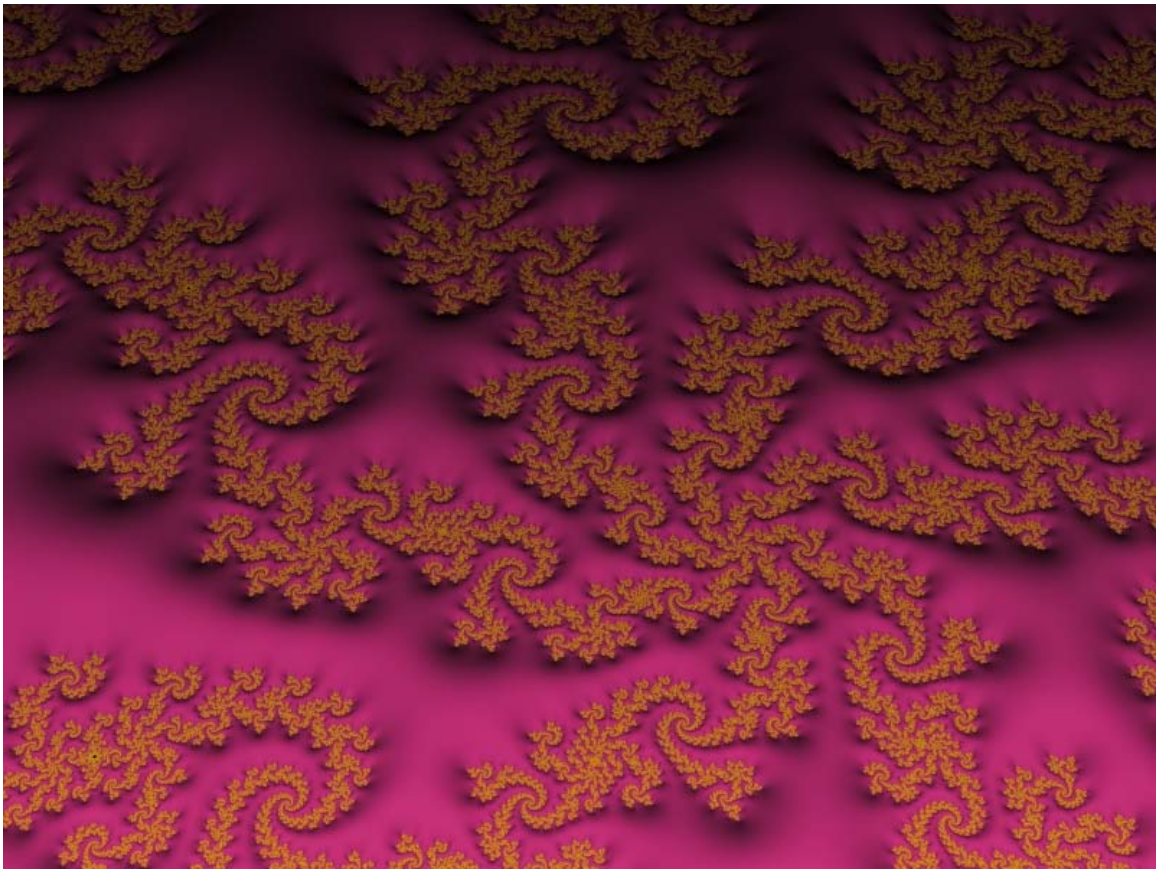
The three pictures show landscapes formed from the distance function for a family of iterations of the form $z^2 + az^4 + c$. If, in a light from the sun. Then we imagine the rays are parallel (and given by two angles), and we let the colour of a point on the surface be determined by the angle between this direction and the slope of the surface at the point. The intensity (on the earth) is independent of the distance, but the light grows whiter because of the atmosphere, and sometimes the ground looks as if it is enveloped in a veil of mist (second picture). We can also let the light be "artificial", as if it issues from a lantern held by the observer. In this case the colour must grow darker with the distance (third picture).



Natural light



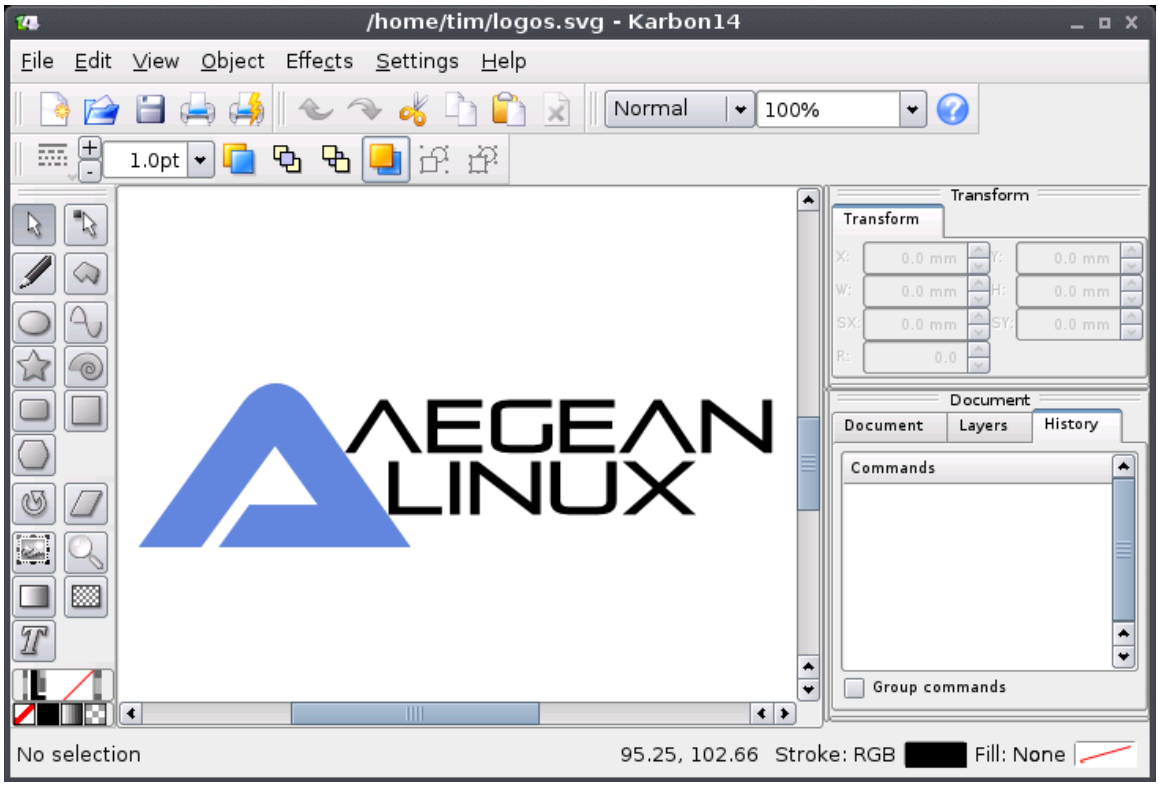
Natural light and mist



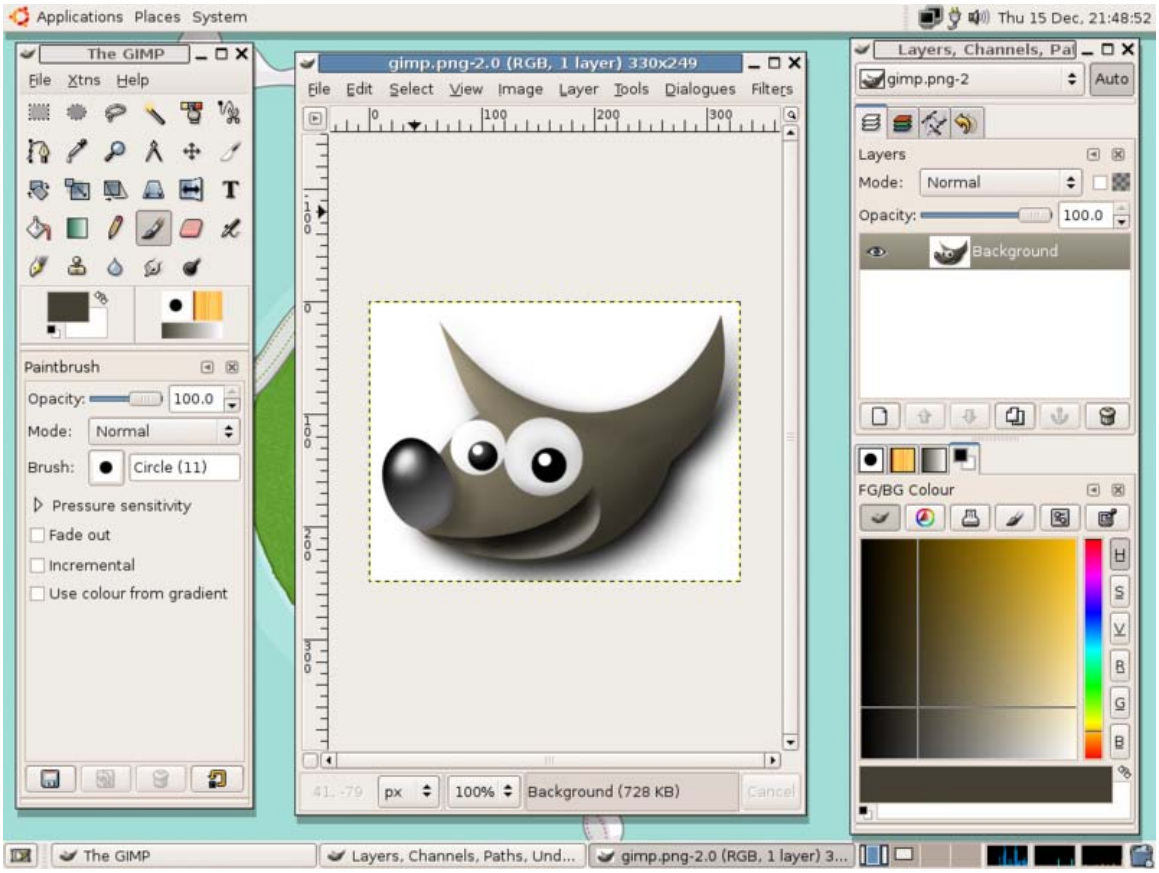
Artificial light

Chapter 5

Graphic Art Software



A screenshot of Karbon14 vector graphic software running on an AegeanLinux desktop



A screenshot of the GIMP 2.2.8 raster graphic software

Graphic art software is a subclass of application software used for graphic design, multimedia development, specialized image development, general image editing, or simply to access graphic files. Art software uses either raster or vector graphic reading and editing methods to create, edit, and view art.

Many artists and other creative professionals today use personal computers rather than traditional media. Using graphic art software may be more efficient than rendering using traditional media by requiring less hand–eye coordination, requiring less visualization skill, and utilizing the computer's quicker (sometimes more accurate) automated rendering functions to create images. However, advanced level computer styles, effects and editing methods may require a steeper learning curve of computer technical skills than what was required to learn traditional hand rendering and visualization skills. The potential of the software to enhance or hinder creativity may depend on the intuitiveness of the interface.

Specialized software

Most art software includes common functions, creation tools, editing tools, filters, and automated rendering modes. Many, however, are designed to enhance a specialized skill or technique.

Graphic design software

Graphic design professionals favor general image editing software and page layout software commonly referred to as desktop publishing software.

Multimedia development software

Multimedia development professionals favor software with audio, motion and interactivity such as software for creating and editing hypermedia, electronic presentations (more specifically slide presentations), computer simulations and games.

Image development software

Image development professionals may use general graphic editors or may prefer more specialized software. Although images can be created from scratch with most art software, specialized software applications or advanced features of generalized applications are used for more accurate visual effects. These visual effects include:

Traditional medium effects

Vector editors are ideal for solid crisp lines seen in line art, poster, woodcut ink effects, and mosaic effects.

Some generalized image editors, such as Photoshop are used for digital painting (representing real brush and canvas textures such as watercolor or burlap canvas) or handicraft textures such as mosaic or stained glass. However, unlike Photoshop, which was originally designed for photo editing, software such as Corel Painter and Photo-Paint were originally designed for rendering with digital painting effects and continue to evolve with more emphasis on hand-rendering styles that don't appear computer generated.

Photorealistic effects

Unlike traditional medium effects, photorealistic effects create the illusion of a photographed image. Specialized software may contain 3D modeling and ray tracing features to make images appear photographed. Some 3D software is for general 3D object modeling, whereas other 3D software is more specialized, such as Poser for characters or Bryce for scenery. Software such as Photoshop may be used to create 3D effects from 2D (flat) images instead of 3D models. AddDepth is a discontinued software for extruding 2D shapes into 3D images with the option of beveled effects. MetaCreations Detailer and Painter 3D are discontinued software applications specifically for painting texture maps on 3D Models.

Hyperrealistic effects

Specialized software may be used to combine traditional medium effects and photorealistic effects. 3-D modeling software may be exclusively for, include features for, or include the option of 3rd party plugins for rendering 3-D models with 2-D effects (e.g. cartoons, illustrations) for hyperrealistic effects. Other 2-D image editing software may be used to trace photographs or rotoscope animations from film. This allows artists to rapidly apply unique styles to what would be purely photorealistic images from computer generated imagery from 3-D models or photographs. Some styles of hyperrealism may require motion visual effects (e.g. geometrically accurate rotation, accurate kinetics, simulated organic growth, life-like motion constraints) to notice the realism of the imagery. Software may be used to bridge the gap between the imagination and the laws of physics.

Specialized graphic format handling

This may include software for handling specialized graphic file formats such as Fontographer software, which is dedicated to creating and editing computer fonts. Some general image editing software has unique image file handling features as well. Vector graphic editors handle vector graphic files and are able to load PostScript files natively. Some tools enable professional photographers to use nondestructive image processing for editing digital photography without permanently changing or duplicating the original, using the Raw image format. Other special handling software includes software for capturing images such as 2D scanning software, 3D scanning software and screen-capturing, or software for specialized graphic format processing such as raster image processing and file format conversion.

Chapter 6

Spriting and Pixel Art

Spriting

Spriting is usually defined as the act of creating partially transparent 2D raster graphics for use in video games, commonly referred to as sprites; by extension, it is also used to refer to the act of creating pixel art, though not all sprites are necessarily done in that style. Pixel art comprises a large part of "sprite art" as a whole; though technological advances since the mid-nineties allowed pre-rendered raytraced imagery, or essentially any 2-dimensional image style to be used as a sprite. In some communities, "pixel art" is considered a synonym of "sprite art", and classification of artwork as "sprite art" is held to the same standards, though pixel art itself is not limited to the creation of sprites.

Though sprites have been a major component of many early video games, the modern, "mainstream" activity called "Spriting" arose with the advent of widely-available computer graphics programs capable of editing and saving raster images. Such programs include MacPaint, and the later Microsoft Paint. The distinctive style of imagery used in many early computer and arcade games inspired people to create similar works of their own. Having the tools available to do this allowed many people to experiment with what was previously a prohibitively difficult-to-enter craft. With the advent of the internet, practitioners of spriting were able to collaborate and share their creations, which established spriting as a proper hobby, and also exposed these artists to prospective employers, and vice-versa.

Spriting is primarily done for the direct purpose of creating video game artwork, especially by professional artists. By hobbyists, though, it is often done to create stand-alone artwork, or as part of a larger piece of art, such as a web comic. The use of sprites as cookie-cutter elements of comic strips has led to a genre called sprite comics.

Spriting as a hobby

Modified commercial sprites

Despite copyright concerns, many hobbyists new to spriting begin their work by editing sprite imagery made for commercial video games; often games seen on console platforms like the SNES or Sega Genesis. Fans of these images collect transcribed copies of them in

common image formats, and post them on websites for others to see. This process of extracting the imagery is called "ripping" or "dumping". In "ripping" a person collects the imagery via screen captures of an emulator running the source game - this practice is in enough demand that some emulators, like ZSNES, have a feature to optionally display only desired layers of the game's imagery, making it easier to copy. "Dumping" involves a more sophisticated way of directly extracting the images from the game; this is often rather difficult, since on systems like the SNES, the larger images seen on-screen are stored in several smaller parts. These images are collected into compilations known as sprite sheets, large raster images which each hold all of the frames associated with a single character, or a single terrain environment. These are the de facto standard for trading ripped commercial sprites online.

Those who edit these images, which generally depict characters in the game, often begin by simply shifting the color palette; thus turning all of a characters primary costume color into a different color. Later, they will redraw small parts of the image, making slight changes to the costuming, etc. This is generally much less intimidating than creating a full, original work, and allows them practice in imitating and matching the styles used by professional artists. This progresses to modifications which are sufficiently extensive as to make the source work they were derived from, unrecognizable.

Revamping (Reshading)

Some spriters specialize in renewing the colors of old game sprites to make them look fresh. This process is called revamping.

Most revampers choose an old sprite and replace the colors with those of a more recent one. Others recolor them with a set color palette.

There is also a form of spriting called devamping, considered the opposite of revamping. It is the process of changing the color of a new sprite to make them look older.

A good example of Revamping/Reshading is used in the crossover fan game Mushroom Kingdom Fusion.

Recolors

Recolors of sprites are when you take a sprite, and change the pallette to make the sprite a different color.

This is unprofessional, and only used as a hobby to some who don't edit or customize sprites.

Styles of spriting

A sprite style is the way a certain sprite is made to look. Each style has methods and rules that must be followed in order to keep sprites uniform, even if they represent different

characters entirely. Sprites of different styles often clash with each other, but mixing any style is never accepted. There are many different styles, even in games of the same series; some spriters also create their own styles.

Sprite styles may vary by size, palette, shading, detail, and further artistic touches.

Original sprites

With sufficient skill, often drawing from many general aspects of illustration (shading, color theory, foreshortening, and often comic art), an artist can create professional-quality sprite images and pixel art from scratch.

Artists will make sprites-to-order as a hobby, for either themselves, or friends. These sprites, created completely from scratch, but generally imitating sprites from a commercial game, are called "Custom" sprites, and are often seen as avatars, buddy icons, small animations, and elements of web comics. They will also often be used for small-scale video game projects.

Original Sprites are also used for advertisement. Sometimes you will see a spriter, or a small group of spriters, use a specific icon or logo made entirely from scratch. Most spriters that sprite for others yet do not have a group of spriters to work with them are called either Spriters-For-Hire, or Lone Spriters. After a while, once these spriters become more popular around forums and sometimes even YouTube, their icon or logo will become famous and well-known throughout the Internet.

Spriting as a profession

With the advent of dedicated 3D hardware, sprites lost their near-monopoly on video-game imagery. However, sprite art remains a popular medium for video-game imagery, and has witnessed a comeback in the commercial markets for handheld consoles (like the Game Boy Advance and Nintendo DS) or mobile phone games. The companies that create these games hire many of the best sprite artists who maintain online portfolios of their work, and who openly seek such employment. Recently a demand for spriters has opened up in the MMORPG field, for games such as Habbo Hotel.

Professional sprite artists often delve into other areas of illustration, including animation, comics, and computer-generated art, often doing those on a professional basis as well.

Tools

Spriting needs only a computer and a graphic editing program, although other tools can help. Depending on the target use of the sprite, it may be necessary to have a program capable of adding transparency information (e.g. an "alpha channel") to sprites. For handheld/mobile games, it is also necessary to post-process the imagery, converting it to a format native to the hardware the game will run on. When editing pixel images, a professional artist will typically use either a program designed for sprite-editing and

animation, and/or will modify the workspace of a major image-editor like Photoshop to accommodate sprite editing. The following programs are prevalent:

Software

The needs of pixel art are very different from those of high-resolution illustration, of image retouching and of digital photo management; not all image editors are suitable for spriting.

Common programs

- Microsoft Paint is included in Microsoft Windows. It lacks many advanced features, but basic graphic editing is very mature and suitable for pixel by pixel editing.

Programs designed for sprite editing

- GraphicsGale, for Windows, has both freeware and ¥1995 (roughly $20) shareware versions, and has features designed for animation and cursor creation.
- Pro Motion is available for Windows, and costs $78. It is often used by professionals, and offers many animation features, and features tailored to producing art for the Game Boy Advance, or Mobile Phones.
- Pixen is free, open-source, and available for Mac OS X. It offers many tools tailored to sprite creation and animation.

Major image editors

- The GIMP is a free, open-source image editor, available for all major platforms. The GIMP-GAP plugin greatly enhances its animation capabilities.
- Adobe Photoshop is a commercial image editor available for Windows and Mac OS X. In order to make a sprite in this program, users mostly have to use the pencil tool.
- Paint.NET, built with the .NET framework, is open-source freeware capable of similar functionality as Adobe Photoshop and a number of professional tools.
- GrafX2 is an open-source picture editor very similar to Deluxe Paint and others from 80's and 90's.

Hardware

A scanner can be useful for transferring penciled sketches/designs of a sprite into a computer. Advanced sprite artists often create their sprites directly on a computer although, a graphics tablet can greatly ease the majority of the work; especially the sketching and blocking stages, although the finer details will inevitably need to get tweaked with the mouse.

Methodology

There are several ways to create a sprite. Virtually all of these involve some way of "planning out" the form of the final sprite image. By laying out simple, and quick to draw suggestions of the final form, an artist can evaluate and correct the general direction that the construction of an image is moving towards, eliminating errors in perspective, anatomy, and foreshortening before the major work of drawing begins. These methods are very similar to other forms of illustration.

"Blocking" is a term used to describe what painters do - a spriter will lay out large regions of color/shade, trying to lay them in what he will think to be their final position, but starting with the biggest regions first. The artist will then sculpt these regions into the correct shape, and add in additional highlights and shadows to form the image, sometimes finishing the image by adding cartoon-like outlines to frame parts of the image.

"Line Art" is the outlining used to draw many styles of illustration. These are usually shades of black (Pure black or dark grey, usually) but this is not always true. It can stand on its own without coloration, and many spriters will draw black and white outlines of their intended final images before filling them in with color, and shading those filled regions. More experienced sprite artists will often color most of this line art; giving it a darker hue of the region it surrounds, rather than using jet black for all of their outlines.

After any of these methods are followed, the sprite artist will clean the image of any extraneous pixels unrelated to their illustration, apply proper transparency, especially to the blank canvas surrounding their illustration, and then export the image to a format suitable for sending to their client or friends. The PNG and to a lesser degree, GIF formats are suitable, and JPEG is notable for not being so - the lossy nature of JPEG compression often mars some of the finer details of sprites, but more importantly, can shift the colors of a sprite's pixels. Many videogames that use sprites require the sprites to use an exact, predefined palette of colors, and often create graphical effects by shifting these colors; any colors outside of this scheme will cause this to fail. Specific projects may convert the images to a proprietary format used internally by their game.

Pixel art

Pixel art is a form of digital art, created through the use of raster graphics software, where images are edited on the pixel level. Graphics in most old (or relatively limited) computer and video games, graphing calculator games, and many mobile phone games are mostly pixel art.

History

The term *pixel art* was first published by Adele Goldberg and Robert Flegal of Xerox Palo Alto Research Center in 1982. The concept, however, goes back about 10 years before that, for example in Richard Shoup's SuperPaint system in 1972, also at Xerox PARC.

Some traditional art forms, such as counted-thread embroidery (including cross-stitch) and some kinds of mosaic and beadwork, are very similar to pixel art. These art forms construct pictures out of small colored units similar to the pixels of modern digital computing. A similar concept on a much bigger scale can be seen in the mass games.

Definition

Image filters (such as blurring or alpha-blending) or tools with automatic anti-aliasing are considered not valid tools for pixel art, as such tools calculate new pixel values automatically, contrasting with the precise manual arrangement of pixels associated with pixel art.

Techniques

Drawings usually start with what is called the line art, which is the basic line that defines the character, building or anything else the artist is intending to draw. Linearts are usually traced over scanned drawings and are often shared among other pixel artists. Other techniques, some resembling painting, also exist.

The limited palette often implemented in pixel art usually promotes dithering to achieve different shades and colors, but due to the nature of this form of art this is done completely by hand. *Hand-made* anti-aliasing is also used.

Here are a few parts of the above image of “The Gunk” in detail, depicting a few of the techniques involved:

1. The basic form of dithering, using two colors in a 2×2 checkerboard pattern. Changing the density of each color will lead to different subtones.

2. Stylized dithering with 2×2 pixel squares randomly scattered can produce interesting textures. Small circles are also frequent.

3. Anti-aliasing can be done, by hand, to smooth curves and transitions. Some artists only do this internally, to keep crisp outlines that can go over any background. The PNG alpha channel can be used to create external anti-aliasing for any background.

Saving and compression

Pixel art is preferably stored in a file format utilizing lossless data compression, such as run-length encoding or an indexed color palette. GIF and PNG are two file formats commonly used for storing pixel art. The JPEG format is avoided because its lossy compression algorithm is designed for smooth continuous-tone images and introduces visible artifacts in the presence of dithering.

GIF file (318 bytes)

PNG file (254 bytes)

JPEG file (706 bytes)

Magnified JPEG to show artifacts

Categories

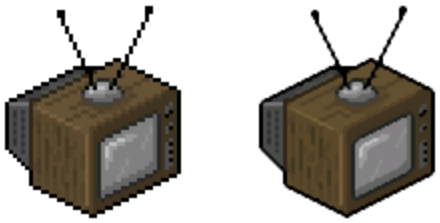
Isometric

Non-isometric

Pixel art is commonly divided in two subcategories: isometric and non-isometric. The isometric kind is drawn in a near-isometric dimetric projection. This is commonly seen in games to provide a three-dimensional view without using any real three-dimensional processing. Technically, an isometric angle would be of 30 degrees from the horizontal, but this is avoided since the pixels created by a line drawing algorithm would not follow a neat pattern. To fix this, lines with a 1:2 pixel ratio are picked, leading to an angle of about 26.6 degrees (arctan 0.5).

Non-isometric pixel art is any pixel art that does not fall in the isometric category, such as views from the top, side, front, bottom or perspective views. These are also called Planometric views.

Scaling



2x zoom interpolated using the 2xSaI algorithm

When pixel art is displayed at a higher resolution than the source image, it is often scaled using the nearest neighbor interpolation algorithm. This avoids blurring caused by other algorithms, such as bilinear and bicubic interpolation—which interpolate between adjacent pixels and work best on continuous tones, but not sharp edges or lines. Nearest-neighbor interpolation preserves these sharp edges, but it makes diagonal lines and curves look blocky, an effect called pixelation. Thus, hybrid algorithms have been devised to interpolate between continuous tones while preserving the sharpness of lines in the piece; such attempts include the 2xSaI and Super Eagle algorithms.

Uses

Pixel art was very often used in older computer and video console games. With the increasing use of 3D graphics in games, pixel art lost some of its use. Despite that, this is still a very active professional/amateur area, since mobile phones and other portable devices still have low resolution and then require a skillful use of space and memory. Sometimes pixel art is used for advertising too. One such company that uses pixel art to advertise is Bell. The group eboy specializes in pixel graphics for advertising and has been featured in magazines such as *Wired*, *Popular Science*, and *Fortune 500*.

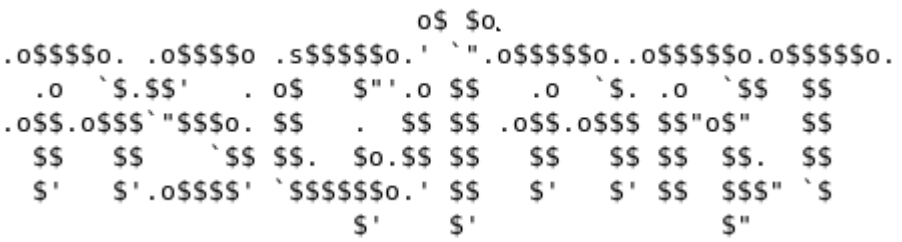
Icons for operating systems with limited graphics abilities are also pixel art. The limited number of colors and resolution presents a challenge when attempting to convey complicated concepts and ideas in an efficient way. On the Microsoft Windows desktop icons are raster images of various sizes, the smaller of which are not necessarily scaled from the larger ones and could be considered pixel art. On the GNOME and KDE desktops, icons are represented primarily by SVG images, but with hand-optimized, pixel art PNGs for smaller sizes such as 16x16 and 24x24. Another use of pixel art on modern desktop computers is favicons.

Modern pixel art has been seen as a reaction to the 3D graphics industry by amateur game/graphic hobbyists. Many retro enthusiasts often choose to mimic the style of the past. Some view the pixel art revival as restoring the golden age of second and third generation consoles, where it is argued graphics were more aesthetically pleasing. Pixel art still remains popular and has been used in the virtual worlds Citypixel and Habbo as well as among hand-held devices such as the Nintendo DS and Cellphones.

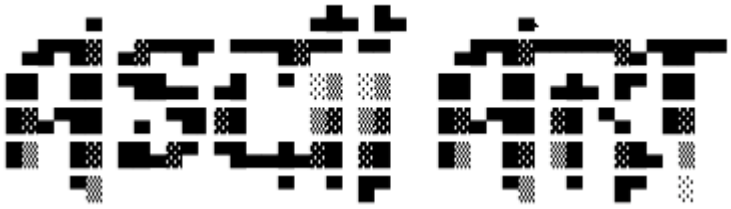
Chapter 7

Introduction to ASCII Art

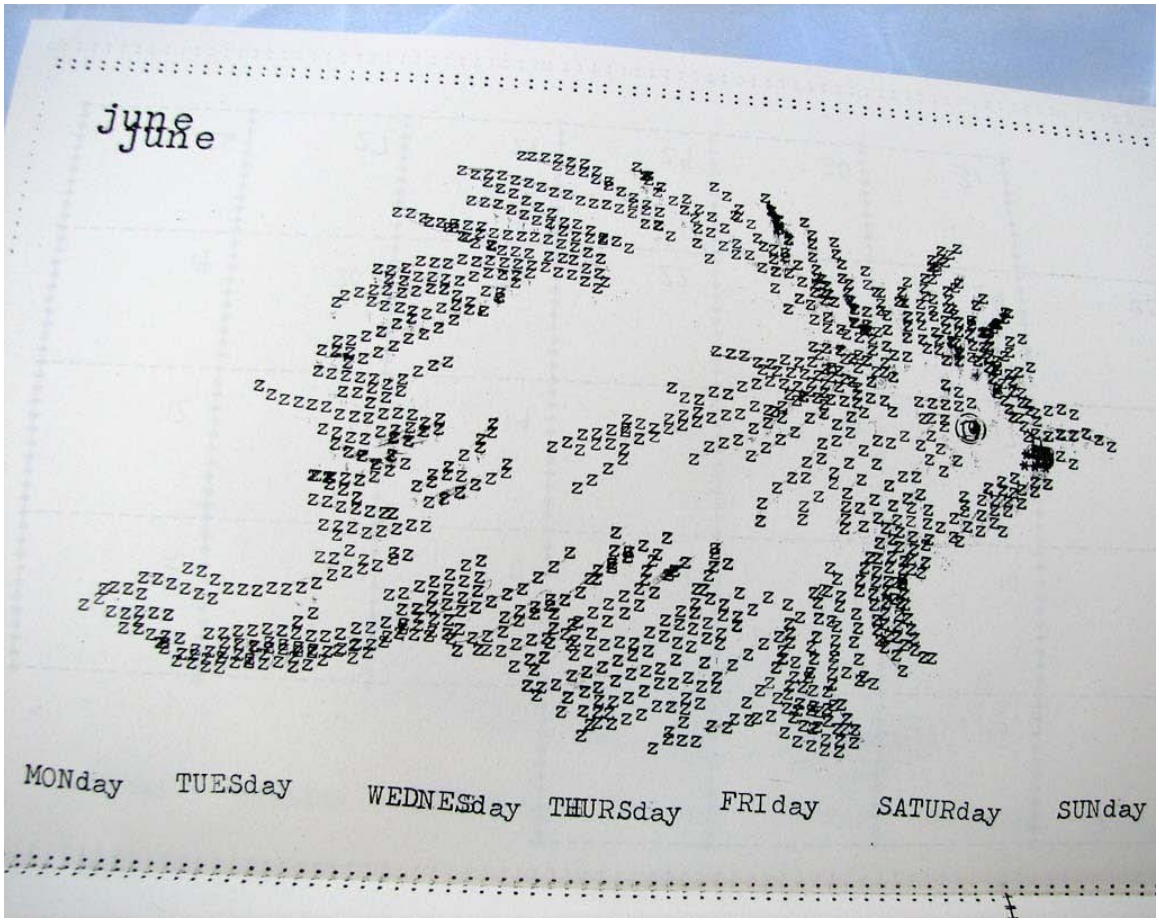
"Oldskool" or "Amiga" style

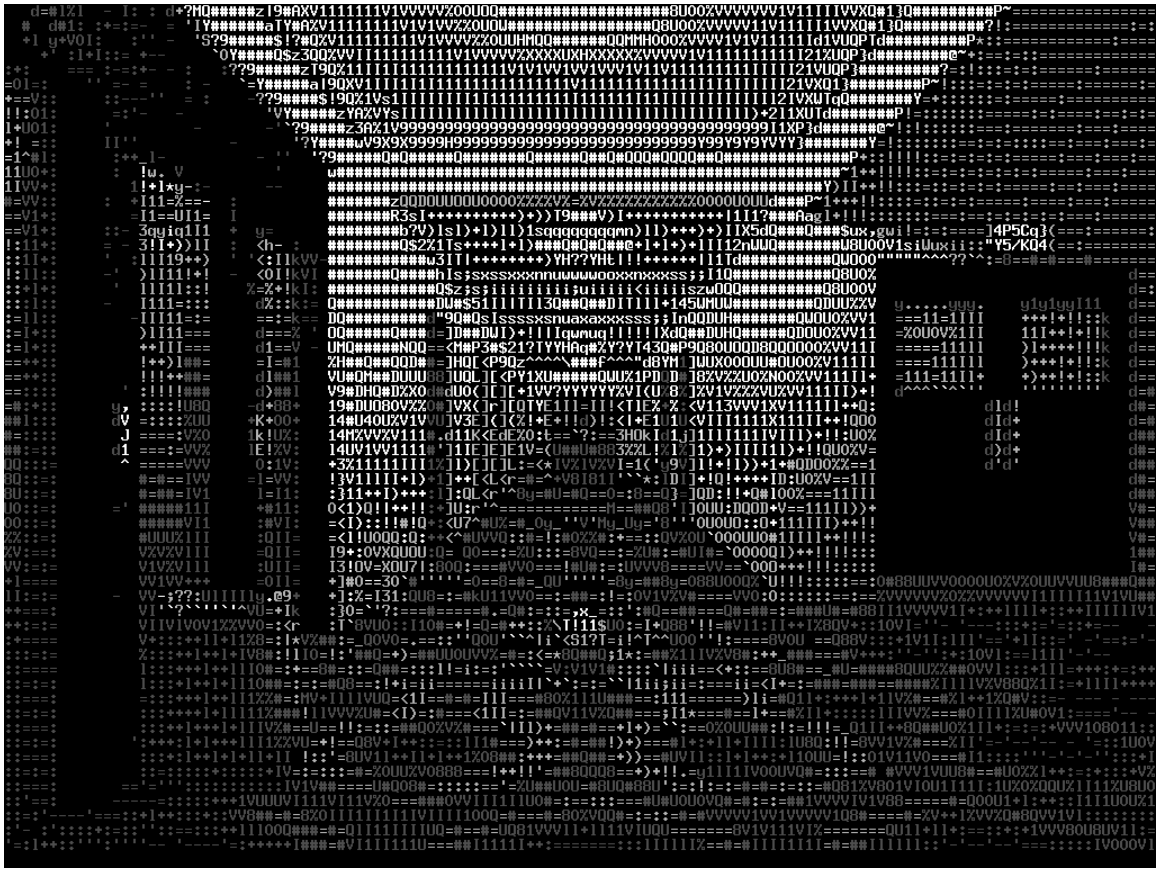


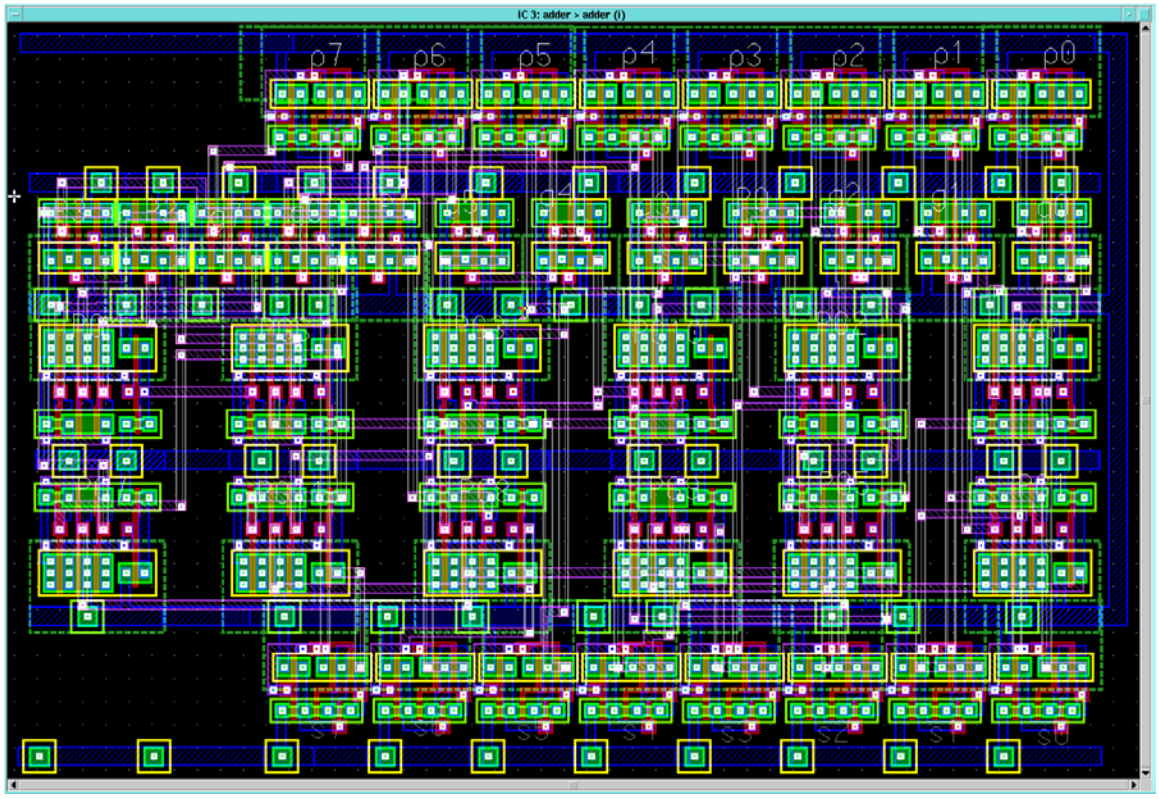
"Newskool" style



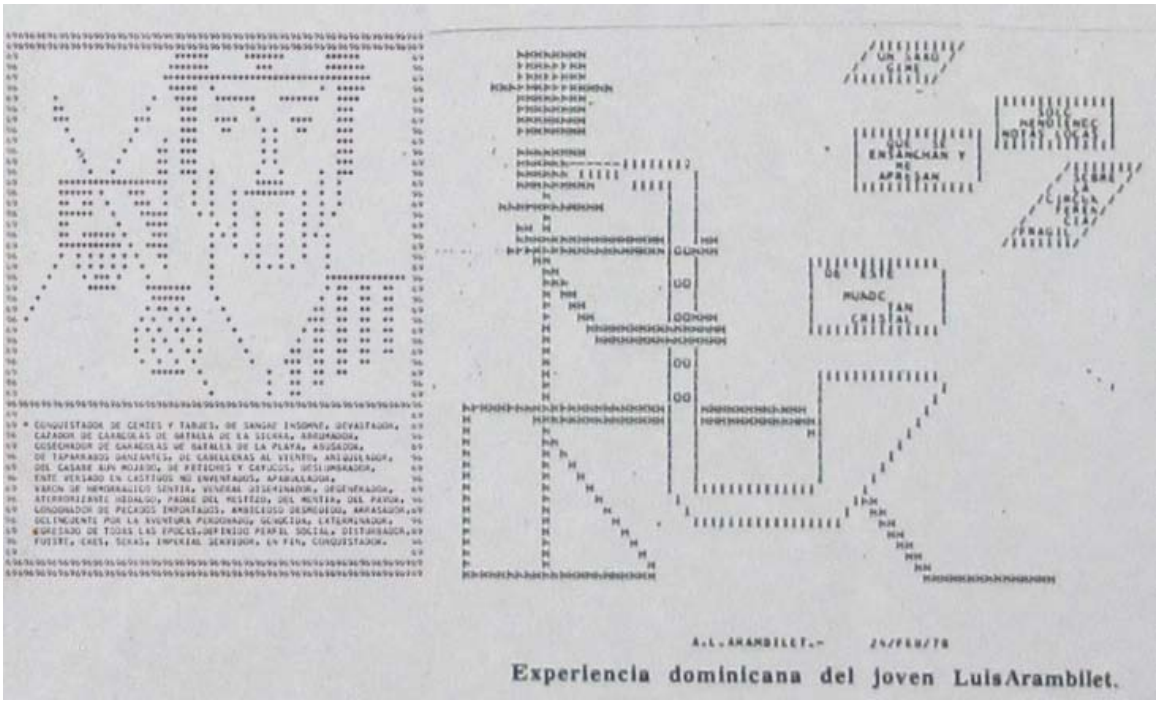
"Block" or "High ASCII" style, cf. ANSI art


june
MONday
TUESday
WEDNESday
THURSday
FRIday
SATURday
SUNday

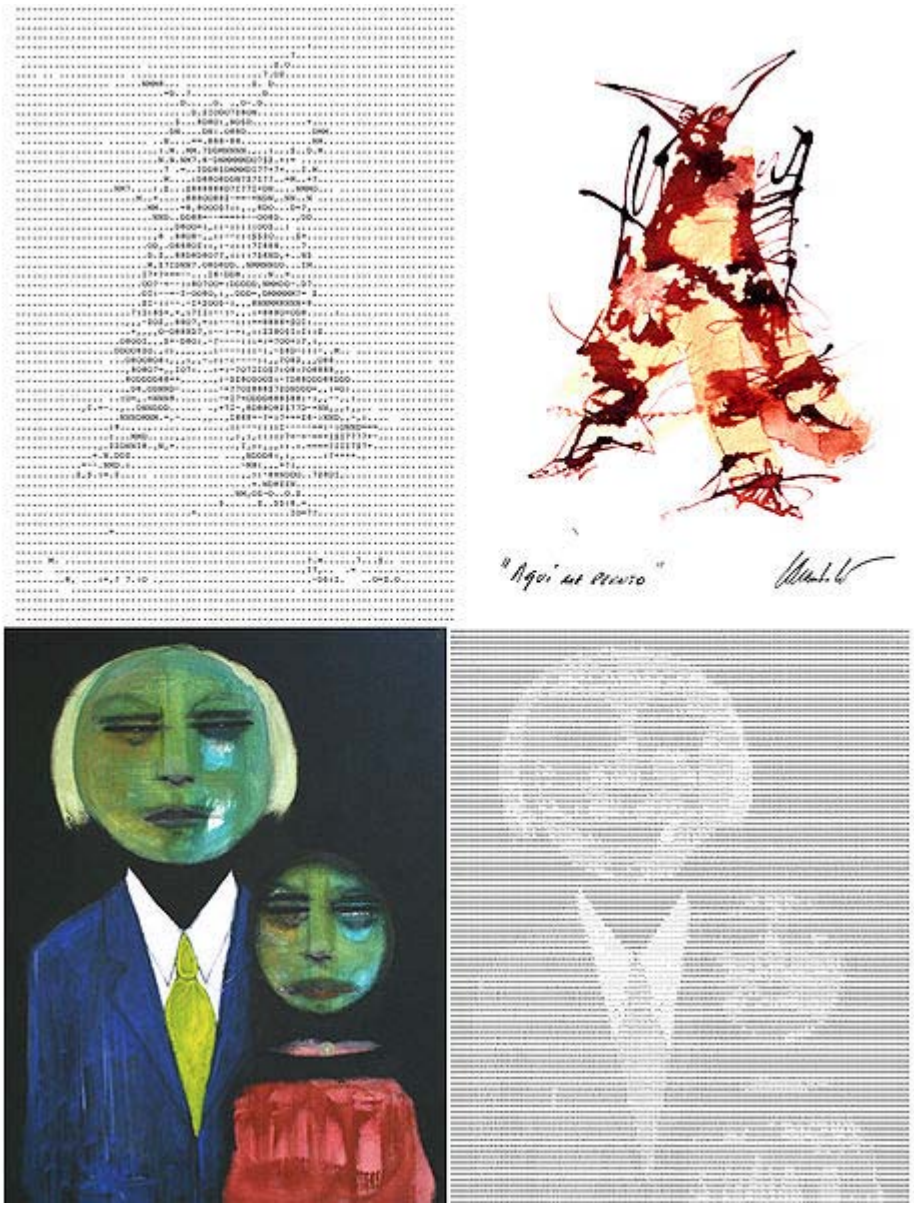



IC 3: adder > adder (i)
p7
p6
p5
p4
p3
p2
p1
p0

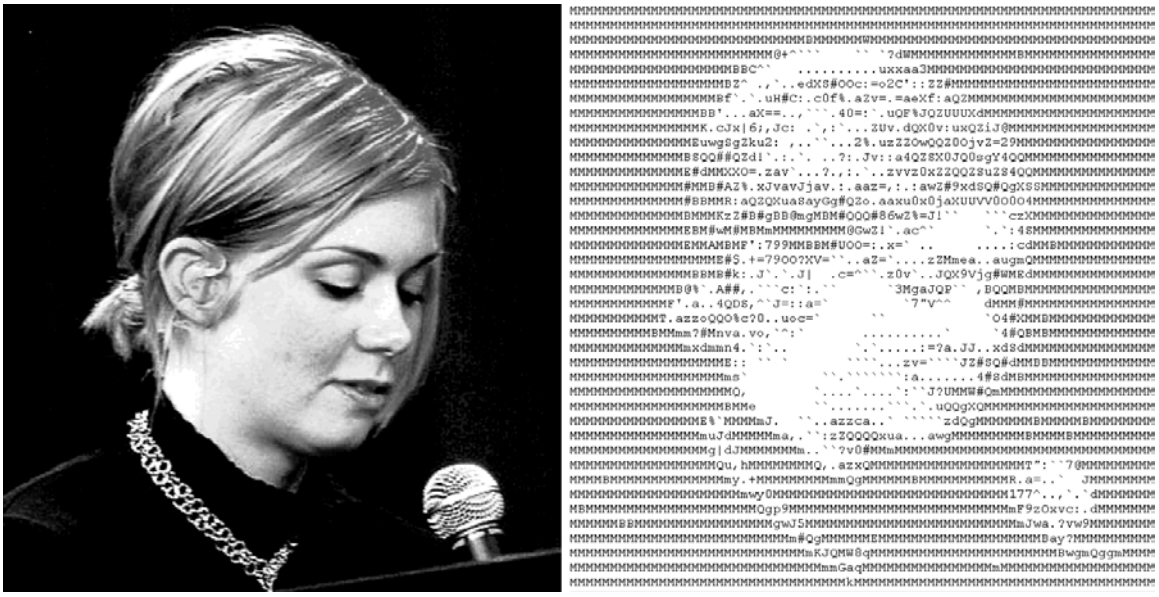
ASCII art is a graphic design technique that uses computers for presentation and consists of pictures pieced together from the 95 printable (from a total of 128) characters defined by the ASCII Standard from 1963 and ASCII compliant character sets with proprietary extended characters (beyond the 128 characters of standard 7-bit ASCII). The term is also loosely used to refer to text based art in general. ASCII art can be created with any text editor, and is often used with free-form languages. Most examples of ASCII art require a fixed-width font (non-proportional fonts, as on a traditional typewriter) such as Courier for presentation.

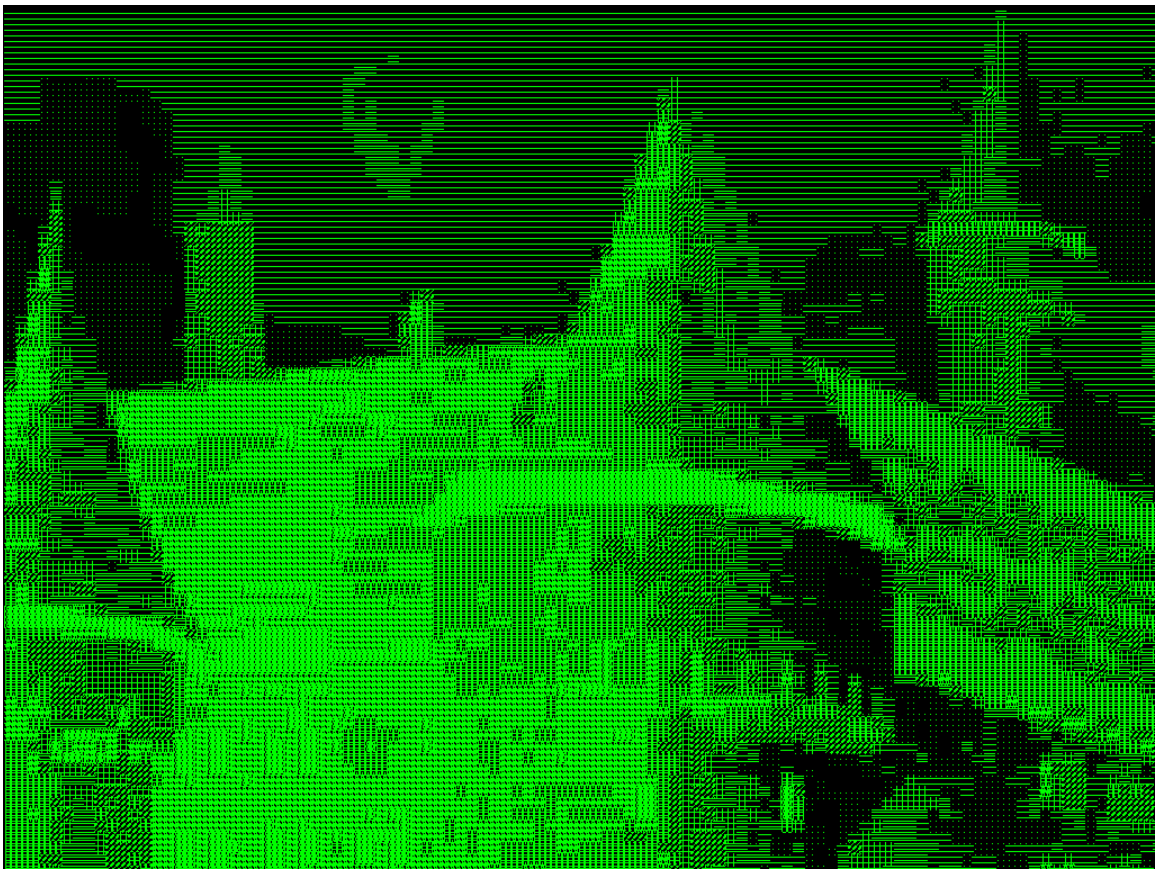


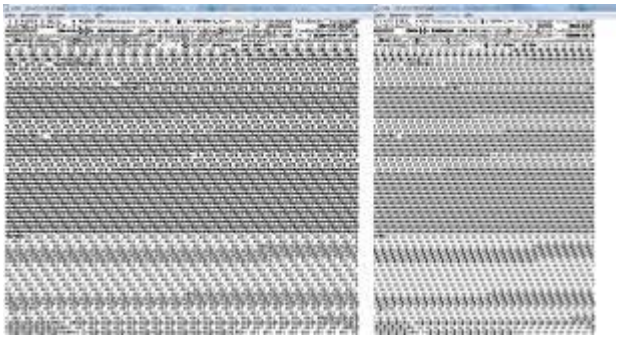
Arambilet:*ASCII ART Conqueror/Saxophonist Created in 1975 with 80 column punched cards, IBM 370-115 CPU, IBM 3203 printer. Published in February 1978 by El Caribe newspaper, section: Arts and Cybernetics".*



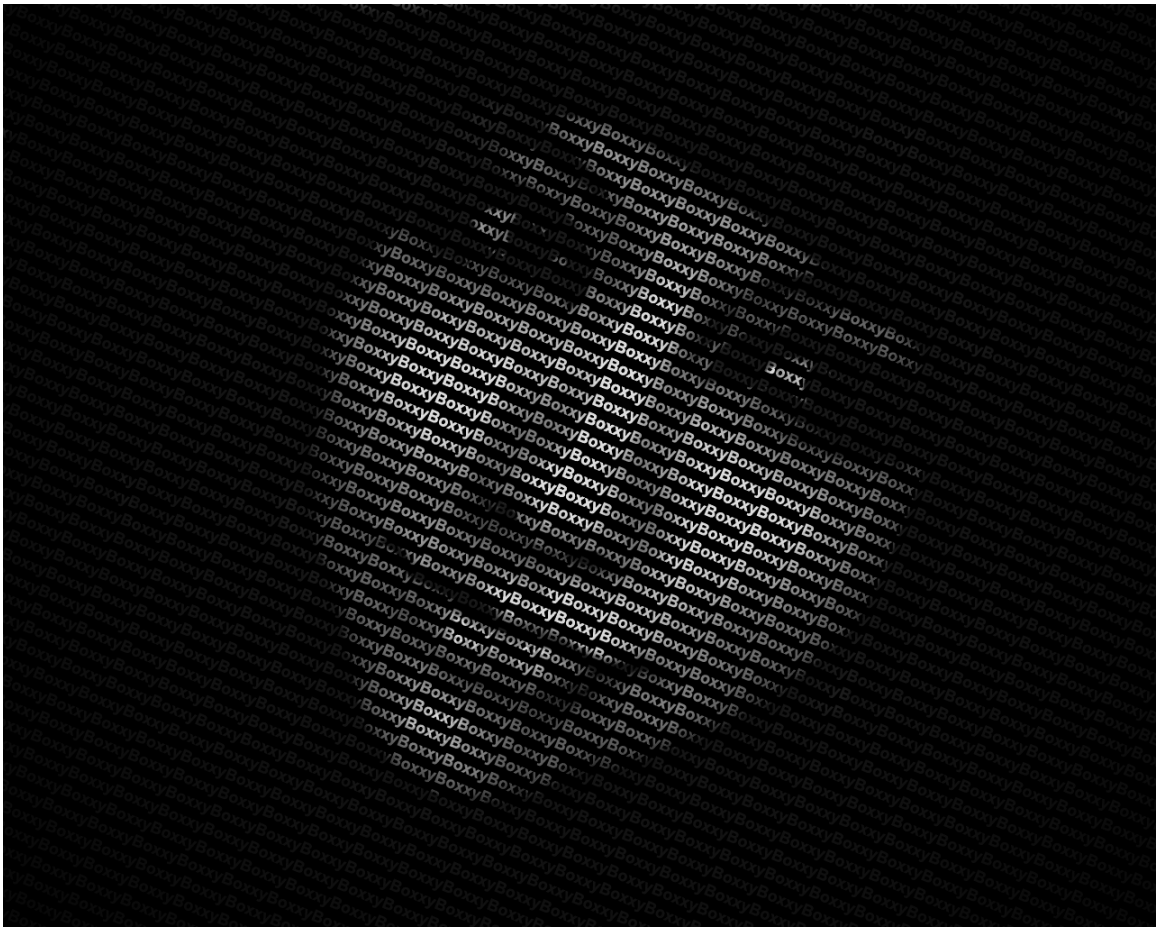
Arambilet:*ARTE ASCII/ASCII ART*







Among the oldest known examples of ASCII art are the creations by computer-art pioneer Kenneth Knowlton from around 1966, who was working for Bell Labs at the time. "Studies in Perception I" by Ken Knowlton and Leon Harmon from 1966 shows some examples of their early ASCII art.



One of the main reasons ASCII art was born was because early printers often lacked graphics ability and thus characters were used in place of graphic marks. Also, to mark divisions between different print jobs from different users, bulk printers often used ASCII art to print large banners, making the division easier to spot so that the results could be more easily separated by a computer operator or clerk. ASCII art was also used in early e-mail when images could not be embedded.

History

Typewriter art

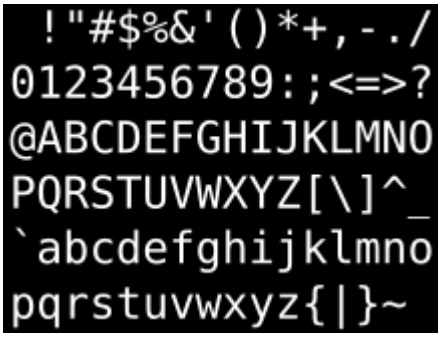
Since 1867 typewriters have been used for creating visual art. The oldest known preserved example of typewriter art is a picture of a butterfly made in 1898 by Flora Stacey.

In the 1954 short film *Stamp Day for Superman*, typewriter art was a feature of the plot.

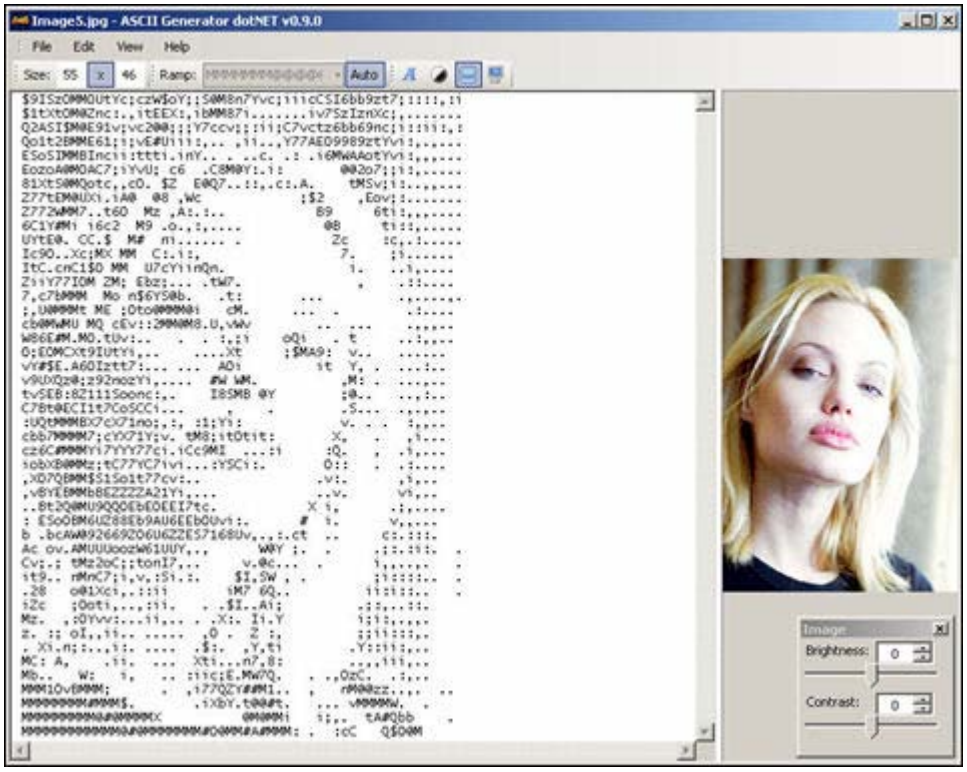
TTY and RTTY

TTY stands for "TeleTYpe" or "TeleTYpewriter" and is also known as Teleprinter or Teletype. RTTY stands for Radioteletype. According to a chapter in the "RTTY Handbook", text images have been sent via teletypewriter as early as 1923. However, none of the "old" RTTY art has been discovered yet. What is known is that text images appeared frequently on radio teletype in the 1960s and the 1970s.

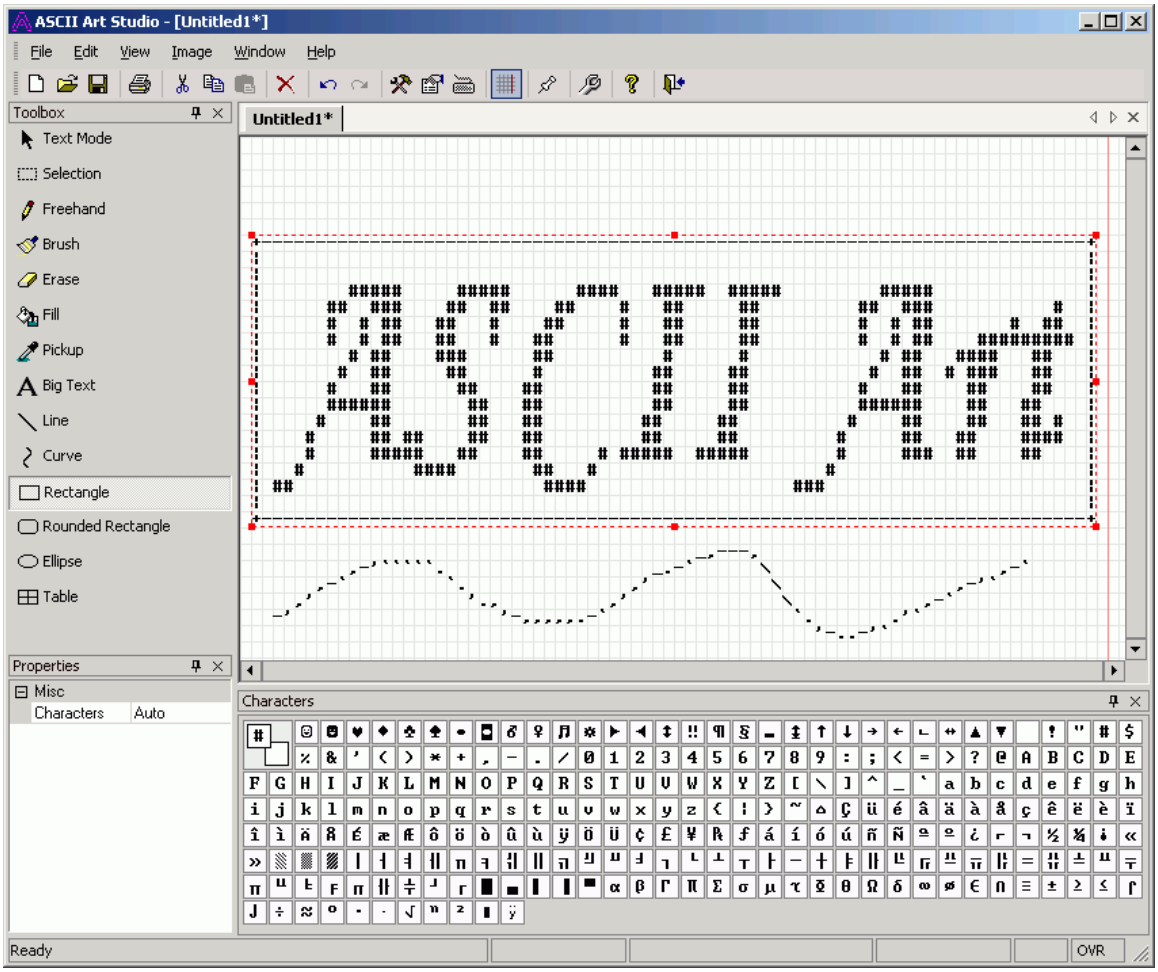
ASCII art

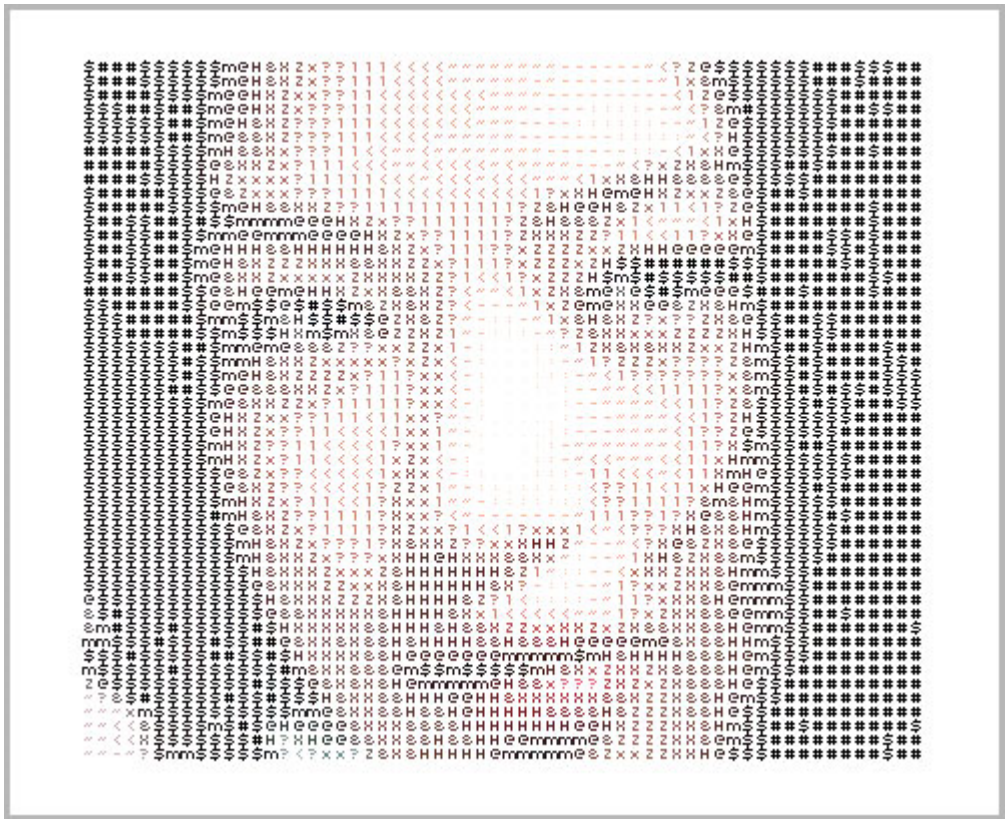


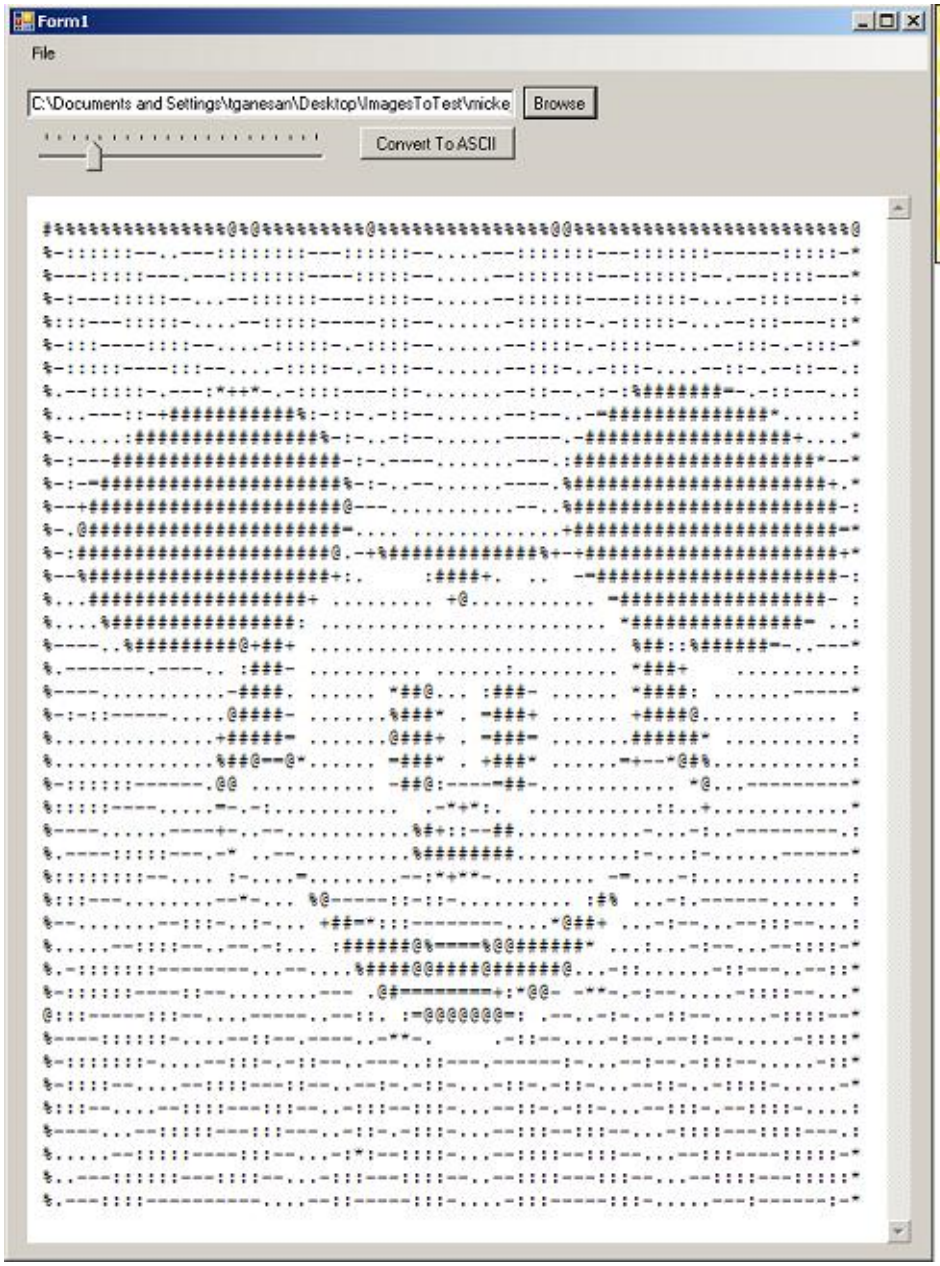
There are 95 printable ASCII characters, numbered 32 to 126.



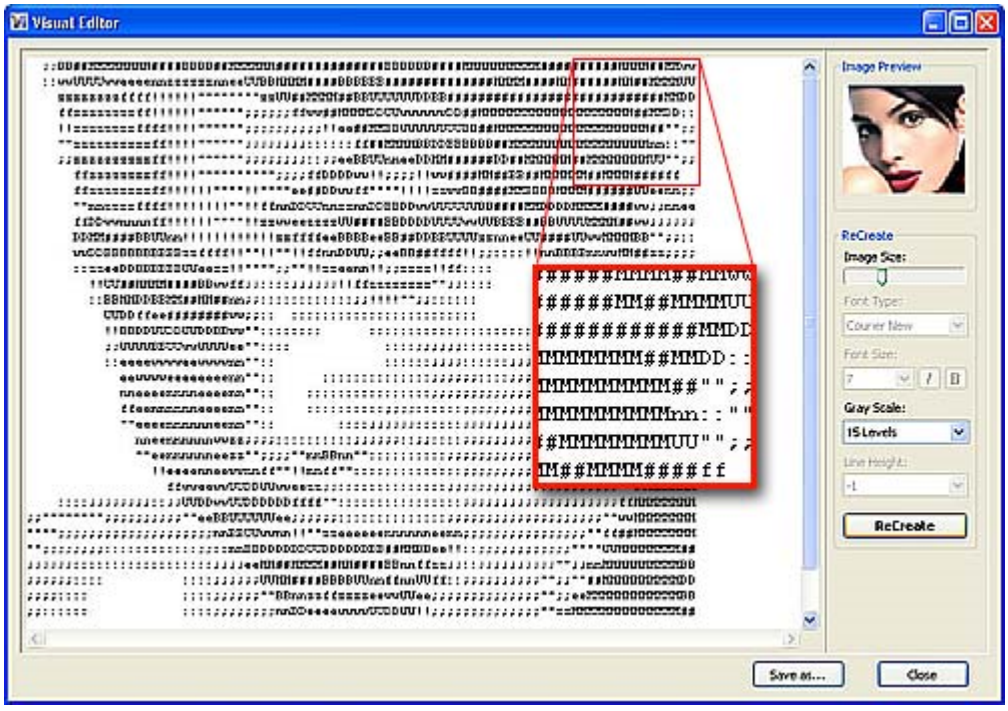
The widespread usage of ASCII art can be traced to the computer bulletin board systems of the late 1970s and early 1980s. The limitations of computers of that time period necessitated the use of text characters to represent images. Along with ASCII's use in communication, however, it also began to appear in the underground online art groups of the period. An ASCII comic is a form of webcomic which uses ASCII text to create images. In place of images in a regular comic, ASCII art is used, with the text or dialog usually placed underneath.


ASCII Art Studio - [Untitled1*]
File
Edit
View
Image
Window
Help
Toolbox
Text Mode
Selection
Freehand
Brush
Erase
Fill
Pickup
Big Text
Line
Curve
Rectangle
Rounded Rectangle
Ellipse
Table
Untitled1*
Properties
Misc
Characters
Auto
Characters
Ready
OVR

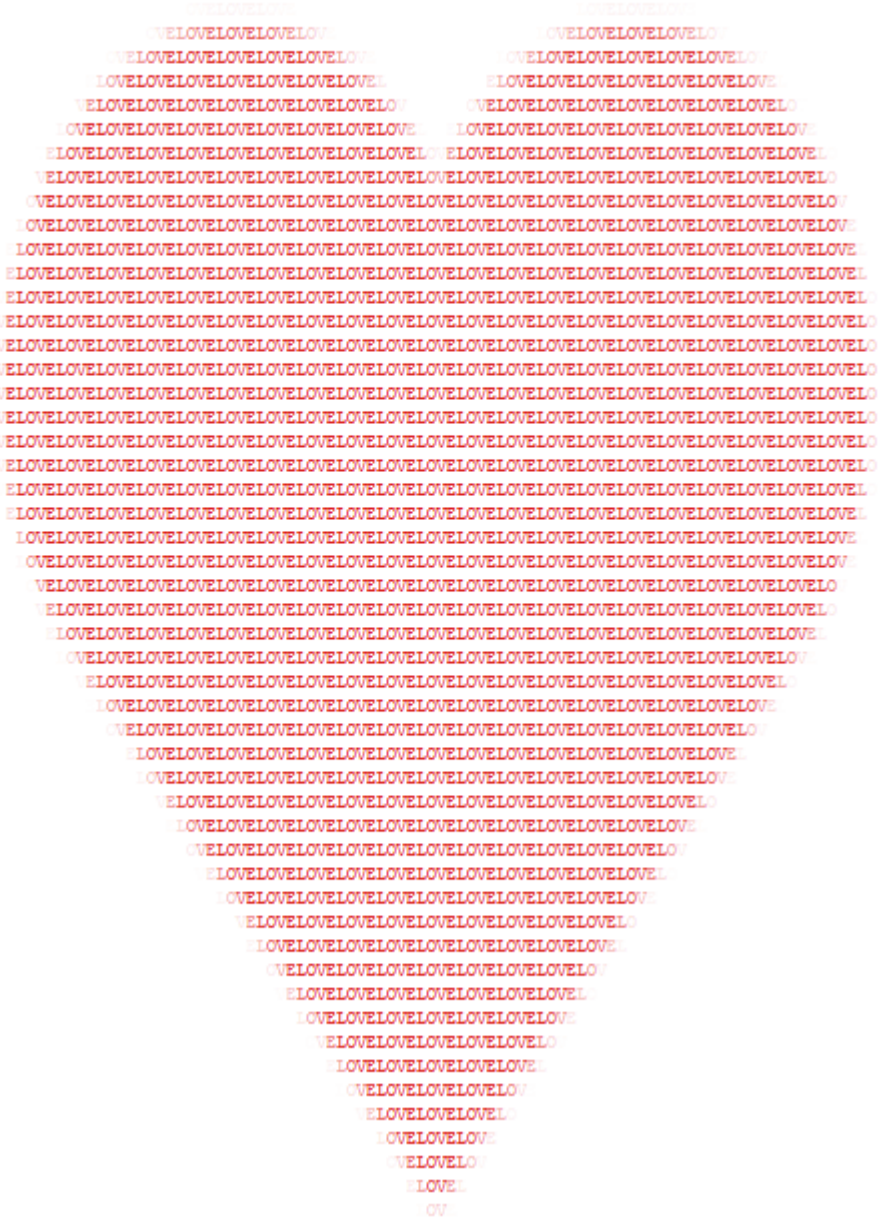




During the 1990s, graphical browsing and variable-width fonts became increasingly popular, leading to a decline in ASCII art. Despite this, ASCII art continued to survive through online MUDs, an acronym for "Multi-User Dungeon", (which are textual multiplayer roleplaying games), Internet Relay Chat, E-mail, message boards and other forms of online communication which commonly employ the needed fixed-width.

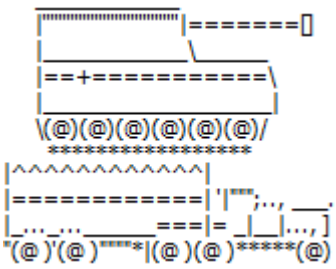

Visual Editor
Image Preview
ReCreate
Image Size:
Font Type:
Courier New
Font Size:
Gray Scale:
15 Levels
Line Height:
ReCreate
Save as...
Close

Text-image:



Over the years, warez groups began to get into the ASCII art scene. Warez groups usually release .nfo files with their software, cracks or general illegal software reverse-engineering releases. The ASCII art will usually include the warez group's name and maybe some ASCII borders on the outsides of the release notes, etc.

Uses



A self-propelled gun and truck made using ASCII art

ASCII art is used wherever text can be more readily printed or transmitted than graphics, or in some cases, where the transmission of pictures is not possible. This includes typewriters, teletypes, non-graphic computer terminals, printer separators, in early computer networking (e.g., BBSes), e-mail, and Usenet news messages. ASCII art is also used within the source code of computer programs for representation of company or product logos, and flow control or other diagrams. In some cases, the entire source code of a program is a piece of ASCII art — for instance, an entry to one of the earlier International Obfuscated C Code Contest is a program that adds numbers, but visually looks like a binary adder drawn in logic ports.

Examples of ASCII-style art predating the modern computer era can be found in the June 1939, July 1948 and October 1948 editions of Popular Mechanics.

"0verkill" is a 2D platform multiplayer shooter game designed entirely in colour ASCII art. MPlayer and VLC media player can display videos as ASCII art. ASCII art is used in the making of DOS-based ZZT games.

Many game walkthrough guides come as part of a basic .txt file; this file often contains the name of the game in ASCII art. Such as below, word art is created using backslashes and other ASCII values in order to create the illusion of 3D.

Types and styles

Different techniques could be used in ASCII art to obtain different artistic effects:

```
.--.      /\
'--'     /_\     (^..^)~
```

- Line art, for creating shapes

```
.g@8g.  db
'Y8@P' d88b
```

- Solid art, for creating filled shapes

```
:$#$: "4b. ':.
:$#$:   "4b. ':.
```

- Shading, using different symbol density and shading for creating gradients or contrasts

```
  |\_/|          ****************************
 / @ @ \         *  "Purrrfectly pleasant"  *
( > º < )        *       Poppy Prinz        *
 `»»x««´         *       (pprinz@...)       *
 /  O  \         ****************************
```

- combinations used as SIG (signature) at the end of an email

Emoticons and verticons

The simplest forms of ASCII art are combinations of two or three characters for expressing emotion in text. They are commonly referred to as 'emoticon', 'smilie', or 'smiley'.

There is another type of one-line ASCII art that does not require the mental rotation of pictures, which is widely known in Japan as kaomoji (literally "face characters".) Traditionally, they are referred to as "ASCII face ".

More complex examples use several lines of text to draw large symbols or more complex figures.

Popular smileys

The list only shows some popular examples for demonstration purposes. Hundreds of different text smileys were developed over time, but only a few were generally accepted, used and understood.

Icon	Meaning	Icon	Meaning
`:) or :] or (: or [: or =) or =] or (",) or ^^ or C: or :3 or`	classic smile	`;| or ;) or ;D or ^.~`	winking

n_n			
:(=(:C	classic sad	:-o	yawn or surprised
:-)	smile	XD or xD	Laughter
>=D	evil laugh		
>=)	evil grin	:o or O_o or O.o or O.O or ._.	surprised
:B	buck-tooth	:,(or :*(or :'(or :_(or ='(or =,(	crying smiley
:-#	with braces or sick smiley	T_T or TT_TT or QQ or ;_; or ;-; or ;A; or T.T	crying
:P or :p or xp or xP or XP or c(: :þ	tongue sticking out (silly)	D:< or]:< or >:[or):< or >:(	angry
:/ or :\ or :| or D:	indifferent; worried, amazed	:0	gasp, surprised, astonished
8D or BD or 8) or B-)	smiley with glasses	-_- or -.-	annoyed, really? not surprised, serious.
;) or ;] or (; or [;	winking smile	:D or xD or XD	huge grin, very happy
¬_¬ or -.- or -.-' or -_- or -_-'	Annoyed, Sweat-drop, not surprised, Facepalm	ʊ_ʊ	Expression of disapproval or disbelief

Styles of the computer underground text art scene

Atari 400/800 ATASCII

The Atari 400/800 did not follow the ASCII standard and had its own character set, called ATASCII. The emergence of ATASCII art coincided with the growing popularity of BBS Systems caused by availability of the acoustic couplers that were compatible with the 8-bit home computers. ATASCII text animations are also referred to as "break animations" by the Atari sceners.

"Block ASCII" / "High ASCII" style ASCII art on the IBM PC

Block ASCII display via Notepad versus ACiDView for Windows

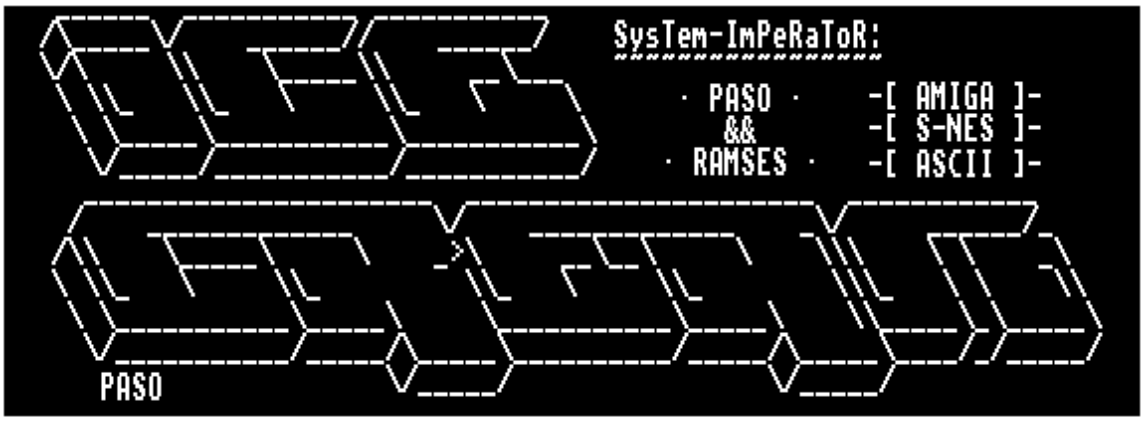
So-called "block ASCII" or "high ASCII" uses the extended characters of the 8-bit code page 437, which is a proprietary standard introduced by IBM in 1979 (ANSI Standard x3.16) for the IBM PC and MS DOS operating system. "Block ASCIIs" were widely used on the PC during the 1990s until the Internet replaced BBSes as the main communication platform. Until then, "block ASCIIs" dominated the PC Text Art Scene.

The first art scene group that focused on the extended character set of the PC in their art work was called "Aces of ANSI Art," or "AAA." Some members of left in 1990, and formed a group called ACiD, "ANSI Creators in Demand." In that same year the second major underground art scene group was founded, ICE, "Insane Creators Enterprise".

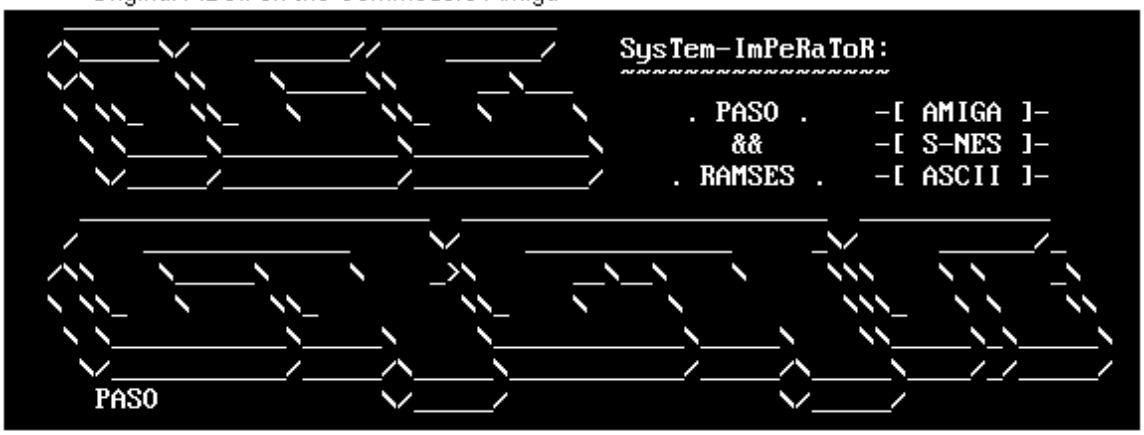
There is some debate between ASCII and block ASCII artist, with "Hardcore" ASCII artists maintaining that block ASCII art is in fact ANSI art, because it does not use the 128 characters of the original ASCII standard. On the other hand, block ASCII artists argue that if their art uses only characters of the computers character set, then it is to be called ASCII, regardless if the character set is proprietary or not.

Microsoft Windows does not support the ANSI Standard x3.16. One can view block ASCIIs with a text editor using the font "Terminal", but it will not look exactly as it was intended by the artist. With a special ASCII/ANSI viewer, such as ACiDView for Windows, one can see block ASCII and ANSI files properly.

"Amiga"/"Oldskool" style ASCII art



Original ASCII on the Commodore Amiga



Same ASCII on the IBM PC

Oldschool/Amiga ASCII look on Commodore Amiga Computer versus look on the IBM PC (notice the tight spacing)

In the art scene one popular ASCII style that used the 7-bit standard ASCII character set was the so called "Oldskool" Style. It is also called "Amiga style", due to its origin and widespread use on the Commodore Amiga Computers. The style uses primarily the characters: `_/\-+=.()<>:`. The "oldskool" art looks more like the outlined drawings of shapes than real pictures. This is an example of "Amiga style" (also referred to as "old school" or "oldskool" style) scene ASCII art.

The Amiga ASCII Scene surfaced in 1992, 7 years after the introduction of the Commodore Amiga 1000. The Commodore 64 PETSCII scene did not make the transition to the Commodore Amiga as the C64 demo and warez scenes did. Among the first Amiga ASCII art groups were ART, Epsilon Design, Upper Class, Unreal. This means that the text art scene on the Amiga was actually younger than the text art scene on the PC. The Amiga artists also did not call their ASCII art style "Oldskool". That term was introduced on the PC. When and by whom is unknown and lost in history.

The Amiga style ASCII artwork was most often released in the form of a single text file, which included all the artwork (usually requested), with some design parts in between, as opposed to the PC art scene where the art work was released as a ZIP archive with separate text files for each piece. Furthermore, the releases were usually called "ASCII collections" and not "art packs" like on the IBM PC.

In text editors

```
 _____ ___ ____ _       _
|  ___|_ _/ ___| | ___| |_
| |_   | | |   _| |/ _ \ __|
|  _|  | | |_| | |  __/ |_
|_|   |___\____|_|\___|\__|
```

This kind of ASCII art is hand made in a text editor. Popular editors used to make this kind of ASCII art include CygnusEditor aka CED (Amiga) and EditPlus2 (PC).

Oldskool font example from the PC, which was taken from the ASCII Editor *FIGlet*.

Newskool style ASCII art

Newskool ASCII Screenshot

"Newskool" is a popular form of ASCII art which capitalizes on character strings like "$#Xxo". In spite of its name, the style is not "new"; on the contrary, it was very old but

fell out of favor and was replaced by "Oldskool" and "Block" style ASCII art. It was dubbed "Newskool" upon its comeback and renewed popularity at the end of the 1990s.

Newskool changed significantly as the result of the introduction of extended proprietary characters. The classic 7-bit standard ASCII characters remain predominant, but the extended characters are often used for "fine tuning" and "tweaking". The style developed further after the introduction and adaptation of Unicode.

Methods for generating ASCII art

While some prefer to use a simple text editor to produce ASCII art, specialized programs have been developed that often simulate the features and tools in bitmap image editors. For Block ASCII art and ANSI art the artist almost always uses a special text editor, because the required characters are not available on a standard keyboard.

The special text editors have sets of special characters assigned to existing keys on the keyboard. Popular MS DOS based editors, such as TheDraw and ACiDDraw had multiple sets of different special characters mapped to the F-Keys to make the use of those characters easier for the artist who can switch between individual sets of characters via basic keyboard shortcuts. PabloDraw is one of the very few special ASCII/ANSI art editors that were developed for MS Windows XP.

Non fixed-width ASCII

Most ASCII art is created using a monospace font, where all characters are identical in width (Courier New is a popular monospace font). Early computers in use when ASCII art came into vogue had monospace fonts for screen and printer displays. Today most of the more commonly used fonts in word processors, web browsers and other programs are proportional fonts, such as Arial or Times New Roman, where different widths are used for different characters. ASCII art drawn for a fixed width font will usually appear distorted, or even unrecognizable when displayed in a proportional font.

Some ASCII artists have produced art for display in proportional fonts. These ASCIIs, rather than using a purely shade-based correspondence, use characters for slopes and borders and use block shading. These ASCIIs generally offer greater precision and attention to detail than fixed-width ASCIIs for a lower character count, although they are not as universally accessible since they are usually relatively font-specific.

Animated ASCII art

Animated ASCII art started in 1970 from so-called VT100 animations produced on vt100 terminals. These animations were simply text with cursor movement instructions, deleting and erasing the characters necessary to appear animated. Usually, they represented a long hand-crafted process undertaken by a single person to tell a story.

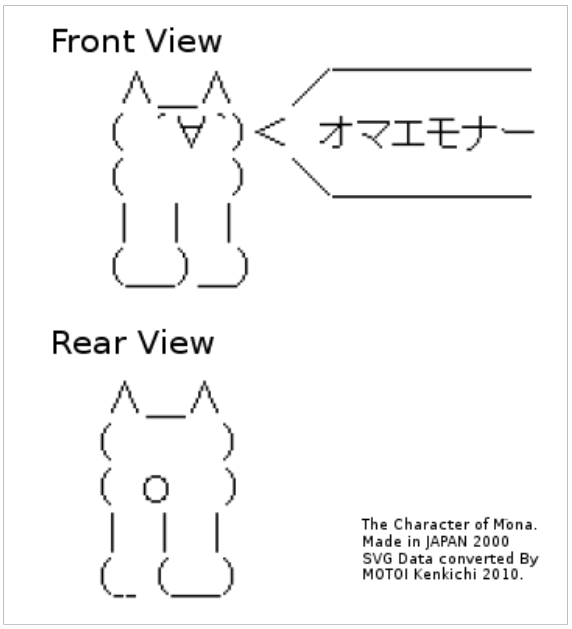
Contemporary web browser revitalized animated ASCII art again. It became possible to display animated ASCII art via JavaScript or Java applets. Static ASCII art pictures are loaded and displayed one after another, creating the animation, very similar to how movie projectors unreel film reel and project the individual pictures on the big screen at movie theaters. A new term was born: *ASCIImation* - another name of Animated ASCII Art. A seminal work in this arena is the Star Wars ASCIImation. More complicated routines in JavaScript generate more elaborate ASCIImations showing effects like Morphing effects, star field emulations, fading effects and calculated images, such as mandelbrot fractal animations.

There are now many tools and programs that can transform raster images into text symbols; some of these tools can operate on streaming video. For example, the music video for pop singer Beck Hansen's song "Black Tambourine" is made up entirely of ASCII characters that approximate the original footage.

Other text-based art

There are a variety of other types of art using text symbols from character sets other than ASCII and/or some form of color coding. Despite not being pure ASCII, these are still often referred to as "ASCII art". The character set portion designed specifically for drawing is known as the line drawing characters or pseudo-graphics.

Shift_JIS



Monā (モナー *Monā*?) Posted on **2ch** (２ちゃんねる *Nichanneru*?) in 2000

A large character selection and the availability of fixed-width characters allow Japanese users to use Shift JIS as a text-based art on Japanese websites.

Overprinting (surprint)

In the 1970s and early 1980s it was popular to produce a kind of text art that relied on overprinting — the overall darkness of a particular character space dependent on how many characters, as well as the choice of character, printed in a particular place. Thanks to the increased granularity of tone, photographs were often converted to this type of printout. Even manual typewriters or daisy wheel printers could be used. The technique has fallen from popularity since all cheap printers can easily print photographs, and a normal text file (or an e-mail message or Usenet posting) cannot represent overprinted text. However, something similar has emerged to replace it: shaded or colored ASCII art, using ANSI video terminal markup or color codes (such as those found in HTML, IRC, and many internet message boards) to add a bit more tone variation. In this way, it is possible to create ASCII art where the characters only differ in color

Chapter 8

ANSI Art and Tiled Printing

ANSI art

ANSI art is a computer artform that was widely used at one time on BBSes. It is similar to ASCII art, but constructed from a larger set of 256 letters, numbers, and symbols — all codes found in IBM code page 437, often referred to as extended ASCII and used in MS-DOS and Unix environments. ANSI art also contains special ANSI escape sequences that color text with the 16 foreground and 8 background colours offered by ANSI.SYS, an MS-DOS device driver loosely based upon the ANSI X3.64 standard for text terminals. Some ANSI artists take advantage of the cursor control sequences within ANSI X3.64 in order to create animations, commonly referred to as *ANSImations*. ANSI art and text files which incorporate ANSI codes carry the *de facto* `.ANS` file extension.

ANSI art is considerably more flexible than ASCII art, because the particular character set it uses contains symbols intended for drawing, such as a wide variety of box-drawing characters and block characters that dither the foreground and background color. It also adds accented characters and math symbols that often find creative use among ANSI artists.

The popularity of ANSI art encouraged the creation of a powerful shareware package called TheDraw coded by Ian E. Davis in 1986. Not only did it considerably simplify the process of making an ANSI art screen from scratch, but it also included a variety of "fonts", large letters constructed from box and block characters, and transition animations such as *dissolve* and *clock*. No new versions of TheDraw emerged after version 4.63 in 1993, but in later years a number of other ANSI editors appeared, some of which are still maintained today.

The decline of both BBSes and DOS users has made it difficult for many users to even view ANSI animations. As a consequence, this form of art is no longer practiced to the degree it once was.

Entry ANSI art screen for ZZT

The popular game creation system (GCS) *ZZT* used ANSI graphics exclusively. A later GCS based on the same concept, *MegaZeux*, allowed users to modify the extended ASCII character set as well.

Trade Wars 2002, a multiplayer BBS game that remains popular 20 years after its release in 1986, used ANSI graphics to depict ships, planets, and important locations, and included cutscenes and even a cinema with ANSI animations. Many of these ANSI graphics were created by Drew Markham, who went on to form Xatrix/Gray Matter Interactive and develop *Redneck Rampage* and *Return to Castle Wolfenstein*, among other titles.

Dwarf Fortress is another game, first released in 2002, leveraging ANSI graphics. Dwarf Fortress' default graphics is exclusively ANSI code page 437 characters.

In 1987 Amiga program *Skyline BBS* was the first BBS management program featuring an extension to ANSI Art as a true form of script markup language communication protocol. It was called Skypix and was capable to give the user a complete graphical interface, featuring rich graphic content, changeable fonts, mouse-controlled actions, animations and sound.

Tiled printing

A tiled printing of a subway station hung in a hallway (source photo)

Tiled printing, sometimes known as **rasterbating**, is a feature of many computer programs that enables them to print images larger than a standard page. The program overlays a grid on the printed image in which each cell (or tile) is the size of a printed page and then prints each tile. A person can then arrange the tiles to reconstruct the full image.

Tiled printing has been widespread since the days of mainframe computers. Programs were available to convert images to ASCII art that, when printed large enough and viewed sufficiently far away, appeared to be smoothly shaded. Modern software may use halftoning to achieve a similar effect.

Another form of tiled printing, inspired by continuous feed printers, involves making a long message of letters, possibly with inline graphics of the same height, and printing it sideways over several pages to make a banner. This type of printing is usually associated with The Print Shop, a 1980s software package.

Inexpensive ink jet printers now allow people to make tiled printouts that do not sacrifice the original image's resolution. These decorations are sometimes called *rasterbations*, after a popular tiled printing program, "The Rasterbator."

World records

The Groton School Rasterbation

Rasterbation is often the subject of record-breaking attempts to create the largest and most impressive tiled prints. The title of the world's largest rasterbation was previously held by "the Doomtech crew". However, in June 2007 the graduating class of the University of Toronto Schools, a Toronto high school, produced a rasterbation incorporating 1462 sheets of 8.5"x14" legal-sized paper. This achievement surpassed the previous record by over two hundred sheets, but then was overtaken by a group from Groton School in May, 2008. The Groton School Rasterbation used more than 1500 sheets of 11x14" paper and was 100' tall, making it the largest ever. The Groton School rasterbation was made by a team of only six consisting of Ian MacLellan, Piers MacNughton, Lacarnly Creech, Hannah Jeton, Theodore Frelinghuysen and Chris Pitsiokos.

Chapter 9

ANSI Escape Code and TheDraw

ANSI escape code

ANSI escape sequences are characters embedded in the text used to control formatting, color, and other output options on video text terminals. Almost all terminal emulators designed to show text output from a remote computer, and (except for Windows) to show text output from local software, interpret at least some of the ANSI escape sequences.

History

Almost all manufacturers of video terminals added vendor-specific escape sequences to do operations such as placing the cursor at arbitrary positions on the screen. As these sequences were all different, elaborate libraries such as termcap were created so programs could use the same API for all of them. Another problem was that most designs required sending numbers (such as row & column) as the binary values of the characters, for some programming languages and for systems that did not use ASCII internally this was often difficult or impossible.

The first standard for **ANSI escape sequences** was **ECMA-48**, adopted in 1976. It was a continuation of a series of character coding standards, the first one being ECMA-6 from 1961, a 6 bit standard from which ASCII originates. ECMA-48 has been updated several times and the current edition is the 5th from 1991. It is also adopted by ISO and IEC as standard **ISO/IEC 6429**. The name ANSI escape sequence dates from the years 1981 to 1997, but in 1981 **ANSI** adopted ECMA-48 as the standard **ANSI X3.64** (and later, in 1997, withdrew it).

The first popular video terminal to support these sequences was the Zenith Z19 in 1977. Far more influential was the Digital VT100 introduced in 1978. The popularity of these gradually led to more and more software, especially bulletin board systems, into assuming the escape sequences worked, leading to almost all new terminals and emulator programs supporting them.

Support

Most terminal emulators running on Unix-like systems (such as xterm and the OS X Terminal) interpret ANSI escape sequences. The native text console in Linux (the text seen when X is not running) also interprets them. Terminal programs for Microsoft Windows designed to show text from an outside source (a serial port, modem, or socket) also interpret them. Some support for text from local programs on Windows is offered through alternate command processors such as JP Software's 4NT, Michael J. Mefford's ANSI.COM, and Jason Hood's ansicon.

Many Unix applications (e.g. ls, grep, vim, and emacs) can generate them, and until 2000 or so this was the primary way they produced their display, even if that display was almost always inside a terminal emulator on a graphics display. Utility programs such as tput output them, as well as in low-level programming libraries, such as termcap or terminfo, or a higher-level library such as curses.

Windows and DOS

MSDOS 1.0 did not support the ANSI or any other escape sequences. Only a few control characters (CR, LF, BS) were interpreted, making it impossible to do any kind of full-screen application. Any display effects had to be done with BIOS calls (or far more often by directly manipulating the IBM PC hardware).

MSDOS 2.0 introduced the ability to add a device driver for the ANSI escape sequences – the *de facto* standard being `ANSI.SYS`, but others are used as well. This continued to work through Windows 98. Extreme slowness and the fact that it was not installed by default made usage by software almost non-existent, software continued to directly manipulate the hardware to get the text display needed.

Console windows in Windows versions based on NT (Windows NT 4.0, Windows 2000, Windows XP, Windows Server 2003, Windows Vista and Windows Server 2008) do not support ANSI Escape sequences at all (though ANSI.SYS continued to work for 16-bit legacy programs executing under the NTVDM). Software can manipulate the console with ioctl-like Console API interlaced with the text output, for instance:

```
SetConsoleTextAttribute(ConOut,
background|FOREGROUND_RED|FOREGROUND_INTENSITY);
```

Sequence elements

Escape sequences start with the character **ESC** (ASCII decimal 27/hex 0x1B/octal 033). For two character sequences the second character is in the range ASCII 64 to 95 (**@** to **_**). Most of the sequences are however more than two characters, and start with the characters **ESC** and **[** (left bracket). This sequence is called CSI for **Control Sequence Introducer** (or Control Sequence Initiator). The final character of these sequences is in the range ASCII 64 to 126 (@ to ~).

There is a single-character **CSI** (155/0x9B/0233) as well. The **ESC**+**[** two-character sequence is more often used than the single-character alternative. Only the two-character sequence is recognized by devices that support just ASCII (7-bit bytes) or devices that support 8-bit bytes but use the 0x80–0x9F control character range for other purposes. On terminals that use UTF-8 encoding, both forms take 2 bytes, but the two-character sequence is more clear.

Though some encodings use multiple bytes per character, the following discussion is restricted to ASCII characters, and so assumes each character is directly represented by a single-byte.

Non-CSI Codes

Note: other C0 codes besides ESC - commonly BEL, BS, CR, LF, FF, TAB, VT, SO, and SI - may produce similar or identical effects to some control sequences when output.

ESC N = SS2, ESC O = SS3 - select a single character from one of the alternate character sets.

ESC ^ = PM, ESC _ = APC - these each take a single string of text, terminated by ST (ESC \ ). They are ignored by xterm

ESC P = DCS: Device control string, ESC] = OSC: Operating system command - these are similar to CSI, but not limited to integer arguments. Because they are frequently used, in many cases BEL is an acceptable alternative to ST. E.g., in xterm, the window title can be set by: "OSC0;this is the window titleBEL"

Note: pressing special keys on the keyboard, as well as outputting many xterm CSI, DCS, or OSC sequences, often produces a CSI, DCS, or OSC sequence.

CSI Codes

The general structure of most ANSI escape sequences is `CSI [`*`private mode characters(s?)`*`] n1 ; n2... [`*`trailing intermediate character(s?)`*`] letter`. The final byte, modified by private mode characters and trailing intermediate characters, specifies the command. The numbers are optional parameters. The default value used for omitted parameters varies with the command, but is usually 1 or 0. If trailing parameters are omitted, the trailing semicolons may also be omitted.

The final byte is technically any character in the range 64 to 126 (hex 0x40 to 0x7e, ASCII @ to ~), and may be modified extended with leading intermediate bytes in the range 32 to 47 (hex 0x20 to 0x2f).

The colon (0x3a) is the only character not a part of the general sequence; it was left for future standardization, so any sequence containing it should be ignored. Although

multiple private mode characters or trailing intermediates are permitted, there are no such known usages.

If there are any leading private mode characters, the main body of the sequence could theoretically contain any order of characters 0x30 - 0x3f instead of a well-formed semicolon-separated list of numbers, but all known terminals are nice and just use them as a flag. Sequences are also private if the final byte is in the range 112 to 126 (hex 0x70 to 0x7e, ASCII *p* to ~).

Examples of private escape codes include the **DECTCEM** (DEC text cursor enable mode) shown below. It was first introduced for the VT-300 series of video terminals.

The existence of a C0 control, DEL (0x7f), or a high characters is undefined. Typically, implementations will either cancel the sequence or execute the control and then continue parsing the CSI sequence.

Some ANSI escape sequences (not a complete list)

Code	Name	Effect
CSI *n* A	CUU – CUrsor Up	Moves the cursor *n* (default 1) cells in the given direction. If the cursor is already at the edge of the screen, this has no effect.
CSI *n* B	CUD – CUrsor Down	
CSI *n* C	CUF – CUrsor Forward	
CSI *n* D	CUB – CUrsor Back	
CSI *n* E	CNL – Cursor Next Line	Moves cursor to beginning of the line *n* (default 1) lines down.
CSI *n* F	CPL – Cursor Previous Line	Moves cursor to beginning of the line *n* (default 1) lines up.
CSI *n* G	CHA – Cursor Horizontal Absolute	Moves the cursor to column *n*.
CSI *n* ; *m* H	CUP – CUrsor Position	Moves the cursor to row *n*, column *m*. The values are 1-based, and default to 1 (top left corner) if omitted. A sequence such as CSI ;5H is a synonym for CSI 1;5H as well as CSI 17;H is the same as CSI 17H and CSI 17;1H
CSI *n* J	ED – Erase Data	Clears part of the screen. If *n* is zero (or missing), clear from cursor to end of screen. If *n* is one, clear from cursor to beginning of the screen. If *n* is two, clear entire screen (and moves cursor to upper left on MS-DOS ANSI.SYS).
CSI *n* K	EL – Erase in Line	Erases part of the line. If *n* is zero (or missing), clear from cursor to the end of the line. If *n* is one, clear from cursor to beginning of the line. If *n* is two, clear entire line. Cursor

		position does not change.
CSI *n* S	SU – Scroll Up	Scroll whole page up by *n* (default 1) lines. New lines are added at the bottom. (not ANSI.SYS)
CSI *n* T	SD – Scroll Down	Scroll whole page down by *n* (default 1) lines. New lines are added at the top. (not ANSI.SYS)
CSI *n* ; *m* f	HVP – Horizontal and Vertical Position	Moves the cursor to row *n*, column *m*. Both default to 1 if omitted. Same as CUP
CSI *n* [;*k*] m	SGR – Select Graphic Rendition	Sets SGR parameters. After CSI can be zero or more parameters separated with ;. With no parameters, CSI m is treated as CSI 0 m (reset / normal), which is typical of most of the ANSI escape sequences.
CSI 6 n	DSR – Device Status Report	Reports the cursor position to the application as (as though typed at the keyboard) ESC[*n*;*m*R, where *n* is the row and *m* is the column. (May not work on MS-DOS.)
CSI s	SCP – Save Cursor Position	Saves the cursor position.
CSI u	RCP – Restore Cursor Position	Restores the cursor position.
CSI ?25l	DECTCEM	Hides the cursor.
CSI ?25h	DECTCEM	Shows the cursor.

SGR (Select Graphic Rendition) parameters

Code	Effect	Note
0	Reset / Normal	all attributes off
1	Bright (increased intensity) or Bold	
2	Faint (decreased intensity)	not widely supported
3	Italic: on	not widely supported. Sometimes treated as inverse.
4	Underline: Single	
5	Blink: Slow	less than 150 per minute
6	Blink: Rapid	MS-DOS ANSI.SYS; 150 per minute or more; not widely supported
7	Image: Negative	inverse or reverse; swap foreground and background
8	Conceal	not widely supported
9	Crossed-out	Characters legible, but marked for deletion. Not widely supported.
10	Primary(default) font	
11-19	*n*-th alternate font	Select the *n*-th alternate font. 14 being the fourth alternate font, up to 19 being the 9th alternate

		font.
20	Fraktur hardly ever supported	
21	Bright/Bold: off or Underline: Double	bold off not widely supported, double underline hardly ever
22	Normal color or intensity	neither bright, bold nor faint
23	Not italic, not Fraktur	
24	Underline: None	not singly or doubly underlined
25	Blink: off	
26	Reserved	
27	Image: Positive	
28	Reveal	conceal off
29	Not crossed out	
30–37	Set text color	30 + x, where x is from the color table below
38	Set xterm-256 text color	next arguments are 5;x where x is color index (0..255)
39	Default text color	implementation defined (according to standard)
40–47	Set background color	40 + x, where x is from the color table below
48	Set xterm-256 background color	next arguments are 5;x where x is color index (0..255)
49	Default background color	implementation defined (according to standard)
50	Reserved	
51	Framed	
52	Encircled	
53	Overlined	
54	Not framed or encircled	
55	Not overlined	
56-59	Reserved	
60	ideogram underline or right side line	hardly ever supported
61	ideogram double underline or double line on the right side	hardly ever supported
62	ideogram overline or left side line	hardly ever supported
63	ideogram double overline or double line on the left side	hardly ever supported
64	ideogram stress marking	hardly ever supported
90–99	Set foreground color, high intensity	aixterm (not in standard)
100–109	Set background color, high intensity	aixterm (not in standard)

Colors

Color table

Intensity	0	1	2	3	4	5	6	7
Normal	Black	Red	Green	Yellow	Blue	Magenta	Cyan	White
Bright	Black	Red	Green	Yellow	Blue	Magenta	Cyan	White

There are two other color standards CSS/HTML standard colors and X Window colors which standardize both the color names and associated RGB color values, but the escape sequence standard only specifies the color names, not RGB values. The chart below shows default RGB assignments for some common terminal programs, together with the CSS and the X-Window colors for these color names.

		Windows XP command prompt	Terminal.app	PuTTY	xterm	CSS/HTML	X Window
normal	**black**	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0	0, 0, 0
	red	128, 0, 0	194, 54, 33	187, 0, 0	205, 0, 0	255, 0, 0	255, 0, 0
	green	0, 128, 0	37, 188, 36	0, 187, 0	0, 205, 0	0, 255, 0	0, 128, 0
	yellow	128, 128, 0	173, 173, 39	187, 187, 0	205, 205, 0	255, 255, 0	255, 255, 0
	blue	0, 0, 128	73, 46, 225	0, 0, 187	0, 0, 238	0, 0, 255	0, 0, 255
	magenta	128, 0, 128	211, 56, 211	187, 0, 187	205, 0, 205	255, 0, 255	255, 0, 255
	cyan	0, 128, 128	51, 187, 200	0, 187, 187	0, 205, 205	0, 255, 255	0, 255, 255
	white	192, 192, 192	203, 204, 205	187, 187, 187	229, 229, 229	255, 255, 255	255, 255, 255
bright/light	**black**	128, 128, 128	129, 131, 131	85, 85, 85	127, 127, 127		
	red	255, 0, 0	252,57,31	255, 85, 85	255, 0, 0		
	green	0, 255, 0	37, 188, 36	49, 231, 34	0, 255, 0	144, 238, 144	144, 238, 144
	yellow	255, 255, 0	234, 236, 35	255, 255, 85	255, 255, 0	255, 255, 224	225, 255, 224
	blue	0, 0, 255	88, 51, 255	85, 85,	92, 92,	173, 216,	173, 216,

			255	255	230	230
magenta	255, 0, 255	249, 53, 248	255, 85, 255	255, 0, 255		
cyan	0, 255, 255	20, 240, 240	85, 255, 255	0, 255, 255	224, 255, 255	224, 225, 255
white	255, 255, 255	233, 235, 235	255, 255, 255	255, 255, 255		

In July 2004 the blue colors of xterm changed, (0,0,205) --> (0,0,238) for normal and (0,0,255) --> (92,92,255) for bright. As of 2010 old xterm versions still linger on many computers though.

Examples

`CSI 2 J` — This clears the screen and, on some devices, locates the cursor to the y,x position 1,1 (upper left corner).

`CSI 32 m` — This makes text green. On MS-DOS, normally the green would be dark, dull green, so you may wish to enable Bold with the sequence `CSI 1 m` which would make it bright green, or combined as `CSI 32 ; 1 m`. MS-DOS ANSI.SYS uses the Bold state to make the character Bright; also the Blink state can be set (via `INT 10, AX 1003h, BL 00h`) to render the Background in the Bright mode. MS-DOS ANSI.SYS does not support SGR codes 90–97 and 100–107 directly.

`CSI 0 ; 6 8 ; "DIR" ; 13 p` — This re-assigns the key F10 to send to the keyboard buffer the string "DIR" and ENTER, which in the DOS command line would display the contents of the current directory. (MS-DOS ANSI.SYS only) This is a private-use code (as indicated by the letter p), using a non-standard extension to include a string-valued parameter. Following the letter of the standard would consider the sequence to end at the letter D.

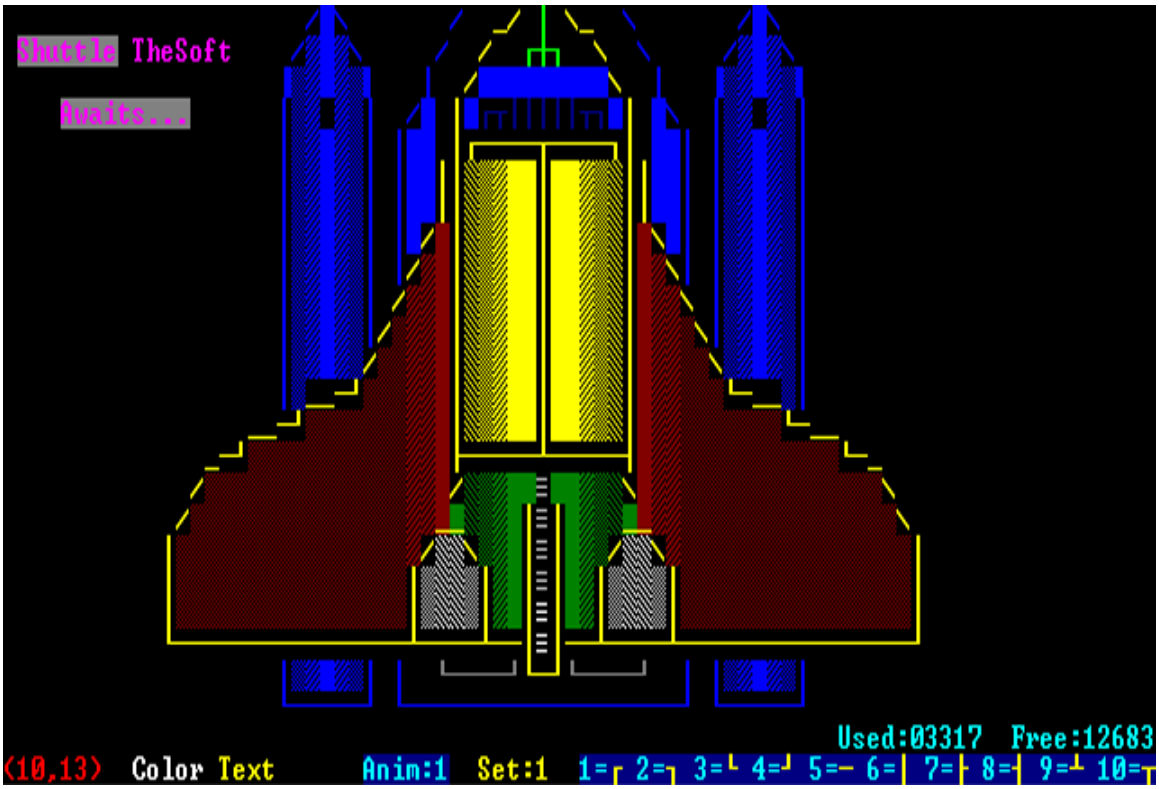
`CSI s` — This saves the cursor position. Using the sequence `CSI u` will restore it to the position. Say the current cursor position is 7(y) and 10(x). The sequence `CSI s` will save those two numbers. Now you can move to a different cursor position, such as 20(y) and 3(x), using the sequence `CSI 20 ; 3 H` or `CSI 20 ; 3 f`. Now if you use the sequence CSI u the cursor position will return to 7(y) and 10(x). Some terminals require the DEC sequences `ESC 7` / `ESC 8` instead which is more widely supported.

TheDraw

TheDraw ANSI Logo. Visible on the splash screen when the editor is loaded

TheDraw is a text editor for MS DOS to create ANSI and animations as well as ASCII art. The editor is especially useful to create or modify files in ANSI format and text documents, which use the graphical characters of the IBM ASCII code pages, because they are not supported by Microsoft Windows anymore. The first version of the editor was developed in 1986 by Ian E. Davis of TheSoft Programming Services. The last public version of the editor was version 4.63, which was released in October 1993.

TheDraw was one of the first ANSI editors that supported ANSIs longer than 25 rows. The limit in the latest available version is still 100 rows. Other editors, such as ACiDDraw are able to support ANSIs larger than 100 lines for a single ANSI/ASCII (ACiDDraw supports 1,000 lines). The animation mode is limited to 50 lines (rows). The column width can be extended from the standard 80 characters to 160, but this also reduces the row limit down to 50.



Sample ANSI by TheSoft loaded in TheDraw

Compatibility with Microsoft Windows

The program runs stably in a DOS Window on Windows XP and allows the user to maintain mouse control. Used with Windows Vista however, TheDraw performs with less predictable results. It works with DOSBox.

Significant features

Some of the features of the editor include:

- Mouse support to select blocks of text within the editor (even under Windows in window and full-screen mode)
- For the selected area/block exist a number of unique functions
    - Fill function to change the color of a whole section of the text
    - Copy/Move and Paste function to copy/move entire blocks of text within the document.
    - Erase function that clears the selected area of any characters without the surrounding characters changing position.
    - Replace function to replace the content of the selected area with the content of the TheDraw "clipboard".
    - Load/Save function to save only the selected area or load an ANSI/ASCII from the hard disk into the selected area (replace).
- Font manager to create/modify and organize ASCII and ANSI fonts to be used within the editor
- Additional file formats in addition to ANSI (.ANS) and text (.ASC).
    - The proprietary PCBoard (.PCB) and Wildcat! BBS (.BBS) file formats.
    - The AVATAR (.AVT) file standard defined by FidoNet.
    - Save files for various programming languages, including Assembler (.ASM), C (.H) and Turbo Pascal (.PAS).
    - Ability to save ANSIs as .COM binary, .BIN and object code (.OBJ).
    - Other supported formats: BSave (.BSV) and backup files (.BAK)
- Comprehensive help screens
- Preset transition animations to wipe or change the image
- ANSI animation support (creation and modification)
- The "Draw Mode" used automatically the appropriate character from the currently selected set to draw lines and corners by simply using the cursor keys (up/down/left/right)
- The default character sets can be modified and extended

Chapter 10

ASCII Stereogram and BSAVE (Graphics Image Format)

ASCII stereogram

ASCII stereograms are a form of ASCII art based on stereograms to produce the optical illusion of a three-dimensional image by crossing the eyes appropriately using a single image or a pair of images next to each other.

To obtain the 3D effect (in Figure 1 for instance), it is necessary for the viewer to diverge their eyes such that two adjacent letters in the same row come together. To help in focusing, try to make the two capital Os at the top look like three. Ensure that the image of the central dot is stable and in focus. Once this has been done, look down at the rest of the image and the 3D effect should become apparent. If the Os at the bottom of Figure 1 look like three, then the effect is reversed. It is also possible to obtain opposite 3D effects by crossing the eyes rather than diverging them.

Figure 1: Near and far distances.

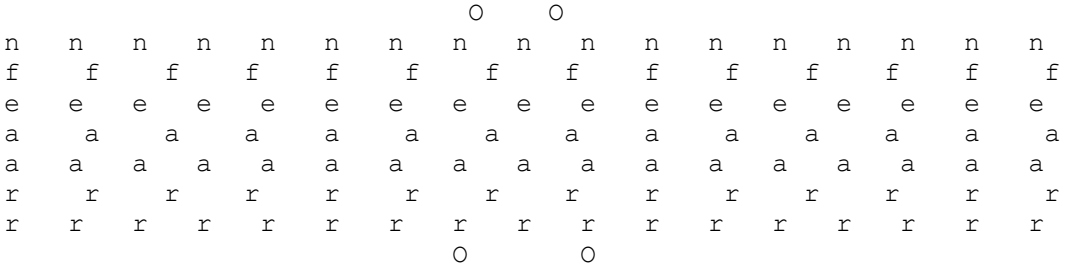


Figure 2 demonstrates the effect even more dramatically. Once the 3D image effect has been achieved, moving the viewer's head away from the screen increases the stereo effect even more. Moving horizontally and vertically a little also produces interesting effects.

Figure 3 shows a **Single Image Random Text Stereogram** (**SIRTS**) based on the same idea as a Single Image Random Dot Stereogram (SIRDS). The word "Hi" in relief can be seen when the image clicks into place.

Some people have included stereograms in their "signature" at the end of electronic mail messages and news articles. Figure 4 is such an example.

Figure 2: A floor, wall and ceiling.

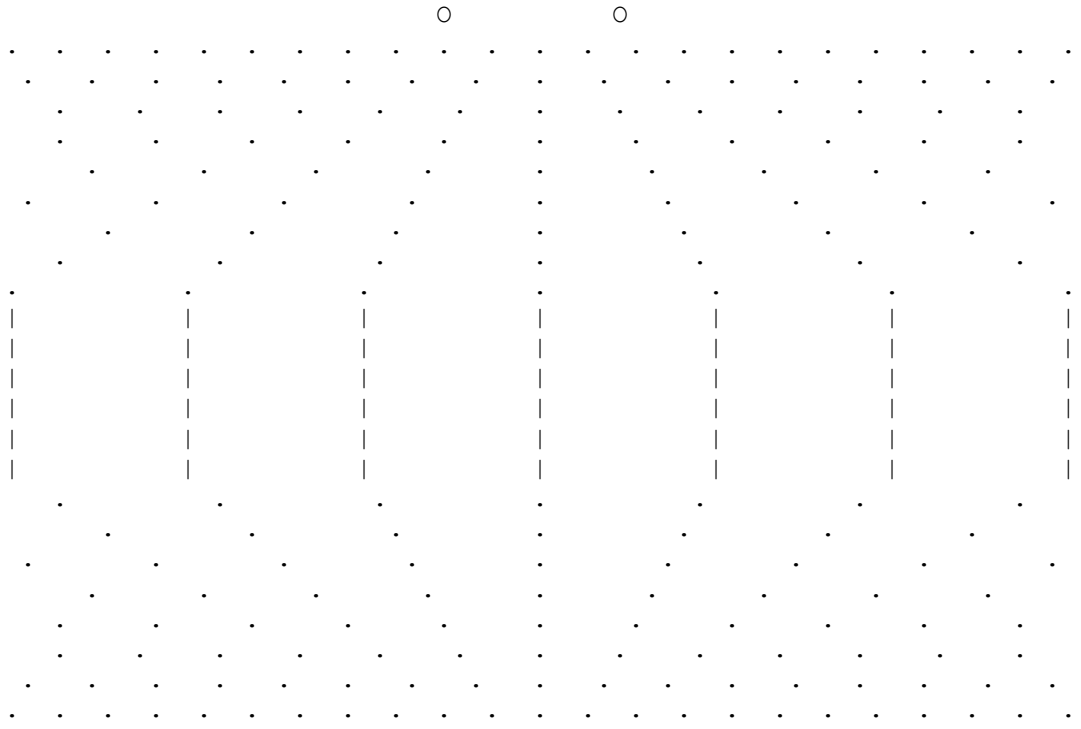


Figure 3: A single image random text stereogram.

```
                        O               O
OIWEQPOISDFBKJFOIWEQPOISDFBKJFOIWEQPOISDFBKJFOIWEQPOISDFBKJF
EDGHOUIEROUIYWEVDGHOXUIEROIYWEVDGHEOXUIEOIYWEVDGHEOXUIEOIYWE
KJBSVDBOIWERTBAKJBSVEDBOIWRTBAKJBSOVEDBOWRTBAKJBSOVEDBOWRTBA
SFDHNWECTBYUVRGSFDHNYWECTBUVRGSFDHCNYWECBUVRGSFDHCNYWECBUVRG
HNOWFHLSFDGWVRGHNOWFGHLSFDWVRGHNOWSFGHLSDWVRGHNLOWSFGLSDWVRG
YPOWVXTNWFECHRGYPOWVEXTNWFCHRGYPOWNVEXTNFCHRGYPWOWNVETNFCHRG
SVYUWXRGTWVETUISVYUWVXRGTWVETUISVYUWVXRGWVETUISVYUWVXRGWVETU
WVERBYOIAWEYUIVWVERBEYOIAWEYUIVWVERBEYOIWEYUIVWLVERBEOIWEYUI
EUIOETOUINWEBYOEUIOEWTOUINWEBYOEUIOEWTOUNWEBYOETUIOEWOUNWEBY
WFVEWVETN9PUW4TWFVEWPVETN9UW4TWFVETWPVET9UW4TWFBVETWPET9UW4T
NOUWQERFECHIBYWNOUWQXERFECIBYWNOUWFQXERFCIBYWNOFUWFQXRFCIBYW
VEHWETUQECRFVE[VEHWERTUQECFVE[VEHWQERTUQCFVE[VEOHWQERUQCFVE[
UIWTUIRTWUYWQCRUIWTUYIRTWUWQCRUIWTXUYIRTUWQCRUIBWTXUYRTUWQCR
IYPOWOXNPWTHIECIYPOWTOXNPWHIECIYPONWTOXNWHIECIYLPONWTXNWHIEC
R9UHWVETPUNRQYBR9UHWVETPUNRQYBR9UHWVETPUNRQYBR9UHWVETPUNRQYB
```

Figure 4: A stereogram signature.

```
 IIIIIIIIIIIIII   IIIIIIIIIIIIII
 H ( )  \|/    H   H   ( )   \|/ H
 H(   ) -O-    H   H  (   )  -O- H
 H     )/|\    H   H (     ) /|\ H
 H======^======H   H======^======H
 H- |----@-----H   H----| ---@---H
```

```
 H /|\ @\|/ @  H   H   /|\@ \|/@ H
 H    \|/  \|/ H   H     \|/  \|/H
 III^IIIIIII^III   III^IIIIIII^III
Wide eyed stereo  Wide  eyed  stereo
```

Moving animated versions of ASCII stereograms are possible too.

Text Emphasis

The stereo effect can be used to highlight individual words in a text, as a sort of "secret message". The effect can be disguised when the paragraph is block justified.

Figure 5: Emphasized single words.

```
According   to   the  According   to   the
police     inspector, police     inspector,
Edward John Billings, Edward John  Billings,
there are  too  many  there are  too  many
individuals too close individuals too close
to the case to make   to the case to make
an arrest.  I asked   an arrest.  I asked
Mary Smith to comment Mary Smith  to comment
on the case, but she  on the case,  but she
declined to comment,  declined to comment,
because she is soon   because she is  soon
to  be  married  to   to  be  married  to
Howard D. Fredericks, Howard D. Fredericks,
the victim's uncle.   the victim's uncle.
Charles Wilson, the   Charles Wilson, the
victim's     brother, victim's     brother,
stated that the chaos stated that the chaos
was  responsible for  was responsible  for
at least five suicide at least five suicide
attempts last week    attempts last week
alone.                alone.
```

Acknowledgements

Figures 1, 2, 3 and 4 are due to David B. Thomas, Jonathan Bowen, Charles Durst and Marty Hewes respectively. These four stereograms appeared on the publicly accessible *alt.3d* USENET newsgroup.

Originally adapted from an article on *ASCII Stereograms* by the author of that article (and with his permission).

BSAVE (graphics image format)

A **BSAVE Image** (aka "BSAVED Image") as it is referenced in a graphics program is an **image file format** created usually by saving raw video memory to disk (sometimes but not always in a BASIC program using the BSAVE command).

This format was in general use when the IBM PC was introduced. It was also in general use on the Apple II in the same time period. The Commodore 128 followed with the addition of the BSAVE and BLOAD Commands a short time later.

On the IBM, BSaved graphics and text images could be created for any video mode, with more complexity for the newer modes. On the Apple II and Commodore 128 BSaved Graphics were generally all that was used.

Typical file format

The BSAVED format is a device-dependent raster image format; the file header stores information about the display hardware address, and the size of the graphics data. The graphics data follows the header directly and is stored as raw data in the format of the native adapter's addressable memory.

There is no file compression, and therefore these load very quickly and without much programming when displayed in native mode.

No additional information such as (screen resolution, color depth and palette information, bit planes and so on) is stored. Video adapters were simple when this format was in wide use and the other information to load these could usually be inferred by programs that loaded these.

Origin

The BASIC programming language was shipped as part of the operating system on the first IBM PCs, Apple and Commodore 8-bit (like Commodore 64/128) computers. On computers that did not start up in BASIC, BASIC was loaded by running a program called an interpreter. The user could then type BASIC commands in "immediate mode" or by creating and/or running a numbered BASIC program within the interpreter.

One of the commands that early BASIC offered was BSAVE (Binary Save) and another (complementary) command was BLOAD (Binary Load). Using the BSAVE command, an addressable area and length of memory could be saved to disk as a named file (referred to as an "Image"). This "Image" of saved memory could then be reloaded from disk into addressable memory later with the BLOAD command. If the BSAVEd image contained **program code** it could be executed, if data it could be used again, and if the BSAVEd image contained graphics it could be viewed. The video area of memory was addressable.

The PUT and GET commands were used in addition to the BSAVE and BLOAD commands on the IBM PC to allow "clips" of the screen (or the entire screen) to be pre-formatted for BSAVE and BLOAD. These commands added image height and width to the BSAVE format, and were later carried over into the C programming language by some compiler vendors for the MS-DOS platform as the putimage() and getimage() run-time library functions. PUT and GET allowed display modifier verbs which resembled functions in the Windows Graphical Device Interface (GDI) used by programmers later.

Microsoft evolved and produced the BASIC interpreters that were bundled with the IBM PC, Apple II, and Commodore 128, and included the ability to BSAVE and BLOAD RAM memory images on all 3 platforms. While it is not clear whether or not Microsoft and its founder Bill Gates originated the BSAVE format they can be clearly credited with providing continued support for the BSAVEd Graphics Image throughout the history of its use by making it accessible to all users of these platforms and by completely documenting its use.

The BSAVE format of that time can be compared to some intents and purposes to the Windows BMP and DIB that came later.

Historical use

It was generally more common for computer users of the day (most who knew how to program in BASIC to some degree) to use the BLOAD command to load a graphics image that had been BSAVED than to load a BSAVED executable or data image.

Most early drawing programs allowed their graphics images to be BSAVED, making them easily available by an average user to be BLOADED back into video memory and viewed in a simple external program written in BASIC.

Recent use IBM PC

Common use of BSAVED images in graphics programs continued well into the 1990s but their use was primarily limited to users of programs like the Freeware program PicEM and the Shareware program VGACAD (which saved the MCGA display to a BSAVED file format called BLD BLoaDablepicture), and by beginning programmers and enthusiasts.

With the passing of MS-DOS so did the use of DOS programs that saved in the BSAVED format. Their use by programmers has stopped because the BLOAD command is no longer supported in modern programming languages which discourage accessing video memory directly.

Specifications IBM PC

CGA specification

CGA memory starts at memory segment B800h offset 0. Up to 16K of memory is available for video.

The amount of memory used for display depends on the video mode selected. A graphics screen requires 16000 bytes of memory and an 80 x 25 text screen requires 4000 bytes. A 40 x 25 text screen requires 2000 bytes.

The 3 common video modes are monochrome graphics, 4-color graphics or 80 column x 25 row color text.

A low-resolution 160 x 100 x 16 color graphics mode unofficially called **"XCGA"** and not supported in BIOS (interrupt 10h) also existed and had limited popularity amongst enthusiasts for a while. Technically this mode worked like color text mode.

40 column text mode was not generally used for BSAVED images.

Text

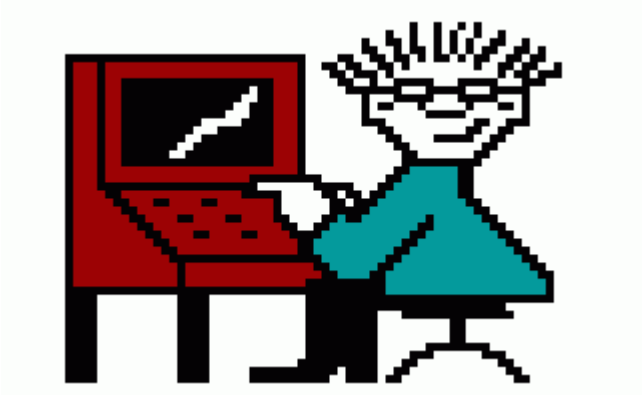


Fig. 2 Typical **BSAVED Text Screen** using Graphics Characters

80 column text is stored in 4 pages of 4000 bytes each. Generally only page 1 was used for BSAVED text images. Each character in an 80 column x 25 row text screen is stored in a contiguous byte array of character, attribute pairs of 2 bytes per text character. The attribute byte stores the foreground color in the low nibble and the background color and blink attribute in the high nibble. The colors 0-16 correspond to the colors offered by the basic 16 color EGA.

These could be cross-loaded to a low-end monochrome text video display with only 4K of memory with varying degrees of success since the color attributes on monochrome displayed differently.

The file extension that became popular for the BSaved text screen format was .BSV.

In the BBS days before the Internet was widely used, many Sysops used a program called TheDraw© from TheSoft Programming Services in Fremont, CA as the tool-of-choice to design and build user interface screens. Programmers used TheDraw© to design text-based user interfaces, or to edit screens that had been captured from other text programs, and then often would display these edited screens in their own programs, or as slideshows for proof-of-concept and so-forth. TheDraw© saved in various formats including BSAVED with the default file extension of .BSV.

Header

The following C language source code documents the header and tailer in a BSAVED text image:

```
/* a Microsoft compatible bsaved memory text image format descriptor */
unsigned char BSV_header={

    '\xfd',          /* ID Flag = file descriptor identifier bsaved
file */

    '\x00', '\xb8',  /* base address     = LSB | MSB original segment
*/
    '\x00', '\x00',  /* offset from base = LSB | MSB original offset
*/

    '\xA0', '\x0F'   /* data size = LSB | MSB of bytes to be loaded +
*/
    };

unsigned char BSV_image[4000];

unsigned char BSV_tailer={
    '\x1A'                      /* traditionally used by BASIC */
    };                          /* as a terminator */
```

Example

By contrast the following tiny GWBASIC program demonstrates how easy it was to load these in BASIC.

```
1  DEF SEG=&HB800
2  KEY OFF
3  CLS
4  FILES "*.BSV"
5  INPUT"Enter Name of Image To Load. <Blank Ends> : ",A$
10 IF A$="" THEN SYSTEM
30 BLOAD A$,0
40 A$=INPUT$(1)
50 GOTO 3
```

Graphics

The CGA Video Card (and other IBM PC video cards which support CGA which until recently were almost all) has two interleafs when in monochrome or 4-color Graphics mode. Visible memory is 16000 bytes. There is 192 bytes of unused data between the first and second interleaf, and 192 bytes of unused data following the second interleaf. The byte array is identical for either mode.

Typical variations

File Size 16392 bytes Header Size 7 bytes First Interleaf 8000 bytes Unused 192 Bytes Second Interleaf 8000 bytes Unused 192 Bytes Tailer Size 1 byte

Variations on the above include a smaller BSaved file format that does not load the unused memory after the second interleaf, nor the traditional trailer byte 0x1a (CPM EOF). This resulted in a slight saving of disk space.

File Size 16199 bytes Header Size 7 bytes First Interleaf 8000 bytes Unused 192 Bytes Second Interleaf 8000 bytes

A variation of the BSAVED CGA Image used by the earliest version of PCPaint stored a signature followed by a 2 byte palette index in the unused area directly following the first interleaf.

In the beginning BSAVED CGA Images were usually saved with either a .BAS or a .PIC extension. In the end, the .BAS extension was almost always used.

Pixel data

Monochrome data is stored as 1 bit per pixel with pixel(0,0) (x,y) being the high bit in the first byte and pixel(0,7) being the low bit in the first byte. The horizontal resolution for a monochrome CGA image is 640 pixels.

Color data is stored as 2 bits per pixel with pixel(0,0) (x,y) being the 2 highest bits in the first byte and pixel(0,3) being the 2 lowest bit in the first byte. The horizontal resolution for a monochrome CGA image is 640 pixels. The nominal color values are either BLACK, CYAN, MAGENTA, WHITE or BLACK, GREEN, RED, and ORANGE in color order depending on which of the two palettes was selected, with variations on border color (color 0). A third palette was available on some cards as well with nominal values of BLACK, CYAN, RED, WHITE. The horizontal resolution for a monochrome CGA image is 320 pixels.

The interleaf breaks on 80 byte boundaries which means that byte[80] is displayed starting at pixel(2,0), and conversely byte[8192] is displayed starting at pixel(1,0), and so it goes. Both color and monochrome have a vertical resolution of 200 scanlines.

Cross-loading

Fig. 3 4-Color **BSAVED Graphics Screen** displayed in Monochrome

Because the monochrome and color images could be cross-loaded, a color image displayed in monochrome would be naturally dithered because of the bit alteration provided by the bit pairs in colors 1 and 2. Color 0 was still BLACK and color 3 was still WHITE. This dithering was fairly good for printouts of color images on the black and white printers of the day.

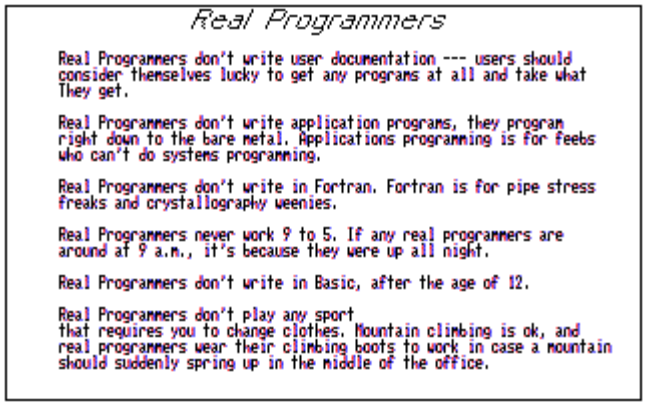


Fig. 4 Monochrome **BSAVED Graphics Screen** displayed in 4-Color CGA Mode

Monochrome images displayed in color didn't do so well. The aliasing of pixels that were not paired on bit boundaries created CYAN and MAGENTA anomalies which usually looked terrible.

Header

The following C language source code documents the header and tailer in a BSAVED graphics image:

```
/* a Microsoft compatible bsaved memory image format descriptor */
unsigned char PUT_header={

    '\xfd',          /* ID Flag = file descriptor identifier bsaved
file */

    /* BASIC will use original segment and offset information  */
    /* to reload a memory image unless "DEF SEG" has been used */
    /* and then an explicit offset is used as the second arg   */
    /* of the bload command.  If an offset is specified without*/
    /* first calling DEF SEG, The image will then be loaded to */
    /* the offset specified in the last segment pointed to     */
    /* by the last call to DEF SEG. DEF SEG without args returns */
    /* to DGROUP (the default data segment)                     */

    /* if loading a data array like an image fragment */
    /* we would first dimension the array as an integer array */
    /* which allocates memory just like malloc() and by the way */
    /* not like the Dim command works in VB.NET in 2007 */

    /* we would normally implement using an array in memory and */
    /* VARSEG and VARPTR to obtain the window for the array's   */
    /* memory location, i.e. override the defaults by windowing */
    /* to the array base using DEF SEG = VARSEG(arrayname(0)) then to
*/
    /* fill the array we would use BLOAD arrayname,
VARPTR(arrayname(0)) */
    /* using VARPTR to point to the offset from the memory base segment
*/

    /* remember that this is intel byte order - unsigned short in win32
*/
    '\x00', '\xb8',  /* base address     = LSB | MSB original segment
*/
    '\x00', '\x00',  /* offset from base = LSB | MSB original offset
*/

    '\x00', '\x40'   /* data size = LSB | MSB of bytes to be loaded +
*/
    };

unsigned char PUT_image[16384];

unsigned char PUT_tailer={
    '\x1A'                      /* traditionally used by BASIC */
    };                          /* as a terminator */
```

Example

The following GWBASIC program shows by how easy it was to load and save these in BASIC. It loads an existing image, draws a border around it and saves it.

```
10 INPUT"PICNAME : ", A$
20 INPUT"border color : ",B$
30 B%=VAL(B$)
40 SCREEN 1
50 DEF SEG=&HB800
55 BLOAD A$
60 LINE (0,0)-(319,199),B%,B
61 LINE (1,1)-(318,198),B%-1,B
62 LINE (2,2)-(317,197),B%,B
63 LINE (3,3)-(316,196),0,B
70 BSAVE A$,0, 16384
80 SYSTEM
```

XCGA low resolution graphics

XCGA mode as it was known as in the late 1980s enjoyed a brief notoriety amongst enthusiasts but never served any practical purpose because it was too coarse for even for the primitive graphics of the day.

Even so, computers equipped only with a CGA video adapter were still prevalent, with users who wanted to display graphics in more than the 4 color maximum that CGA graphics offered.

XCGA images were generally created by reducing images produced in higher resolution to a lower resolution facsimile by the use of file conversion software. This was generally done by enthusiasts as more of a novelty than for any real purpose.

BSAVED XCGA Images were usually saved with a .BAS or a .CGX extension.

Memory

The memory on XCGA was a contiguous byte array (just like MCGA mode) and was not interleaved like other CGA graphics modes. It started at memory segment B800 like all CGA video memory, and displaying the first 16000 bytes of the segment.

Header

The BSAVED image header was the same as the other CGA BSAVED headers, but the data length was 16000.

Pixel data

Pixels were in a raw format, arranged in pixel pairs called "big pixels" with 2 bytes per each 2 pixel-color pair stored in a contiguous byte array of 8000 big pixels.

Since XCGA was technically a color text mode, the first byte in a big pixel was actually the top two scanlines of a "text graphics" character in video ROM. As in color text mode the second byte stored the foreground color in the low nibble and the background color in the high nibble.

The colors 0-16 corresponded to the colors offered by the basic 16 color EGA.

By selecting the appropriate 2 pixel text block graphics character from the video ROM, even though the technical resolution was 80 x 100, the effective resolution was 160 x 100. Fig. 2 above uses the same graphics character normally used in an XCGA image to display a graphic in standard color text mode

Each scanline was 160 bytes long.

Example

The following QuickBasic source code shows how a low resolution BSAVED Image with 16 Colors was displayed. The commands are the same as displaying BSAVE images in standard CGA graphics or text modes with the addition of setting of registers on the CGA's MC6845 CRT controller.

```
CGXNAME$ = Command$:                    'get name from Command line
On Error GoTo NoFile

Open CGXNAME$ + ".CGX" For Input As 1:  'make sure it exists
Close

GoSub SetXCGA                           'trigger 160x100x16 mode
DEF SEG = &HB800                        'change DSEG to screen
BLOAD CGXNAME$ + ".CGX", 0              'dump picture to screen
a$ = INPUT$(1)
Screen 2: Screen 0: End                 'restore text mode and exit

NoFile: Beep:
        Print "Cannot find " + CGXNAME$
        End

SetXCGA:

'WARNING: Changing these registers settings may cause a CRASH !

DEF SEG = 0
POKE &H465, 0: OUT &H3D8, 0:
POKE &H466, 0: OUT &H3D9, 0
OUT &H3D4, 0: OUT &H3D5, 113
OUT &H3D4, 1: OUT &H3D5, 80
OUT &H3D4, 2: OUT &H3D5, 90
OUT &H3D4, 3: OUT &H3D5, 10
OUT &H3D4, 4: OUT &H3D5, 127
OUT &H3D4, 5: OUT &H3D5, 6
OUT &H3D4, 6: OUT &H3D5, 100
OUT &H3D4, 7: OUT &H3D5, 112
```

```
OUT &H3D4, 8: OUT &H3D5, 2
OUT &H3D4, 9: OUT &H3D5, 1
OUT &H3D4, 10: OUT &H3D5, 32
OUT &H3D4, 11: OUT &H3D5, 0
POKE &H465, 9: OUT &H3D8, 9
Return
```

EGA VGA specification

BSAVE images using multiple color planes and color registers were not common but they did exist.

Additional program code was needed to save and display these which was more complicated than the average user of the day could easily write. By the time the EGA came out fewer users bothered with programming in BASIC so the use of these was primarily restricted to **beginning programmers and enthusiasts** who were unwilling or unable to work in more capable programming languages than BASIC, or who were unwilling or unable to decipher the more advanced graphics formats that had started to emerge with file compression and color palettes.

MCGA specification

BSAVE images using 256 colors and color palettes were not common but they did exist. Additional program code was needed to save and display these which was more complicated than the average user of the day could easily write.

By the time the MCGA came out fewer users bothered with programming in BASIC so the use of these was primarily restricted to **beginning programmers and enthusiasts** who were unwilling or unable to work in more capable programming languages than BASIC, or who were unwilling or unable to decipher the more advanced graphics formats that had started to emerge with file compression and color palettes.

Two MS-DOS based utilities, PICEm and VGACad, offered saving of BSAVE MCGA images, with the latter also offering a BSAVE-style palette save. VGACad used the file extensions .BLD (BLoaDablePicture) and .PLT.

The problem with saving and loading an MCGA BSaved image was less difficult than saving and loading an EGA Bsaved image. The MCGA was organized in one contiguous byte array of 64000 bytes each with a value of 0-255, and each representing a pixel.

Loading and saving these was virtually the same as the CGA Bsaved format with the difference being that there were too many colors to infer the color palette, and the color palette was not easily saved unless the BASIC programmer understood some assembly language. If the programmer did understand assembly language, it usually wasn't very long before that understanding extended to more capable languages than BASIC and more advanced graphics formats than the BSAVE format.

Example

The following QuickBASIC program shows by how hard it was to load these in BASIC, especially by comparison to the small amount of programming that the GCA equivalent required.

```
'This source code is herewith released by me into the Public Domain.
'Bill Buckels, May 23, 2007
'You have a royalty-free right to use, modify, reproduce and distribute
this
'source code and the binaries that it produces in any way you find
useful

'MCGVU.BAS by Bill Buckels 1991
'an MCGA image viewer
'revised to use VGACad file extensions 2007

'CALL ABSOLUTE Statement Programming Examples
'demonstrate the use of ML routines to implement MCGA mode
'and setting of VGA Palette Registers in QuickBASIC 4.5

'Must be compiled using BC.EXE and ON ERROR option

IF COMMAND$="" THEN
   PRINT "MCGVU(C) Copyright 1991 by Bill Buckels
   PRINT "USAGE IS MCGVU <filename> <no extension>
   END
END IF

ON ERROR GOTO ENDER

DIM PCOL(384)
DIM R(256), G(256), B(256)

'parse the file name

PROOT$=""
TARGET$ = COMMAND$
TARGET = LEN(TARGET$)

FOR I = 1 TO TARGET
           A$=MID$(TARGET$,I,1)
           IF A$ = "." OR A$=" " THEN
                       I = TARGET
           ELSE
               PROOT$ = PROOT$ + A$
           END IF
NEXT I

' The MCGA Image and the Palette were saved outside of here
' Both are in BSAVED format
PCOLOR$=PROOT$ +".BLD"
PNAME$ =PROOT$ +".PLT"

'arrays to store machine language routines
```

```
DIM SETMCGA(5)
DIM SETDAC(10)
DIM CLEARDAC(14)

       ' Poke the machine-language program into the array.

       '---------------------- set MCGA Screen Mode ----------------
       RESTORE MCGABYTES
       READ NUMBYTES  'prepare assembler routine to set to MCGA mode
       DEF SEG=VARSEG(SETMCGA(0)) ' Window the segment.
       FOR J = 0 TO NUMBYTES-1
           READ VALUE
           POKE VARPTR(SETMCGA(0))+J,VALUE
       NEXT J
       CALL ABSOLUTE(VARPTR(SETMCGA(0)))
       DEF SEG   ' Restore the segment.

      '--------------------- set DAC REGISTERS --------------------
       RESTORE CLEARBYTES
       READ NUMBYTES  'prepare assembler routine to clear DAC regs
       DEF SEG=VARSEG(CLEARDAC(0)) ' Window the segment.
       FOR J = 0 TO NUMBYTES-1
           READ VALUE
           POKE VARPTR(CLEARDAC(0))+J,VALUE
       NEXT J
       DEF SEG   ' Restore the segment.

      '-------------------- load the ML routine ---------------
      RESTORE DACBYTES
      READ NUMBYTES  'load DAC registers
      DEF SEG=VARSEG(SETDAC(0)) ' Window the segment.
      FOR J = 0 TO NUMBYTES-1
             READ VALUE
             POKE VARPTR(SETDAC(0))+J,VALUE
      NEXT J

PALREAD:

    '--------------------- Read the palette File ----------------

    DEF SEG   'restore segment
    SEGMENT = VARSEG(PCOL(0))
    OFFSET  = VARPTR(PCOL(0))
    DEF SEG = SEGMENT
    BLOAD PCOLOR$,OFFSET

    '--------------------- load the color registers -------------

    FOR I = 0 TO 255
              R(I)  =PEEK(OFFSET)
              G(I)  =PEEK(OFFSET+1)
              B(I)  =PEEK(OFFSET+2)
              OFFSET=OFFSET+3
    NEXT I
   '-------------------- reset the DAC to 0 ---------------------
       DEF SEG=VARSEG(CLEARDAC(0)) ' Window the segment.
       CALL ABSOLUTE(VARPTR(CLEARDAC(0)))
```

```
       DEF SEG   ' Restore the segment.

   '------------------ load the image to the screen ------------

   DEF SEG= &HA000
   BLOAD PNAME$,0

   '----------------- initialize the registers ----------------

   FOR I = 0 TO 255
         IF NOT (R(I)+G(I)+B(I)) THEN        ' if the palette entry is
used
            DEF SEG=VARSEG(SETDAC(0))        ' Window the segment
            POKE VARPTR(SETDAC(0))+4,I       ' register number
            POKE VARPTR(SETDAC(0))+6, R(I)   ' red gun value
            POKE VARPTR(SETDAC(0))+8, G(I)   ' green gun value
            POKE VARPTR(SETDAC(0))+10,B(I)   ' blue gun value
            CALL ABSOLUTE(VARPTR(SETDAC(0))) ' do an open ML call
         END IF
   NEXT I

  ' wait for key press

  DEF SEG
  A$=INPUT$(1)

ENDER:
  SCREEN 2
  SCREEN 0
  CLEAR
  END

  '-------------------- Machine Language Routines --------------------

MCGABYTES:
       'Set Video Mode to MCGA mode 13H
       DATA 9
       DATA &H55, &HB4, &H00, &HB0, &H13, &HCD, &H10, &H5D, &HCB

DACBYTES:
      'SET DAC REGISTERS
      'The following assembly language  routine has 4- null characters
      'Which are replaced with Register Number, Red, Green, And Blue
      DATA 19
      DATA &H55, &H33, &HDB, &HB3, &H00
      DATA &HB6, &H00
      DATA &HB5, &H00
      DATA &HB1, &H00
      DATA &HB4, &H10, &HB0, &H10, &HCD, &H10, &H5D, &HCB

CLEARBYTES:
     'Clear the DAC registers
     DATA 27
     DATA &H55, &H33, &HDB, &H8B, &HF3, &HB8, &H10, &H10, &H33, &HC9
     DATA &H33, &HD2, &HCD, &H10, &H8B, &HDE, &H81, &HFB, &HFF, &H00
     DATA &H74, &H03, &H46, &HEB, &HEC, &H5D, &HCB
```

IBM PC BSAVE image fragments

PUT

Example

```
'This source code is herewith released by me into the Public Domain.
'Bill Buckels, May 26, 2007
'You have a royalty-free right to use, modify, reproduce and distribute
this
'source code and the binaries that it produces in any way you find
useful

'BPIC.BAS by Bill Buckels 1991
'PUT loader routine
'Written in QuickBASIC Version 4.5

DEFINT A-Z

'check for a color card
CLS
DEF SEG=&HB800
BYTE=PEEK(0)
IF NOT BYTE = 32 THEN
         PRINT"Sorry... CGA compatible adapter required
         END
END IF
'switch back to default segment
DEF SEG

'allocate memory for picture buffer
DIM PIC(8002)
SCREEN 1
COLOR 17,1

'Menu Routine

DO
 IF COMMAND$="" THEN
    CLS
    PRINT"Image Fragment Load
    FILES"*.PUT
    INPUT "Enter pic name :", PICNAME$
  ELSE
    PICNAME$=COMMAND$
  END IF

  IF NOT PICNAME$ = ""  THEN
           CLS
           GOSUB LOADPIC
           KEYPRESS$=INPUT$(1)
  END IF
  IF NOT COMMAND$ = "" THEN
                      PICNAME$=""
  END IF
LOOP UNTIL PICNAME$ =""
```

```
END
```

GET

Example

```
'This source code is herewith released by me into the Public Domain.
'Bill Buckels, May 30, 2007
'You have a royalty-free right to use, modify, reproduce and distribute
this
'source code and the binaries that it produces in any way you find
useful

' This is a corrected and rewritten example based on the QBASIC manual
' Demonstrating GET, BSAVE, PUT and BLOAD.

' Both the GWBASIC manual and the QBASIC manual presented their
' examples in a confusing manner, with too much or too little
information.
' The QBASIC manual further complicated the matter with incorrect
calculations.

' Here's a less-confusing example with properly calculated values.

' A BASIC Screen Fragment is stored in a byte array
' Preceded with a header giving the image size.
' Header - 4 bytes
' X - width of image in bits - short integer (2 bytes)
' Y - height of image in rasters - short integer (2 bytes)
' Image data - Variable
' Image data is arranged in rasters. Each raster is byte-aligned.

' In the example below, CGA 4-color screen mode is used.
' Rasters are stored at 2 bits per pixel (4 pixels per byte).
' Since the Cube is 101 pixels wide, 26 bytes are required per raster.
' The last 6 bits (3 pixels) of each scanline are unused (padding)
' Since the Cube is 101 rasters high, image data is 26 bytes x 101 =
2626 bytes.
' The Header requires 4 bytes.
' The array size required to store the image is 2630 bytes.
' Since a short integer is 2 bytes in size and since an integer array
is
' required to use the GET and PUT commands, an integer array size of
2630/2 =
' 1315 is required for this example.

' Allocate enough memory in an integer array to hold an image fragment
DIM fragment(1315)
fragmentSize = 2630
fragmentName$ = "MAGCUBE.PUT"
SCREEN 1

' Create an image fragment
GOSUB DRAWCUBE

' Transfer the image fragment into the integer array
```

```
GET (140, 25)-(240, 125), fragment

' BSAVE
' Point to the fragment array and BSAVE to disk.
DEF SEG = VARSEG(fragment(1))
BSAVE fragmentName$, VARPTR(fragment(1)), fragmentSize
DEF SEG                  ' Restore default BASIC segment.

LOCATE 1, 1: PRINT fragmentName$ + " Saved. Press a key to load."
KeyPress$ = INPUT$(1)
CLS

' BLOAD
' The image fragment's array size is BSaved
' so the BLoad command knows how to re-load it from disk
DEF SEG = VARSEG(fragment(1))
BLOAD fragmentName$, VARPTR(fragment(1))
DEF SEG                ' Restore default BASIC segment.

' Put the fragment on the screen.
' The fragment dimensions that the GET command stored are now in
' the array, so the PUT command knows the width and height to display
PUT (80, 10), fragment
LOCATE 1, 1: PRINT "Press a key to end..."
KeyPress$ = INPUT$(1)

SCREEN 2
SCREEN 0

END

DRAWCUBE:
' Draw a white box.
LINE (140, 25)-(140 + 100, 125), 3, B
' Draw the outline of a magenta cube inside the box.
DRAW "C2 BM140,50 M+50,-25 M+50,25 M-50,25"
DRAW "M-50,-25 M+0,50 M+50,25 M+50,-25 M+0,-50 BM190,75 M+0,50"
RETURN
```

Specifications Apple II

"Low-resolution" (lo-res) graphics

The Apple II had two "low resolution" (lo-res) graphics screen pages which allowed the simultaneous display of 16 colors at a very coarse resolution (40 x 48). This display mode was really too coarse for "meaningful" graphics display although Apple computer provided an excellent kaleidescope demo called "Rod's Color Pattern" to promote its use. Further, since this area of memory (also shared with the text display) was non-contiguous and the Apple II made use of the "memory holes" at the sides of the display for other purposes like mouse support, it was not practical to "BLoad" previously saved graphics images because they would "clobber" these "memory holes" that may be required for other processes.

"High-resolution" (hi-res) graphics

The Apple II had two "high-resolution" graphics screen pages: "page 1" (occupying the 8 kB of memory beginning at 0x2000) and "page 2" (immediately following at 0x4000). It also had two commonly used "high-resolution" graphics modes: "graphics only" ("pure" graphics mode) and "mixed text and graphics" (only offering a partial display of a graphics image with the bottom 4 lines of the respective text "page 1" or "page 2" at the bottom, leaving only the first 160 lines of the graphics screen visible).

The Apple II "high-resolution" (hi-res) graphics mode could be invoked from AppleSoft BASIC with the HGR and HGR2 commands. HGR enabled hi-res page 1 (occupying the 8 kB of memory beginning at 0x2000) with the bottom 4 lines of text page 1 at the bottom. HGR2 enabled hi-res page 2 (immediately following at 0x4000) with no text area showing. A POKE command could be used to independently enable or disable the text area on either page—though text page 2 was rarely used in BASIC, and BASIC programs normally loaded into its memory.

When ProDOS SYS programs were introduced (written in programming languages other than BASIC; like Assembler, C or Pascal) it was common to use "page 2" graphics only because SYS programs loaded at 0x2000 which conflicted with "page 1".

Resolution

The resolution of an Apple II hi-res graphics screen was 280 pixels x 192 rasters and allowed (on very early Apple II's) up to 4 colors to be displayed simultaneously. The hardware generated color by exploiting the nature of composite color video, essentially fooling the monitor into interpreting color from the pixel positions of an otherwise monochrome bitmap. This frugal design kept the hardware simple, but led to limitations as to what colors could be adjacent to each other on a line. Furthermore, as of the second version of the Apple II (non-plus) motherboard (also very early-on), horizontal groups of 7 pixels could be shifted by one half pixel, allowing 2 more colors (and a redundant white and black). The new colors and the original colors could not coexist within the same 7 pixel group.

Although not invented as a memory saving measure, the Apple II display design allowed a 6 color image to be displayed in only 8K of video memory despite the fact that colors would "bleed" (artifact) into each other if the pixels were incompatibly arranged.

File format

The 8 kB hi-res frame buffer (like all 8-bit Apple II video modes), due to another logic chip conserving idiosyncrasy, was interleaved, and in blocks of 128 bytes, with 8 byte non-displayed "holes" in between. Under Apple DOS all binary files had a header of 4 bytes (2 integers representing the address and length of the save). However, since one of the holes was at the very end of the buffer, it could safely be omitted from the save, keeping the entire package inside of 8 kB (with 4 bytes to spare), which avoided the file

taking another 256 byte block just for those 4 bytes of metadata. Under ProDOS the metadata is stored in the directory entry, so is not even an issue.

Various prefixes and suffices (such as "PIC", "HPIC", or "HGR") were used to indicate that the file was a graphics screen save. ProDOS had a special "fotofile" filetype (inherited from Apple SOS) of identical format. Unfortunately awareness of it was low, and few programs supported it.

Rasters (scanlines)

The rasters in an Apple II BSAVED image are not contiguous, nor are they in a simple sequential order. They match the video memory on the Apple II which was relatively complicated, and are arranged as follows:

Each scanline is 40 bytes long. There are 3 parent interleaves of 64 scanlines each. Starting at the beginning of the image file, scanlines are arranged in 128 byte blocks beginning with scanlines 0, 64, and 128 in the first block (see table below). 8 bytes at the end of each block are unused by the image.

Each parent interleaf has 8 child interleaves of 8 scanlines each, with a spacing of 1024 bytes (1K) between sequential scanlines in each of the 3 parent interleaves.

Starting at the beginning of the file and respective of the Apple II's interleaved display, the first 8 of the 128 byte blocks run as shown below. This also represents the first of 8 scanlines in each of the 24 child interleaves, and is effectively a block group. 8 of these block groups are stored in the file, with the scanlines in the table below being incremented by 1 for each subsequent block group sequence of 1024 bytes, totalling 8192 bytes (8K) of graphics image data.

First of 8 Sequential Scanline Block Groups in Apple II BSAVE Image 1024 Bytes (1K)					
	Block	**Leaf 1 - Offset 0 40 Bytes**	**Leaf 2 - Offset 40 40 Bytes**	**Leaf 3 - Offset 80 40 Bytes**	**Padding - Offset 120 8 Bytes**
Scanlines	1	0	64	128	Unused
Scanlines	2	8	72	136	Unused
Scanlines	3	16	80	144	Unused
Scanlines	4	24	88	152	Unused
Scanlines	5	32	96	160	Unused
Scanlines	6	40	104	168	Unused
Scanlines	7	48	112	176	Unused
Scanlines	8	56	120	184	Unused

Because of the interleaved display and the slow processor speed of the Apple II, when these loaded, a visible "venetian blind" effect was produced creating a "wipe" (fade effect) of the previous graphic if any.

Pixels (colors)

The colors in an Apple II BSAVED image are Black, Green, Violet, Orange, Blue, and White. Pixel color is represented by 2 bits. Although the nominal resolution is 280 x 192, the effective resolution considering colors is really only 140 x 192.

In a 40 byte scanline pixels are stored in an bit array of 280 pixel positions referred to as "columns". The highest bit of each byte is used as a "palette" bit and is not considered a column. The lowest bit represents the first column in the byte. So a scan line beginning with $03 would have the first two pixel positions set as white (with the next 5 pixels black).

If the palette bit in a byte is set, the 4 colors available for the pixels in that byte are black, blue, orange, and white.

If the palette bit is not set, the 4 colors available for the pixels in that byte are black, violet, green, and white

Ignoring the 40 palette bits in a 40 byte scanline, there are 280 columns (bits, pixel positions) but only 140 available pixels. So considering each pixel as being a column pair, with an even and odd column, from left to right starting at column 0, if even is set and odd is not set, the pixel is blue or violet. If even is not set and odd is set, the pixel is orange or green. Any two adjacent columns that are set will be white. This means that white pixels (and black pixels) can be positioned at more places than the other colors, but does not alter the fact that the effective resolution is really only 140 x 192 for an Apple II BSAVED image.

BLOAD example

To BLOAD a previously BSAVED image, use the Apple DOS/ProDOS command:

```
BLOAD <filename>
```

The following AppleSoft BASIC program loads a BSAVED Apple II Graphics Screen under Apple DOS 3.3. First it clears the screen, lists a directory, waits for user input (the BSAVED screen name) and then loads the screen and waits for a keypress, and repeats until the user ends the program.

```
1 REM This source code is herewith released
2 REM by me into the Public Domain.
3 REM Bill Buckels, May 31, 2007
10 TEXT
20 HOME
30 CALL - 1184: CALL 42350
```

```
40 PRINT "---------------------------------"
50 PRINT "ABINLOAD(C) APPLEII SCREEN LOADER"
60 PRINT "COPYLEFT BILL BUCKELS 2006"
70 PRINT "ENTER BIN FILE OR BLANK TO END..."
80 PRINT "---------------------------------"
90 INPUT "?"
100 IF BIN$ = "" THEN GOTO 500
110 HGR2
120 PRINT CHR$(4)"BLOAD "BIN$",A$4000"
130 GET A$
140 GOTO 10
500 TEXT
510 CALL - 1184: CALL 42350
520 END
```

BSAVE example

To BSAVE the HGR screen (starting at memory location $2000, for a length of $1FF8) use the Apple DOS/ProDOS command:

```
BSAVE <filename>,A$2000,L$1FF8
```

To BSAVE the HGR2 screen (starting at memory location $4000, for a length of $1FF8) use the Apple DOS/ProDOS command:

```
BSAVE <filename>,A$4000,L$1FF8
```

Specifications Commodore 64 and 128

The sample screens shown above were converted from the IBM CGA 320 x 200 x 4 Color BSAVED Image in Fig. 1. Due to the lower resolutions than the IBM-PC in both the Commodore's standard native graphics display modes, Fig. 6 sacrifices colors to black and white where needed during conversion resulting in color loss. The colors remain intact when converting to the lower resolution 4 color mode, but some detail is lost. Graphics layout on the C64 was quite restricted even when compared to the IBM-PC around the same time period but offered up to 16 colors which the IBM-PC did not and offered these in about half the RAM memory by comparison.

Graphics

Strictly speaking all the uncompressed Graphics File Formats on the Commodore 64 (C64) and 128 (C128) (and there were many) were in a variation of the BSAVE graphics image format because the Commodore stored all binary files (including Graphics) with a two-byte header which provided the file's originating load address. However, because most of these could not be BLoaded directly into the video areas of the C128 they really are a little too advanced to be considered as BSaved graphics images in the same context as on the IBM-PC and Apple II computers of the same vintage.

C64's did not provide a BSAVE nor a BLOAD command with their version of the Commodore BASIC language (Version 2). It wasn't until the release of Commodore BASIC 7 on the C128 that the BSAVE and BLOAD commands and the BSAVE graphics image format came into existence on the C128.

C64's BASIC did provide a LOAD command which was intended to LOAD binary program files, data files, and BASIC programs, but this command when used in a C64 BASIC program to load binary data files like BSAVE images needed to restart the BASIC program after the ("non-relocating") load.

This lack of support for loading binary data files from a BASIC program without restarting the program itself amounted to a very serious shortcoming in C64 BASIC Version 2 (later corrected by the C128's BLOAD command) which left the C64 user with the option writing a program that could be restarted after the ("non-relocating") load, or the option of slowly reading binary data files one byte at a time, and then moving the data to memory one byte at a time with data conversion further slowing-down the process.

Technically the standard HIRES graphics mode on the C128 was the same as on the C64 except that the default address in RAM where videoram (HIRES colors) are stored on the C128 directly preceded the default screen address (unlike the C64). This made loading a single BSAVE graphic file in either a monochrome or color format (of the uncompressed type produced by DOODLE) possible on the C128.

Resolution and pixel blocks

The two primary graphics mode resolutions that both the C64 and C128 shared were 320 x 200 (HIRES) and 160 x 200 (Multi-Color Mode). Both modes provided selection of colors from a standard built-in palette of colors.

In HIRES mode, the screen was broken-down in blocks of 8 pixels x 8 scanlines. Pixels of two colors from the 16 color palette could be displayed in each block.

In Multi-Color Mode, the screen was broken down in blocks of 4 pixels x 8 scanlines. Pixels of 4 colors from the 16 color palette could be displayed in each block.

However because only 2 colors could be stored in the standard memory area shared by HIRES mode, a separate area of RAM was used to store the other two colors which was not contiguous with the default screen address and which was fixed (even the C128). This meant that a separate BSAVE file containing the remaining 2 colors was required to BSAVE and BLOAD these on the C128 (keeping in mind that the C64 did not support the BSaved format).

This is somewhat similar to the loading of multiple BSaved files on the IBM PC's EGA, VGA, and MCGA Modes discussed above.

File format

The size of a HIRES Monochrome BSAVE image without the two byte header was 8000 bytes (the same size as visible screen memory), and was "linearized". The many file extensions possible included .HIR and .HBM, if a file extension was used at all.

The size of a HIRES Color BSAVE image (of the uncompressed type produced by DOODLE) was 9024 bytes (the same size as videoram plus visible screen memory) without the two byte header, and was "linearized". The file extension was .DD, if a file extension was used at all.

Other variations of HIRES and even Multi-Color BSaved formats could be produced from compatible graphics formats by splitting them into uncompressed pieces with BLoadable addresses. If a split graphics image was used it was generally split into an 8000 byte bitmap and 1000 byte videoram (2 colors) for HIRES mode, and with a third file, a 1000 byte colorram (2 colors), for Multi-Color Mode.

Like all BIN (Binary) files on the C64 and C128, the files had a binary header of 2 bytes (1 integer). The header gave the load address for the file, and the length was already known to BASIC 7. The default load addresses on the C128 were $2000, $1C00, and $D800 for bitmap, videoram, and colorram respectively.

Generally speaking, the popular uncompressed Multi-Color (4 Color) formats produced by the Paintbrush programs of the day could not be BLoaded without splitting them (in a separate program) and performing an additional operation of putting a background color which was stored elsewhere in the originating file into the 1000 high-nibbles of colorram.

Since the screen address on the C64 and C128 was relocatable, the screen address could be detected and used by the loader program when BLoading these split files.

Rasters (scanlines)

The rasters in a C128 BSAVED image are not contiguous, but are in a simple sequential order of pixel blocks (as noted above). They match the video memory on the C128 which in its standard modes was uncomplicated except for its pixel block orientation.

Because of the slow processor speed of the C128 combined with slow storage access, when a HIRES image loaded it was displayed in 25 passes of 40 chunks creating a choppy effect. If the HIRES image had colors it was often blurry and not recognizable since the colors loaded after the image was already being displayed leaving the user with nothing to do but wait until it was finished.

Pixels (colors)

The standard available colors in a C128 image are BLACK, WHITE, RED, CYAN, PURPLE, GREEN, BLUE, YELLOW, ORANGE, BROWN, LIGHT RED,DARK

GREY,MEDIUM GREY,LIGHT GREEN, LIGHT BLUE,and LIGHT GREY. Pixel color is represented by 2 bits in HIRES mode and 4 bits in Multi Color Mode. But as indicated above, not all 16 Colors can be displayed in a pixel block.

BLOAD examples C64

The BLOAD command did not exist on the C64. Instead a BASIC program could use a "non-relocating" LOAD command to load BSAVED Graphics if it was designed to restart after the load, as demonstrated in the following Commodore BASIC 2 program. It loads a colored BSAVED HIRES Graphics Screen on the C64 from 2 separate BSaved Files (a split image), waits for a keypress and ends.

The load address in the 2 files of $2000 for bitmap and $400 for videoram respectively are in the file headers and are used by the LOAD command to read the files directly into the C64's RAM memory.

Compared to BASIC 7 on the C128 (see C128 examples also below), the C64's BASIC 2 was somewhat less straight-forward.

```
1 GOTO 500
2 REM This source code is herewith released
3 REM by me into the Public Domain.
4 REM Bill Buckels, October 26, 2008
100 K$=""
110 GET K$
120 IF K$ = "" THEN 110
125 FOR I=8192 TO 16191: POKE I,0: NEXT I
130 POKE 53265, PEEK(53265) AND 223
135 SYS 49152
140 END
500 IF J=1 THEN 600
510 IF J=2 THEN 100
520 POKE 53272,PEEK(53272) OR 8
530 POKE 53265, PEEK(53265) OR 32
540 FOR I = 1024 TO 2047: POKE I,16 : NEXT I
550 B$="ERINLOG.BHI"
560 J=1
570 LOAD B$,8,1
600 V$="ERINLOG.VHI"
610 J=2
620 LOAD V$,8,1
```

A second (and slower) alternative shown below which does not require a program restart is to open the BSAVED Graphic as a file. The program example below is functionally the same as the one above.

```
1 REM This source code is herewith released
2 REM by me into the Public Domain.
3 REM Bill Buckels, October 2, 2007
5 POKE 53272,PEEK(53272) OR 8
10 POKE 53265, PEEK(53265) OR 32
```

```
20 FOR I = 1024 TO 2047: POKE I,16 : NEXT I
50 OPEN 1,8,2,"ERINLOG.BHI"
51 B$ = CHR$(0)
52 FOR I=1 TO 2:GET #1,A$:NEXT I
54 FOR I=8192 TO 16191
55 GET #1,A$:POKE I,ASC(A$+B$)
56 NEXT I
57 CLOSE 1
60 OPEN 1,8,2,"ERINLOG.VHI"
61 B$ = CHR$(0)
62 FOR I=1 TO 2:GET #1,A$:NEXT I
64 FOR I=1024 TO 2047
65 GET #1,A$:POKE I,ASC(A$+B$)
66 NEXT I
67 CLOSE 1
100 K$=""
110 GET K$
120 IF K$ = "" THEN 110
125 FOR I=8192 TO 16191: POKE I,0: NEXT I
130 POKE 53265, PEEK(53265) AND 223
135 SYS 49152
140 END
```

BLOAD examples C128

The following Commodore BASIC 7 program loads a BSAVED HIRES Monochrome Graphics Screen on the C128 then sets each of the colors for the 1000 pixel blocks to black and white, waits for a keypress and ends. No colors are stored in the image. The load address in the image is at the start of display memory at $2000.

```
1 REM This source code is herewith released
2 REM by me into the Public Domain.
3 REM Bill Buckels, August 19, 2007
10 GRAPHIC 1,1
20 BLOAD "ERINLOG.HIR"
24 REM POKE I,16 will reverse video
25 FOR I = 7168 TO 8167:POKE I,1:NEXT I
30 A$=""
40 GET A$
50 IF A$ = "" THEN 40
60 GRAPHIC 0,1
70 END
```

The following Commodore BASIC 7 program loads a BSAVED HIRES Colored Graphics Screen in DOODLE uncompressed format on the C128, waits for a keypress and ends. The load address in the image is at the start of C128 videoram at $1C00 directly before screen memory at $2000. No color setting is required since the colors are stored in the image and are loaded directly into the C128's HIRES color memory (videoram).

```
1 REM This source code is herewith released
2 REM by me into the Public Domain.
3 REM Bill Buckels, August 19, 2007
```

```
10 GRAPHIC 1,1
20 BLOAD "ERINLOG.DD"
30 A$=""
40 GET A$
50 IF A$ = "" THEN 40
60 GRAPHIC 0,1
70 END
```

The following Commodore BASIC 7 program loads a BSAVED Multi-Colored 4 Color Graphics Screen in 3 separate BSaved Files (a split image) on the C128, waits for a keypress and ends. The load address in the 3 files are $2000 for bitmap, $1C00 for videoram, and $D800 for colorram respectively. No color setting is required since the colors are stored in the videoram and colorram and are loaded directly into the C128's RAM memory.

```
10 GRAPHIC 3,1              :REM  MULTICOLOR GRAPHIC MODE
20 A=PEEK(1): B=PEEK(216)   :REM  SAVE THESE
30 POKE 216,255             :REM  TELL IRQ TO GIVE US VIC CONTROL
40 POKE 1, A AND 252        :REM  SELECT PROCESSOR NYBBLE BANK
50 BLOAD "ERINLOG.BMC"      :REM  LOAD BIT MAP AT $2000
60 BLOAD "ERINLOG.VMC"      :REM  LOAD COLORS 01 AND 10 AT $1C00
70 BLOAD "ERINLOG.CMC"      :REM  LOAD COLORS 11 AT $D800
80 POKE 1,A                 :REM  RESTORE SYSTEM NYBBLE BANK
90 POKE 216,B               :REM  RESTORE SYSTEM VIC CONTROL
100 K$ = ""
110 GET K$                  :REM  WAIT FOR A KEYPRESS
120 IF K$ = "" THEN 110
130 GRAPHIC 0,1             :REM  TEXT MODE
140 END
```

Bitmap converter example

The following C language source code is part of a Windows Program written in Microsoft C and the WIN32 SDK. It demonstrates how a monochrome bitmap in a standard format can be reorganized to display as a BSaved Image on the Commodore 128 (C128).

Programs that convert bitmaps with colors to a C128 HIRES or Multi-Color compatible format use the same basic logic as the example below, with additional logic to organize the color map. Under Windows on the IBM PC these converted images can then be inserted into C128 "disk images" used by C128 emulators and loaded by programs similar to the C128 Basic 7 programs shown above.

```
// This source code is herewith released by me into the Public Domain.
// Bill Buckels, August 25, 2007
int SaveFragmentToC64(char *name,char *printshopbuffer)
{

    // creates a banded C64 image from an 88 x 52 monochrome bitmap
    // ("old printshop" graphic format) which can be bloaded on the C64
    // and centered relative to the default screen address at $2000
    // the image is padded by 4 scanlines on the bottom to adjust
```

```
    // for C64 block increments of 8 scanlines.
    // space is padded to both sides of the image to provide a
contiguous load
    // and for consistency which results in an image of 320 x 56 in
actual size.
    int fh;

    unsigned char header;
    unsigned char buffer[320];

    unsigned x, y, y1, idx, offset, inset;

    // create in subdirectory if possible
    _mkdir("c1541");
    _chdir("c1541");

#ifndef S_IWRITE
#define S_IWRITE    0000200     // write permission, owner
#endif

if((fh = _open(name,O_CREAT|O_TRUNC|O_WRONLY|O_BINARY,S_IWRITE)) == -1)
{
            _chdir("..");
        return FAILURE;
        }

    // create load address to center on C64 screen
    header[0] = 0x4E;  // offset into screen = 72 x 40 = 2880 + 14 +
8192 ($2000)
    header = 0x2B;

     _write(fh,(char *)&header[0],2);

    // 14 + 11 + 15 = 40 byte scanline
    // bitmap picture height = 52 scanlines
    // C64 image height = 56 scanlines

    // 7 bands of 320 bytes instead of the usual 25
    // each band holds 8 scanlines from the original bitmap
    // except the last one which holds only 4
    for (y = 0; y < 7; y++) {
            // pad the band with blackspace
            for (idx = 0; idx < 320; idx++)buffer[idx] = 0;
            y1 = y * 8 * 11;
            inset = 8 * 14;
                for (x = 0; x < 11; x++) {
                     for (idx = 0; idx < 8; idx++) {
                        offset = y1 + (idx * 11) + x;  // down 8, across
11 times

                         if (y == 6 && idx > 3) {
                          // the last band only has 4 rasters
                          // skip reading the last 4
                          inset ++;
                          continue;
                         }
                        buffer[inset] = printshopbuffer[offset];
```

```
                        inset ++;

                    }
                }
                // band was created so write it out
                // clip the padding at the top and bottom of the C64
image
                // so it can be optionally loaded anywhere on C64 screen
                // including to top left or bottom right of C64 screen
                if (y == 0) _write(fh,(char *)&buffer,(320-14)); // top
                else if (y==6) _write(fh,(char *)&buffer[0],(320-15));
// bottom
                else _write(fh,(char *)&buffer[0],320);
        }

     _close(fh);
     _chdir("..");
     return SUCCESS;

}
```

Chapter 11

Cowsay and FIGlet

cowsay

cowsay	
Original author(s)	Tony Monroe
Stable release	3.03 / May 28, 2000
Written in	Perl
Operating system	Cross-platform
Available in	English
License	Artistic License / GNU General Public License

`cowsay` is a program which generates ASCII pictures of a cow with a message. It can also generate pictures using pre-made images of other animals. There is also a related program called **`cowthink`**, with cows with thought bubbles rather than speech bubbles.

```
 _________________________
< Typical cowsay output! >
 -------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

.cow files for Cowsay exist which are able to produce different variants of "cows", with different kinds of "eyes", and so forth. It is sometimes used on IRC, desktop screenshots, and in software documentation. It is more or less a joke within hacker culture, but has been around long enough that its use is rather widespread. In 2007 it was highlighted as a Debian package of the day.

Cowsay is written in the Perl programming language, and as such is easily adaptable to system tasks in Unix, such as telling users their home directories are full, they have new mail, et cetera. Additionally, it is quite adaptable to the Common Gateway Interface.

Example

The Unix command `fortune` can also be piped into the `cowsay` command:

```
baldur@baldur-desktop:~$ fortune | cowsay
 _______________________________________
/ You have Egyptian flu: you're going to \
\ be a mummy.                           /
 ---------------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

And using the parameter `-f` followed by `tux`, one can exchange the cow with Tux, the Linux mascot:

```
baldur@baldur-desktop:~$ fortune | cowsay -f tux
 _____________________________________________
/ You are only young once, but you can        \
\ stay immature indefinitely.                 /
 ---------------------------------------------
   \
    \
        .--.
       |o_o |
       |:_/ |
      //   \ \
     (|     | )
    /'\_   _/`\
    \___)=(___/
```

Parameters

Option	Purpose
-n	Disables word wrap, allowing the cow to speak FIGlet or to display other embedded ASCII art. Width in columns becomes that of the longest line, ignoring any value of **-w**
-W	Specifies width of the speech balloon in columns, i.e. characters in a monospace font. Default value is 40.
-b	“Borg mode”, uses == in place of `oo` for the cow′s eyes.
-d	“Dead”, uses `XX`.
-g	“Greedy”, uses `$$`.

-p	“Paranoid”, uses `@@`.
-s	“Stoned”, uses `**` to represent bloodshot eyes, plus a descending `U` to represent an extruded tongue.
-t	“Tired”, uses `--`.
-w	“Wired”, uses `OO`.
-y	“Youthful”, uses `. .` to represent smaller eyes.

-e eye_string	Manually specifies the cow′s eye-type, e.g. `cowsay -e ^^`.
-T tongue_string	Manually specifies the cow′s tongue shape, e.g. `cowsay -T \(\)` for a pair of parentheses.
-f cowfile	Specifies a .cow file from which to load alternative ASCII art. Accepts both absolute file-paths and those relative to the environment variable `COWPATH`.
-l	Lists the names of available cow-files in the `COWPATH` directory instead of displaying a quote.

FIGlet

FIGlet

Original author(s)	Glenn Chappell, Ian Chai
Initial release	1991 (as "newban") / 1993 (figlet 2.0)
Stable release	2.2.2 / July 2005
Written in	C
Operating system	Unix-like
Platform	Cross-platform
Type	Typesetting
License	AFL v2.1

FIGlet is a computer program that generates text banners, in a variety of typefaces, composed of letters made up of conglomerations of smaller ASCII characters.

Being free software, FIGlet is commonly included as part of many Unix-like operating system (Linux, BSD, etc.) distributions, but it has been ported to other platforms as well. The official FIGlet FTP site includes precompiled ports for the Acorn, Amiga, Apple II,

Atari ST, BeOS, Macintosh, MS-DOS, NextStep, OS/2, and Windows platforms, as well as a reimplementation in Perl (Text::FIGlet). There are third-party reimplementations of FIGLet in Java (including one embedded in the JavE ASCII art editor), JavaScript and PHP. FIGlet was featured as a Debian Package of the Day in 2007.

Behavior

FIGlet can read from standard input or accept a message as part of the command line. It prints to standard output. Some common arguments (options) are:

- `-f` to select a font file.
- `-d` to change the directory for fonts.
- `-c` centers the output.
- `-l` left-aligns the output.
- `-r` right-aligns the output.
- `-t` sets the output width to the terminal width.
- `-w` specifies a custom output width.
- `-k` enables kerning, printing each letter of the message individually, instead of merged into the adjacent letters.

FIGlet based ASCII typefaces

Eric Olson's 2002 FIG typeface family is a series of OpenType fonts similar to the output of FIGlet.

Chapter 12

.nfo

.nfo

Screenshot of a .nfo file

Filename extension	`.nfo`
Internet media type	`text/x-nfo`
Type of format	Plain text

.nfo (also written **.NFO** or **NFO**, a contraction of "info", or "information") is a commonly used three-letter filename extension of ASCII or extended ASCII text files that accompany other files and contain information about them. Such **NFO files** can be viewed with text editors or dedicated NFO viewers. They commonly also contain elaborate ASCII art.

Content of NFO files

NFO files usually contain release information about a software program. They are commonly associated with warez groups who include them to declare credit of and "bragging rights" over said release. Similarly they are often found in demoscene productions, where the respective groups include them for credits, contact details, and the software requirements.

NFO files were common, and sometimes required, during the era of the BBS. A typical warez NFO file was elaborate and highly decorated, and usually included a large ASCII art logo along with software release and warez group information. The designers of these NFO files frequently incorporated extended ASCII characters from the then near-ubiquitous code page 437 character set in the file.

Before Windows 95 was introduced, NFO files also sometimes used ANSI-escape sequences to generate animated ASCII art (ANSI art). These animations, however, required ANSI.SYS to be loaded by the DOS shell. If the user's computer wasn't already configured to load the ANSI.SYS driver, viewing ANSI art required reconfiguring and

rebooting. Because of this, ANSI art was much less common, and getting ANSI art to display correctly on a Windows 95 PC often proved more difficult, leading to a decline of such art in NFO files.

As of 2009, NFO files can still be found in many ZIP archives. In modern day warez NFO files, a large ASCII art logo is frequently shown at the top, followed by textual information below. Instead of using the old code page 437 extended ASCII characters, modern ASCII art uses the current de-facto web standard ISO-8859-1/ISO-8859-15 or Unicode UTF-8 characters.

Compatibility problems

Because the once-ubiquitous ASCII code page 437 was never common on the World Wide Web and is poorly supported by most (as of 2007) modern computers, older NFO files are frequently rendered incorrectly in modern web browsers and on modern operating systems (which mostly no longer use code page 437). To display old NFO files as intended, a dedicated CP437-capable viewing software or conversion is often required.

An added problem lies in the fact that many modern web browsers and text editing programs often use proportional fonts, whereas the ASCII art included in both old and new NFO files is heavily dependent on the file being viewed with a fixed width font. As a workaround, simply using a text editor and selecting a monospace font may, in the absence of incompatible extended ASCII characters, sometimes be sufficient. In this case, the only difference might be that the text in GUI text editors will often be black on a white background (rather than white on a black background as seen when viewing in MS-DOS or Unix console), thus making some of the art appear to be "inverse", like a film negative.

Canonical origins

NFO files were first introduced by "Fabulous Furlough" of the PC organization The Humble Guys, or THG. Such organizations are also known as warez groups or crack groups. The first use came in 1989 on the THG release of the PC game Bubble Bobble. This file was used in lieu of the more common README.TXT or README.1ST file names. The perpetuation of this file extension legacy was carried on by warez groups which followed after THG and is still in use to this day. Hence its strong presence on Usenet newsgroups that carry binaries and on P2P file trading networks.

The Humble Guys later became a demogroup, thus bringing the .nfo file tradition into the demoscene.

File association

On Microsoft Windows, the NFO filename extension is associated with a Microsoft software tool called *System Information* (msinfo32.exe). System Information provides a general overview of a computer's system specifications as well as detailed information on

the system's hardware components. It even includes information about the Windows environment. NFO files that are meant for System Information simply contain all of the information that System Information displays and are saved in an XML format.