Signwave Auto-Illustrator $^{1.1}$

for Carbon Mac OS 8, 9 and X, or Windows 98, NT, 2000 and XP.

Users Guide

Auto-Illustrator Users Guide

Signwave Auto-Illustrator 1.1

http://www.auto-Illustrator.com/

Signwave UK
2nd Floor North
Rutland House
42-46 New Road
London
E1 2AX
United Kingdom

+44(0)7711 623498
mail@signwave.co.uk
support@auto-Illustrator.com

Legal notice

Published by Liquid Press (Institute of Digital Art & Technology) and Spacex in conjunction with the GENERATOR exhibition, organised by Spacex and STAR (Science Technology Art Research) with support from I-DAT and the Arts Council of England.

ISBN 0950751693

TABLE OF CONTENTS

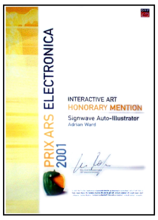
INTRODUCTION

Signwave Auto-Illustrator is a vector-based generative graphic design application for Macintosh and Windows based computers. It is also a software artwork, meaning that it exists not only as a tool for your own use, but also as the result of the creative expression of an artist. This document reveals some of the intentions of Auto-Illustrator as an artistic project, as well as showing you more practically how to use the software.

DIY

Winner of the 2001 Transmediale Artistic Software award

Winner of the 2001 Overall Cubies Awards, Real Software.



PRIX ARS ELECTRONICA 2001

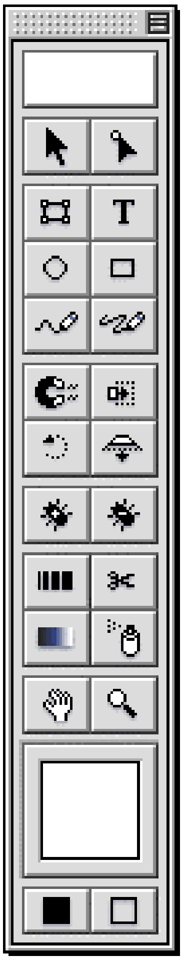
INTERACTIVE ART
HONORARY MENTION
Signwave Auto-Illustrator
Adrian Ward

[illegible]

Honourary mention at the 2001 Prix Ars Electronica Interactive Art category

USING AUTO-ILLUSTRATOR

The Tool Palette



Most of the main functions for manipulating artwork are done using the tools available in the floating tool palette.

Many of the tools are similar to their counterparts in other graphic design applications, although most of them differ slightly and operate in unpredictable ways.

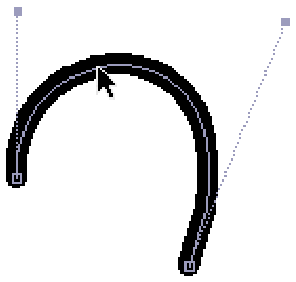
The following pages introduce each of the tool's functions, and describes what sort of effects you can achieve with them.

However, it is important you play with these tools for yourself, to discover exactly what you can do with them, rather than just believing what it says in this manual.

Select Tool

Select objects in your document

Use the select tool to pick out individual objects in your document and move them about. Selected objects are highlighted with a blue hairline, and control points are made visible.



To select a single object, simply click on it once. Where objects overlap each other, the topmost object will be selected.

To select multiple objects, hold down shift while clicking objects. Shift-clicking on an object that is already selected causes it to be unselected.

To quickly select multiple objects in one area, drag a *rubber rectangle* around them. To do this, move your cursor to a location outside any objects, and hold down the mouse button while moving your mouse across your document. Lift the mouse button when all the desired objects are selected.

To move selected objects, simply move your mouse on top of one of the selected objects. Now push down on your mouse button and move the mouse - the objects will be dragged along with the mouse.

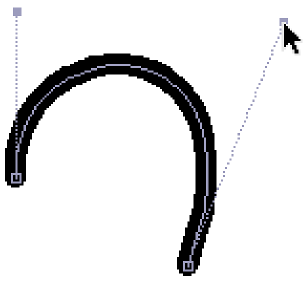
Note

While dragging objects around, you will only see a blue outline version of the currently selected objects. You can change this behaviour by going to the Interface group in the Preferences dialog and unchecking Outline dragging.

Point Tool

Move control points of selected objects

Use the point tool to manipulate the appearance of individual objects in your document. When an object is selected, you can see the individual control points shown as little blue squares between the connecting outline. Splines also show extra control points that extend on dotted handles away from the curve they control.



To move a control point, simply move your mouse over it, and hold down the mouse button while dragging it to a new location. The selected object will change shape accordingly.

For splines, each segment has two control points that define the shape of the curve between the start and the end of the segment. Changing the position of these spline 'handles' will change the shape and nature of the spline segment. Moving one of the regular segment (non-handle) control points will also move the associated extended spline handles - this is to preserve the appearance of your splines.

Polygon Tool

Create new shapes

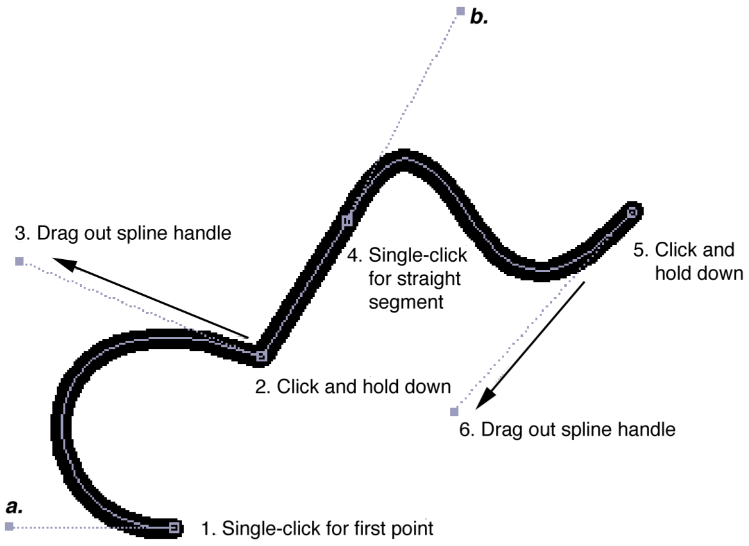
Use the polygon tool to add new shapes to your document. When first selected, the polygon tool is used to add the first point of a new shape into your document - you will not see anything at this stage, because a shape is defined as two or more points.

The next click will produce a straight line between the first point and the second. Subsequent clicks keeps adding straight line segments.

Creating Spline Segments

To create a curved spline between the last point you clicked, and the next one you are about the put down, don't lift the mouse button straight away. Instead, drag the mouse away from the new point slightly - a spline will be created. Continuing to move the mouse controls the position of the second spline control handle, which affects the appearance of the new spline segment.

You can mix and match straight and spline segments in the same shape. Just remember to click-and-release for straight segments, and click-and-drag for spline segments. Also remember you can always use the *Point Tool* to change control points if you don’t get them right first time round.


b.
3. Drag out spline handle
4. Single-click for straight segment
5. Click and hold down
2. Click and hold down
6. Drag out spline handle
a.
1. Single-click for first point

*The sequence of events resulting in an unclosed polygon containing both straight and spline segments. Spline handles **a**. and **b**. were adjusted afterwards using the point tool.*

Filled or Hollow shapes

Use the *Filled* and *Hollow* buttons at the bottom of the tool palette if your newly created shape doesn't appear how you want it. Doing this doesn't interfere with the process of creating new shapes. You can switch between the filled and hollow states while you are creating your polygon.

Closing Shapes

You can choose to leave a shape open (ie, the last point in the shape does not meet the first), or you can close your shape by clicking again on the first point you placed. Note that filled shapes always look closed by nature, but might not actually contain that last control point to close the shape. This subtly affects the behaviour of other tools and functions in the software, so it is important to understand that just because a shape looks closed, it might not actually be so.

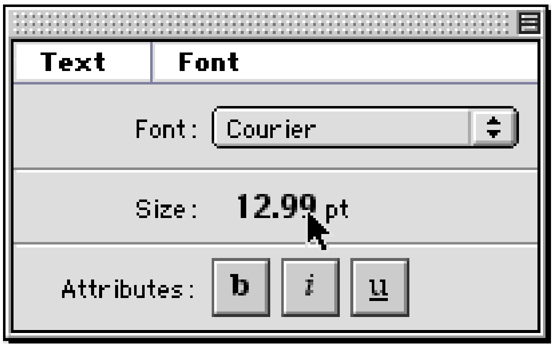
You can always automatically close shapes by using the *Close incomplete shapes* option in the *Object* menu, under the *Path* submenu.

Text Tool

Add text to your document

To add some text to your document, simply take the text tool and click once where you want the text to start. Typing on your keyboard adds characters to this text.

You can alter the settings for your text objects by clicking on the *Font* tab in the *Text options* floating palette. This palette only appears when the text tool is selected, and makes changes to all currently selected text objects. To make changes to the text point size, place your mouse over the size label, hold down the mousebutton and drag up and down.



Font: Courier
Size: 12.99 pt

Using the delete/backspace key changes behaviour when you are editing a piece of text - while there are characters in the selected text object, the delete key removes the last character. When all characters have been deleted, and the text object is empty, pressing the delete key removes the text object altogether from your document.

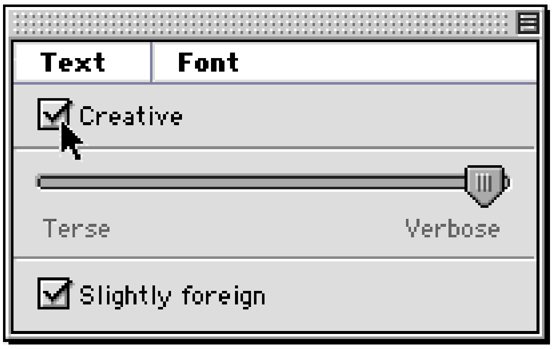
Generative Feature

If you have the *Creative* option ticked in the *Text Options* floating palette (it appears when you select the text tool), the text that appears and the characters you type will be decided by the software. It likes to make up words. Sometimes they make sense.

mnuechoemiats

lloeriaraeq

Luckily, you get some degree of control over this. You can adjust the size of words created by the creative text tool by changing the *Verbosity* slider. You can make words slightly more foreign sounding by ticking the *Slightly foreign* option.


Text | Font

☑ Creative

Terse — Verbose

☑ Slightly foreign

About creative text

The creative text tool applies a weighted stochastic selection algorithm to a predetermined set of vowels, consonants, and certain phonemes to generate words. The weighting of this selection is designed to favour more common combinations found in regular English, although their construction is purely random.

Using the slightly foreign feature substitutes the selection set of combinations for ones that use extented characters not normally found in the English language. This alternative set does not favour any particular language or syntactical arrangement.

Oval Tool

Draw ovals and other round shapes

The oval tool creates round shapes in your document. To use, simply move your mouse cursor where you want the upper-left edge of your oval, and holding down the mouse button, drag an oval out until it matches the shape you require.

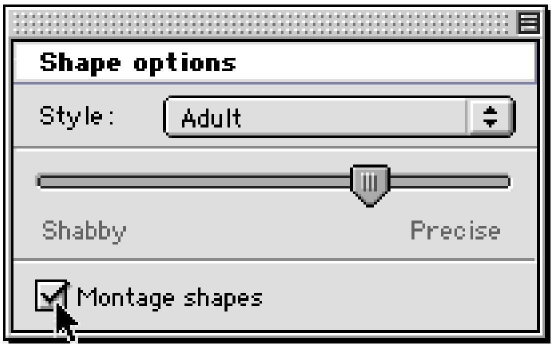
If you hold down the *SHIFT* key while you drag out your oval, the oval will retain an equal aspect ratio - that is, its width will be the same as its height. For ovals, this results in a circle.

Note

The oval tool creates a basic path made up of four spline segments. You need to be aware of this if you intend to modify or mutate your oval shapes.

Generative Feature

The *Shape options* floating palette contains a number of settings that affect the appearance of shapes that you create using the Oval and Rectangle tools.



Style: Adult

✓ Montage shapes

Style

You can choose from three different styles when producing your shapes:

Childish

The childish Auto-Illustrator does not draw plain shapes such as ovals and rectangles, but instead prefers to create smiley faces and houses. Every time you use one of the shape tools, the picture generated will be slightly different.

Adult

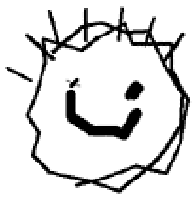
Adult shapes are boring and as you'd expect.

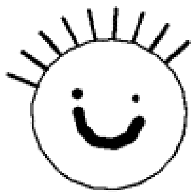
Artistic

The artistic Auto-Illustrator chooses to make a conceptual statement about the dematerialisation of the artwork in the age of digital technology. No visible changes to your document will be seen when you use the artistic shape tools, this indicates a reaction against the usual deterministic paradigm usually associated with software (and, to a degree, graphic design also). You may choose to use this style when you are producing conceptual artwork.

Preciseness

You may determine how precise the generated shapes are by changing the value of this slider.


Shabby childish ovals

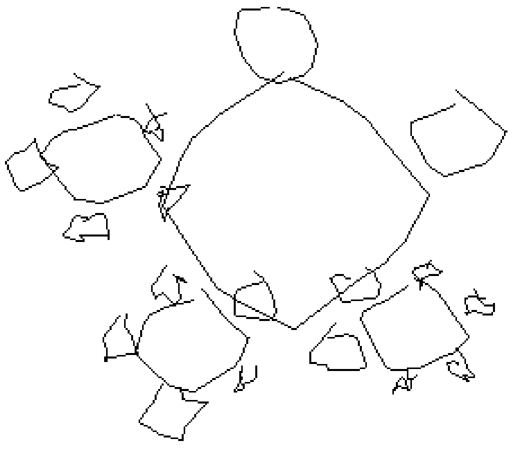

Precise childish ovals

Note

Changing the Preciseness value doesn't affect Artistic shapes. Conceptual art doesn't concentrate much on technique.

Montage shapes

To have Auto-Illustrator montage multiple shapes into your document for you, check the *Montage shapes* box. After you have created your shape, Auto-Illustrator will augment your placement with additional shapes that surround your shape.



Auto-Illustrator deploys a recursive montaging routine with a stochastic decay factor for each child node of the shape - in some cases you may find very few surrounding shapes, other times there may be many. Often, the size of your original shape makes a big difference: large shapes usually produce a greater number of child node shapes, because it takes longer for the decay factor (which regulates the size of the child nodes) to fall below the threshold that determines whether a child node will breed further children.

Rectangle Tool

Draw rectangles and other square shapes

The rectangle tool creates rectangular shapes in your document. To use, simply move your mouse cursor where you want the upper-left edge of your rectangle, and holding down the mouse button, drag a rectangle out until it matches the shape you require.

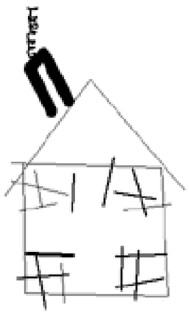
If you hold down the *SHIFT* key while you drag out your rectangle, the rectangle will retain an equal aspect ratio - that is, its width will be the same as its height. For rectangles, this results in a square.

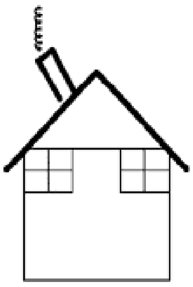
Note

See the Oval Tool for details about the generative nature, styling and montaging options for the Rectangle Tool – they are the same.

Preciseness

You may determine how precise the generated shapes are by changing the value of this slider.





Shabby childish rectangles   Precise childish rectangles

Pencil Tool

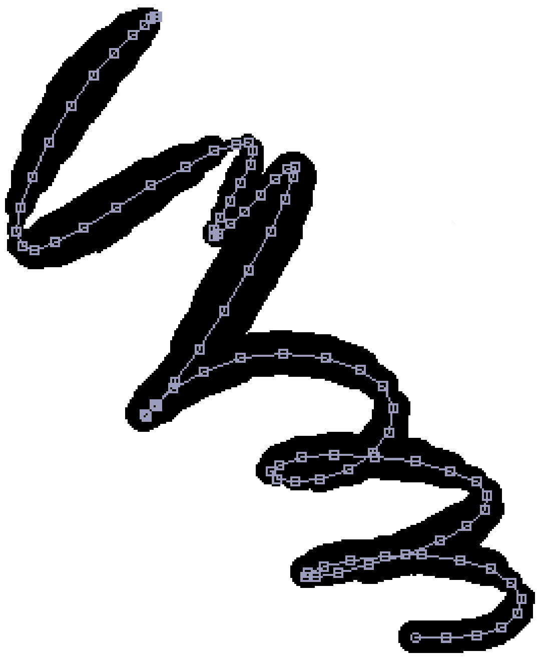
Draw freehand shapes

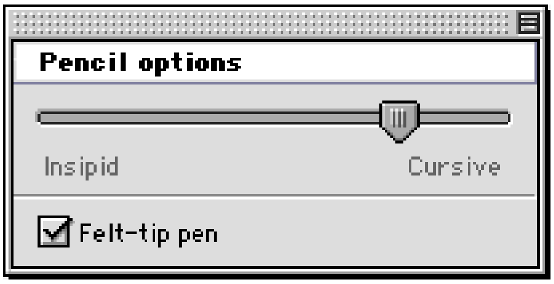
You can use the pencil tool to draw freehand paths into your document. There are two different ways that this is achieved, and both will affect how you modify and change these paths later - You should read the *Felt-tip pen* section below for clarification.

To draw a path using the pencil tool, move your mouse to where you want the path to begin, and holding down the mouse button, move your mouse around your document, following the path you would like the pencil tool to make.

Generative Feature

When you use the pencil tool, Auto-Illustrator doesn't actually use your mouse coordinates to generate the path directly. Instead, it merely takes cues from your mouse coordinates and has its own rules for how the path it draws should look.





[✓] Felt-tip pen

The *Insipid/Cursive* slider controls the behaviour of this routine. If you change the setting to be very cursive, you will find that the pencil tool prefers to make grand, sweeping gestures that deviate wildly from your original mouse coordinates. Insipid use of the pencil tool produces lame and limp paths.

Felt-tip pen

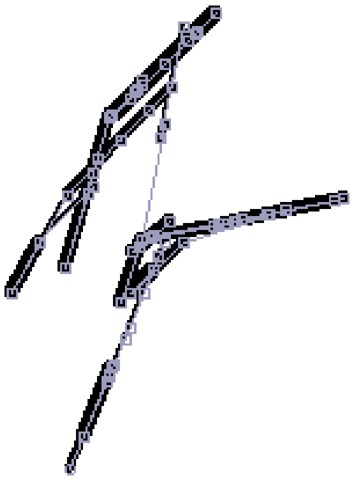
Checking the felt-tip pen option changes the way the pencil tool creates path objects. When checked, the stroke weight of each segment of the path will vary according to how fast the pencil was moving at that time, like pushing down harder on a felt-tip pen may produce bigger lines. As this stroke weight varies between segments, it is necessary to produce this path as lots of individual line objects that are grouped together. This is important to understand if you later try to use some of the distortion and transformation tools, because the line will break apart easily.

Non-felt-tip pen pencil paths (!) are, on the other hand, created as single objects and will not break apart later on. However, this means that the path will have one single stroke weight, which is determined by using the *Stroke Inspector* floating palette. These paths are actually un-closed polygons, and are identical in structure to those produced using the *Polygon tool*.

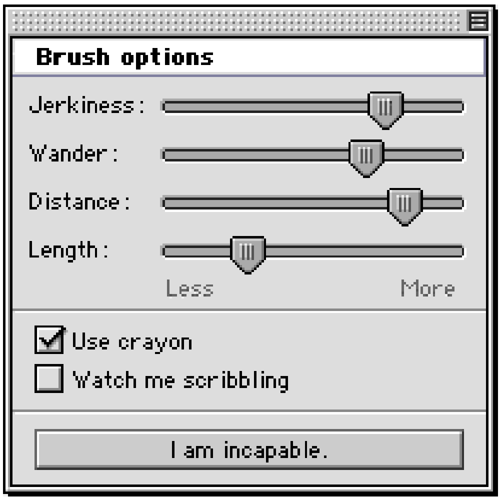
Brush Tool

Make expressions with a brush

The brush tool is a completely autonomous drawing tool that makes marks on your document. When you click somewhere in your document, Auto-Illustrator uses a brush to draw stochastic marks by itself.



You can change the behaviour of the brush tool by changing the settings in the *Brush options* floating palette.



Brush options

Jerkiness:
Wander:
Distance:
Length:
Less More

[✓] Use crayon
[ ] Watch me scribbling

I am incapable.

Jerkiness

As Auto-Illustrator moves the brush across your document, it occasionally jerks the brush in the opposite direction. This slider dictates the chance that this occurs on every movement of the brush. A high jerkiness value means that the brush will jerk every time it is moved, a low jerkiness value produces straighter brush marks with no jerks.

Wander

Similarly, the wander slider affects how much wander will be introduced into the brush path at every movement. Low wander values cause straight lines that follow on from previous segments, high wander values make the brush deviate from that path more.

Distance

The distance slider affects how far the brush should move for each segment. Low distance values produce very short stuttered marks, high values cause long strokes that can span the entire dimensions of your document.

Length

The length slider controls for how long the brush should be used each time. The higher this value, the more marks the brush will make in your document.

Use crayon

The crayon option (formerly known as *Crayola™ Compatability Mode* in Autoshop) causes the brush marks to be rendered with a crayon with varied stroke weight at each segment.

Watch me scribbling

Checking this causes Auto-Illustrator to refresh the document window at each segment drawn, so you can see the brush tool drawing your artwork in real-time.

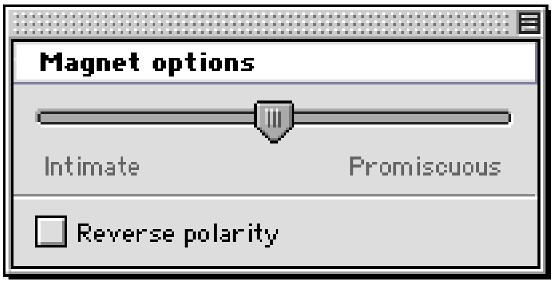
I am incapable

Pushing this button allows Auto-Illustrator to decide settings for itself. Use this button if lacking inspiration.

Magnet Tool

Repels and attracts object control points

The magnet tool is a simple but effective way of distorting the control points that make up the objects in your document. When selected, you can click and drag around your document to see control points reacting to the magnet tool when they are within a certain range of influence.



Promiscuity

Changing the *Promiscuity* slider affects the strength of the magnet. Intimate magnets are weaker and affect only very local control points, whereas promiscuous magnets prefer to exert force on a wider field, and are stronger. However, you may find promiscuous magnets difficult to control, producing undesired results if not controlled properly.

Reverse polarity

This does exactly what it says.

Parallax Tool

Parallax scrolling

If you have objects in your document of varying stroke weights, you can slide them about your document at varying speeds, producing a parallax style distortion that implies a third dimension (Z-depth) that goes into and outwards from your monitor.

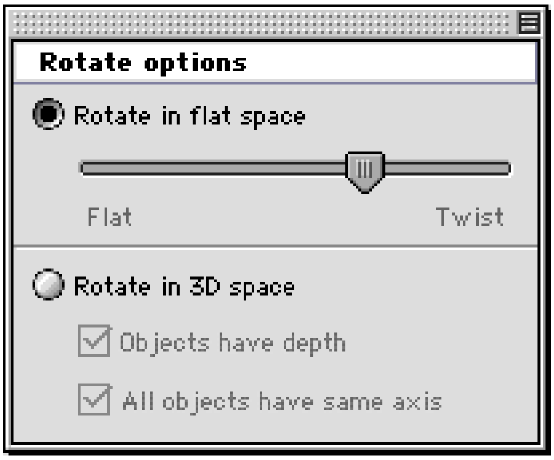
Shapes that do not vary in stroke weight will not slide in parallax.

There are no options for this tool.

Rotate Tool

Twist your document

The rotate tool is a powerful and expressive system for transforming your documents in abstract ways. There are two different ways you can use the rotate tool, one of which may be familiar to you.



Rotate options

● Rotate in flat space

Flat ——————————— Twist

○ Rotate in 3D space

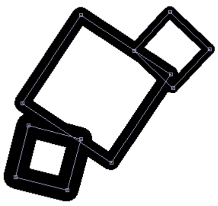
✓ Objects have depth

✓ All objects have same axis

Rotate in flat space

Flat rotations occur in 2D. Select one or more shapes you want to rotate using the select tool, and then select the rotate tool. Ensuring the *Rotate options* floating palette shows the *Rotate in flat space* option, click down anywhere in your document and drag the mouse left or right. The selected objects will rotate around their centre point. This centre point is determined by the average coordinates of all selected objects, if you select different objects the centre point of rotation will be different.

Flat / Twist



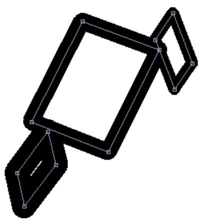
Changing this slider changes the behaviour of flat rotations. When set to twist, objects with thicker stroke weights rotate quicker than thinner stroke weighted objects. If you have several objects all with varying stroke weights (such as produced by the *Weight Tool*), then this will twist your artwork in interesting ways.

Rotate in 3D space

3D rotations are a little more complex computationally, but as a general rule, they extend rotational transformations into a fictional 3rd dimension (a Z-depth), which projects into and out from your monitor.

Objects have depth

When you check this option, 3D rotations assume that an object's Z-depth is determined by its stroke weight, much like the flat rotation's twist feature. Again, objects with varying stroke weights will produce exciting and unexpected results, radically transforming your document into a pseudo-3D space demonstrating the impact of real-world spatial transformations in a 2D-based architecture model.



All objects have same axis

Checking this option ensures that all 3D rotations occur around a single centre-point (the average coordinate of all selected objects) rather than each object having its own centre point for rotation. Playing with this feature in combination with the *have depth* feature results in amazing transformations that decouple one object's 3D space with another while maintaining a uniform spatial transformation property.

Scale Tool

Parallax scaling

Much like the *Parallax tool*, the scale tool changes the scaling of the document, but applies different scaling ratios depending upon the stroke weight of the selected objects in your document. Doing so implies a third dimension (Z-depth) that goes into and outwards from your monitor.

Shapes that do not vary in stroke weight will not scale in parallax.

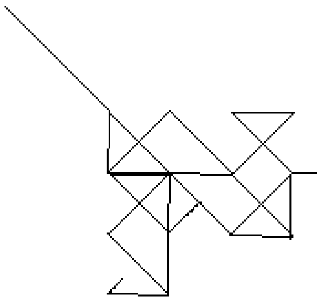
If you hold down the option/alt key while using the scale tool, the parallax scaling feature is disabled, and the Scale Tool behaves just like an ordinary scale tool.

There are no options for this tool.

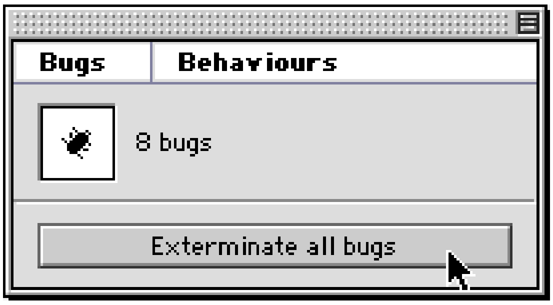
Bug Add Tool

Crawling bugs can draw your artwork for you

You can place bugs onto your document that will wander around and leave lines behind them. When bugs bump into objects in your document, they usually turn in a different direction and keep walking.

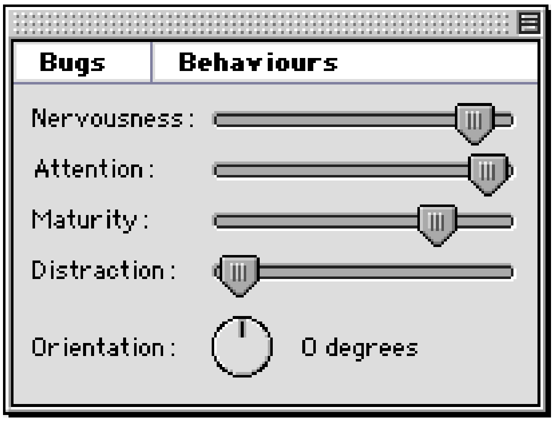


If you place several bugs down on the same document, they'll all start bumping into each other's tracks, and produce a lovely mess.



The *Bug options* floating palette shows you quickly how many bugs are currently wandering around your document, and gives you a quick and easy way to exterminate them all should they be getting too unruly.

Bug Behaviours
By pressing the *Behaviours* tab on the *Bug options* floating palette, you can control the way the bugs behave by adjusting the sliders.



Nervousness
Nervous bugs run quickly.

Attention
A bug that is not paying attention might actually ignore something it walks over, and not change direction as it is supposed to.

Maturity
Mature bugs have learnt not to be so hasty in changing direction - they will wait a bit longer before deciding to change direction. They are less impulsive.

Distraction
A distracted bug is not looking where it is going, and might actually change direction when it is not actually bumping into something in your document. This can be problematic, and you may end up with lots of tiny squiggles because your bugs are too distracted and are always walking in different directions.

Orientating
You can change which direction is North by changing this compass.

Note
Use the Bug Remove Tool to remove bugs you have previously added to your document.

Bugs get saved in their current state when you saved your document. As soon as you open the document again, the bugs will continue where they were left last time you were editing the document. Watch out for that - your documents may not open in the state that you expect them to!

Bug Remove Tool

Remove bugs from your artwork

You can easily debug your document by selecting this tool and clicking on your document. For each click, the last bug you added will be removed.

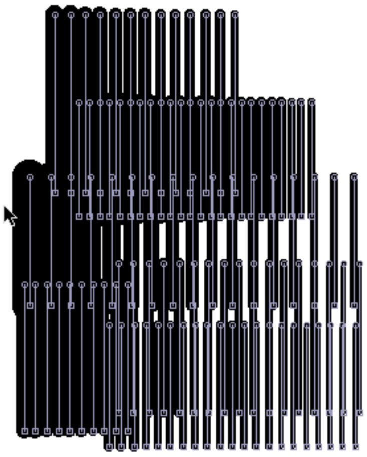
If you want to remove all bugs at once, either click repeatedly with the Bug Remove tool until you get a message saying there are no more bugs, or use the *Exterminate* button in the *Bug options* floating palette, which commits mass genocide with one easy click of the mouse.

Sadly, we were unable to get this tool to automatically debug our own software. It should only be used to debug Auto-Illustrator documents. We cannot guarantee usability for any other form of debugging.

Stroke Tinting Tool

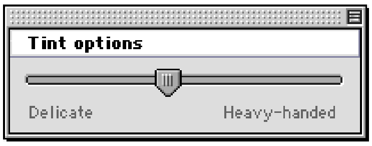
Quickly restroke your objects

When you drag this tool over your document, selected objects will be "tinted" with new stroke weights according to how close they are to the stroking tool.



This is a great way to quickly change the dynamic properties of your document, because other tools such as the *Rotate Tool* and the *Parallax* tools use the stroke weight of each object to determine how they should be distorted.

This is the recommended method for the quickest way to generate interesting dynamic artworks.



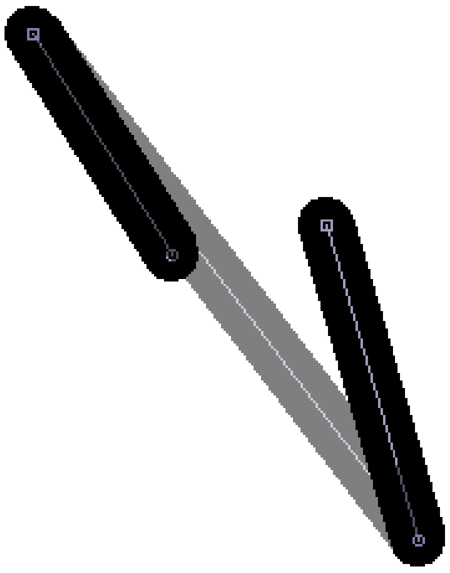
Tint options

By adjusting the *Delicate / Heavy-handed* slider, you can affect how strong the weight tinting tool is. A delicate tinter causes only very close objects to be tinted strongly, whereas heavy-handed tinters are careless, and tend to affect more of the document's objects.

Scissors Tool

Chop up your artwork

The scissors tool can be used to cut paths and edges of polygons in half. Simply click on a path where you want it to be cut in half, and the segment will split in half like a piece of elastic.

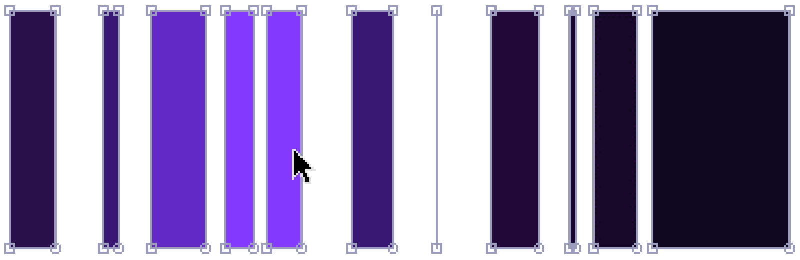


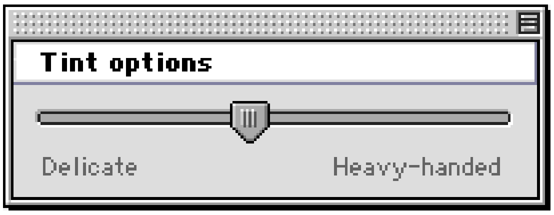
The scissors tool comes with an automatic multimedia effect. To disable this digital multimedia effect, switch off your computer's speakers, or cut the speaker cable with a pair of pliers if there is no switch.

Tinting Tool

Quickly tint your objects

Much like the *Stroke tinting tool*, the tinting tool applies the current selected colour in a gradient fashion to the objects in your document. Objects closer to the tinting tool are brighter in colour, objects further away slowly fade away to black.





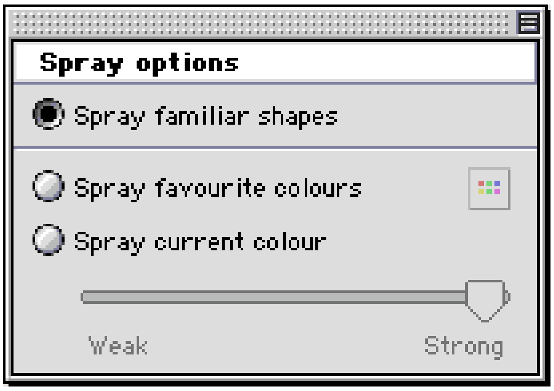
Tint options

By adjusting the *Delicate / Heavy-handed* slider, you can affect how strong the tinting tool is. A delicate tinter causes only very close objects to be tinted strongly, whereas heavy-handed tinters are careless, and tend to affect more of the document's objects.

Spray Tool

Apply repetitions

The spray-can tool can be used in several different ways. In general, it is useful for applying an effect or transformation to the objects in your document in a quick and easy manner.



Spray options

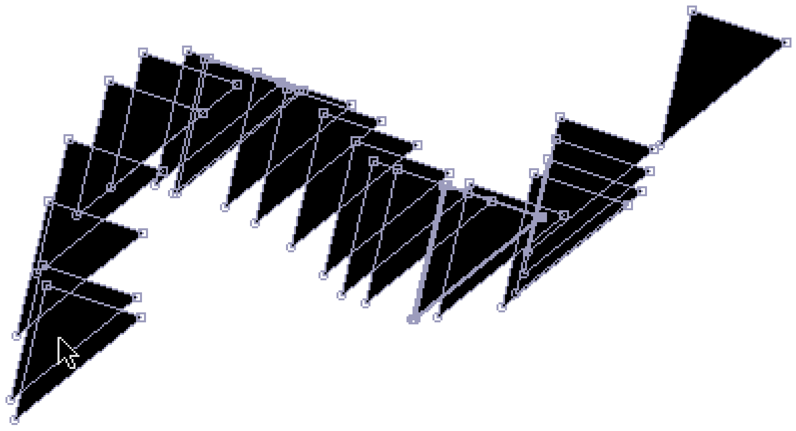
[✓ Spray familiar shapes]

Spray favourite colours

Spray current colour

Spray familiar shapes

Using the spray-can with this option allows you to quickly duplicate shapes that are already in your document. When you click and drag around your document, shapes that you have previously created are duplicated underneath your mouse cursor.



Note

If you haven't already designed any shapes in your document, using the spray-can like this pointless. It sprays nothing onto nothing, and creates nothing from nothing.

Spray favourite colours

This option takes colours from your *Colour Swatch*, and applies them to the objects in your document. Wave the spray can over your document objects to colourise them with your favourite colours.

Spray current colour

Very similar to the *Spray favourite colours* option, except this sprays the current colour onto objects in your document.

Weak / Strong

Change this slider to alter how strong your spray can is.

Hand Tool

Scroll around your document

This tool makes it easy to scroll around your document without using the scroll bars. Simply click and drag in your document to move the current view around. Note that this doesn't actually change the positions of objects in your artwork (use the *Select Tool* for that).

You can hold down the SPACE key before clicking and dragging to quickly switch to the hand tool.

Double-click the hand tool in the tool palette to automatically zoom the document so that it fits in the document window.

Zoom Tool

Zoom in and out

Using this tool you can zoom in and out of your document, inspecting details as you wish. Simple click somewhere in your document to zoom in by 200% to that part, and alt-click to zoom back out.

You can also use the slider in the lower-left corner of the document window to achieve this. There are various shortcut keys and menu options in the *View* menu that switch to pre-set zoom views.

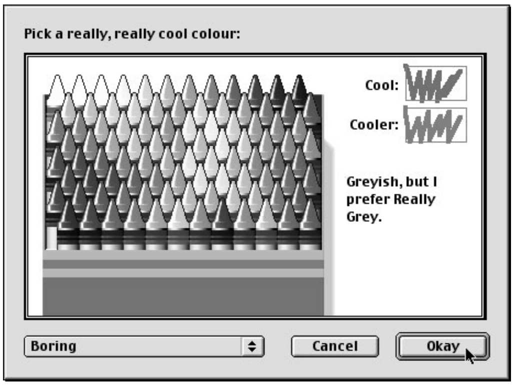
Double-click the Zoom Tool in the tool palette to have the document automatically zoom to 100%

Colour Picker

Select colours

Click on the colour preview in the tool palette to choose a new colour. If you have any objects selected when you do this, those objects will take on the new colour you choose.

Conversely, when you select objects in your document, the current chosen colour changes. If you select more than one object, then the colour shown in the tool palette is the average colour of all the selected objects. This can be a useful feature if you want to average out all the colours in your document.


Pick a really, really cool colour:

Greyish, but I prefer Really Grey.

Generative Feature

When you choose a colour from the crayon colour picker (how useful!), Auto-Illustrator will actually decide if the colour you picked was suitable. If it doesn't think your choice was very good, it will suggest a better, but similar colour. You don't get any choice in this matter. You have to go with Auto-Illustrator's suggestion, whether you like it or not.

Note

You can disable the generative feature by going the General options in the Preferences dialog, and tick the Always agree with user's choice.

For the truly boring Auto-Illustrator user, your computer system's colour picker can be used instead. Tick the Use System colour picker option in the same Preferences dialog.

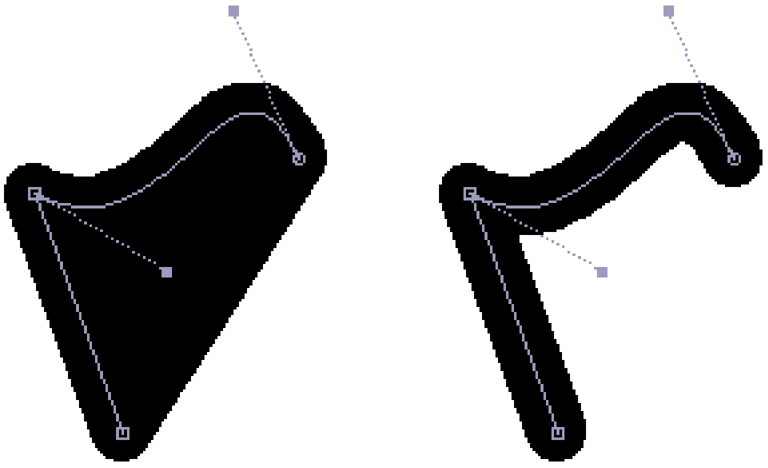
Fill Selector

Fill selected polygons

Any objects in your documents that are polygons (whether closed or un-closed) can be set to be filled in. This means that the polygon will appear to be closed, and will be filled with the object's colour.

Note

It is important to know that even though a shape may look closed, it might not actually be so. For filled shapes, this does not make any difference unless you are planning on transforming your shape using one of the many filters and plugins, as these often operate on the individual segments between the control points of an object. Because (in the example below) the last line doesn't exist, if this shape were to be placed through the Stupid and Pointless filter, for example, only the left and top sides would be affected, because the right side isn't actually real - it just appears to be so because it is filled.



Hollow Selector

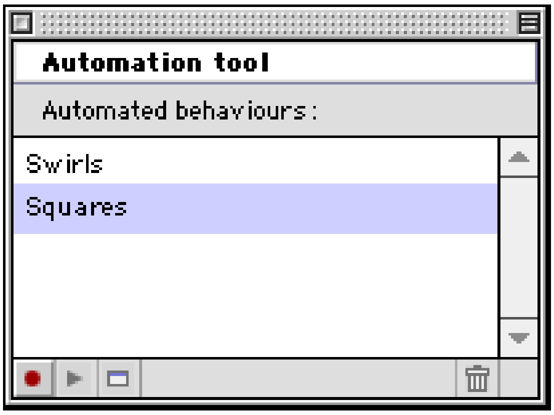
Hollow selected polygons

The Hollow Selector reverts shapes back to their unfilled states.

Advanced features of Auto-Illustrator

There are a number of tools included with Auto-Illustrator that may only occasionally be utilised for specific tasks.

The Automation Tool

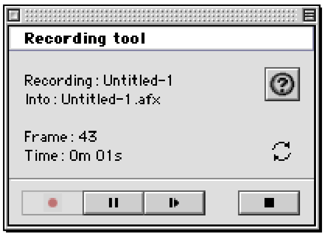


If you learn to achieve a certain visual effect using a tool in a certain way, you can record that behaviour using the Automation Tool. Press the red Record button to create a new Automation, and type a name for it. All mouse activity and tool usage will be recorded in a new Automation. Stop recording by pressing the Recording button again.

To replay an existing Automation, select it from the list, and click the triangular Play button. The actions recorded in that Automation will be replayed. You can use mouse activity from a recording with one tool on another.

Note
Automations are saved as .xeo files in the Automations folder. You can move these to the Plug-Ins folder to turn then into Plug-Ins if you wish.

The Recording Tool


Recording tool

Recording : Untitled-1
Into : Untitled-1.afx

Frame : 43
Time : 0m 01s

You can produce files that record changes in an Auto-Illustrator over time. These can be imported into Signwave Auto-Effects later and used to produce QuickTime™ videos or Shockwave™ Flash animations.

Signwave Auto-Effects can be downloaded for free from
http://www.auto-effects.com/

Press the red Record button in the Recording Tool to start a new recording. Provide a location and name for the new .afx file. Recording proceeds immediately using the currently open document.

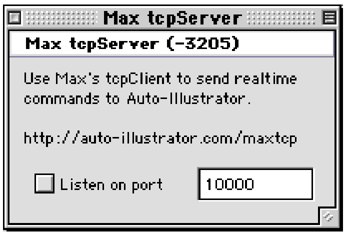
Make your changes as desired. You can use the Pause button and Step button to temporarily suspend and resume recording, in order to use stop-frame animation techniques.

When you've finished your recording, press the Stop button. The .afx file will be written to disk. Auto-Effects can read .afx files and can be used to edit then accordingly.

Note

At the time of publishing, Signwave Auto-Effects is still at beta version 0.1. It is freely downloadable from the URL shown above.

The Max tcpServer / pd netreceive Tool


Max tcpServer

Max tcpServer (-3205)

Use Max's tcpClient to send realtime commands to Auto-Illustrator.

http://auto-illustrator.com/maxtcp

☐ Listen on port 10000

If you use Max (**http://www.cycling74.com/**), you can use the tcpClient object to send real-time data to Auto-Illustrator 1.1 or higher. This can be done over any TCP/IP network, including the Internet – providing no restrictive firewalls are in place. For situations where a firewall might be in use, alter the port number to one that your Network Administrator allows incoming connections to.

The data is sent to Auto-Illustrator in the same format as a xeoObject senddata call. See the xeoObject reference for details of the data format.

Further details, plus the required externals for Max and a demo max patch can be found at
http://www.auto-Illustrator.com/maxtcp/

Signwave would like to thank Jasch of **http://kat.ch/** for his kind assistance in the creation of this tool.

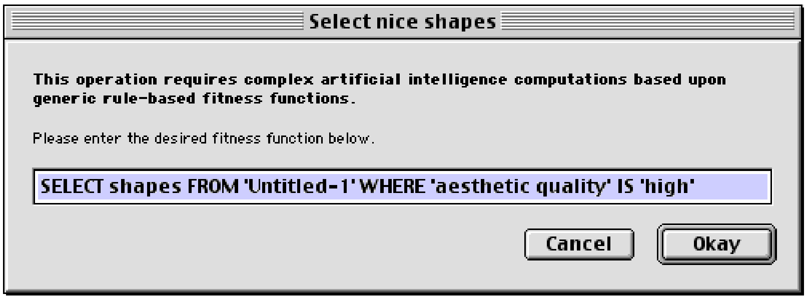
Notes

The Max tcpClient object uses a wrapper protocol to ensure data delivery. In contrast, pd's netsend object doesn't. Use the pd netreceive tool in Auto-Illustrator if you're using pd on a Windows or Linux machine to send data, instead of the tcpServer tool. The Max tcpServer Tool must only be used with the tcpClient Max object.

As the pd netreceive tool uses a plaintext network protocol, you can actually use any networkable program or language for making TCP/IP connections to Auto-Illustrator. Perl, for example, is ideal for this.

Only one incoming connection can be maintained at a time. Ensure you tell Auto-Illustrator to listen to a certain port before you get Max or pd to make a connection.

The Select nice shapes function



This operation requires complex artificial intelligence computations based upon generic rule-based fitness functions.

Please enter the desired fitness function below.

SELECT shapes FROM 'Untitled-1' WHERE 'aesthetic quality' IS 'high'

In the Edit menu, under the Selection submenu, you will find an option called “Select nice shapes…”. As the term “nice” is highly subjective, you are required to enter a fitness function that determines what exactly constitutes “niceness”.

This is done in the form of an SQL statement. This is entirely optional. You may choose to use any language or form of expression you want, even poetry.

The Select nice shapes function is intended to parody Artificial Intelligence research, which has a hang-up about trying to implement basic subjective reactions in a system that is fundamentally pragmatic by nature.

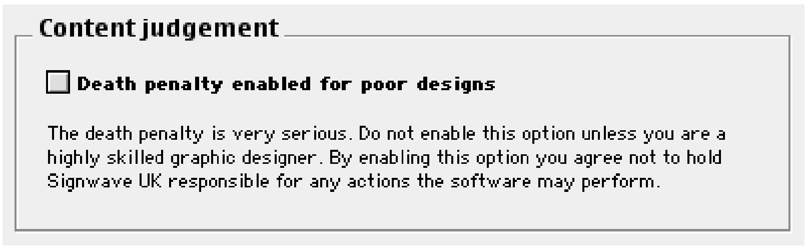
By rejecting the notion that a computer *could* actually decide what is nice (by pretending that doing so is done using tedious Structured Query Language statements), the Select nice shapes function expresses the belief that subjectivity can be embedded into pragmatic code, a fundamental assumption employed throughout the creation of Auto-Illustrator as a generative artwork.

Note

This function actually works, although details of how it does so is a corporate secret. Using the same fitness function will always return the same selection – try subtly varying the function to see different selections. Subjectivity isn’t the sole domain of the organic brain, pragmatic code can also have an idea of what constitutes “nice”!

The Preferences dialog

There are a number of advanced options available to the user in the Preferences dialog box. For example, in the Psychosis section, users can subject themselves to scrutiny by the software by using the Death Penalty option.

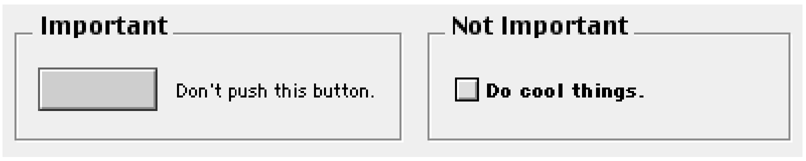


☐ Death penalty enabled for poor designs

The death penalty is very serious. Do not enable this option unless you are a highly skilled graphic designer. By enabling this option you agree not to hold Signwave UK responsible for any actions the software may perform.

When checked, Auto-Illustrator regularly considers the quality of the document being designed. Should the design not be good enough, Auto-Illustrator will erase your work and shut down your computer automatically. Employers in design bureaus may wish to use this feature to decide which graphic designers are worth employing or not. Signwave cannot be held responsible for any actions undertaken by the software when this option is used.

The Important and Not Important sections are used to enhance the end-user experience.



Users are encouraged not to push the button that says “Don’t push this button”. The “Do cool things” option is suitable for inexperienced designers who require a little further inspiration.

TECHNOLOGIES

Auto-Illustrator requires and makes use of wide range of technologies. It exists only because the visionary people at Real Software felt passionate about making REALBasic. It exists because the people that worked on MacOS at Apple thought carefully about the *right way* of doing things.

One of the most exciting technologies that Auto-Illustrator makes use of must surely by KJX™. This wholly proprietary technology is a pure mystery to all developers. It merely exists, and is used in promotional material and packing to encourage users to think that something important is going on.

Here’s some detailed information on how KJX works. Good luck. Don’t get too surprised when you figure out that KJX is pretty useless. Just keep repeating to yourself “KJX is good” until you believe it.

Introducing KJX™ Technology

KJX™ is the revolutionary new technology that allows you to get more graphical power from your vector-based object hierarchy. By taking advantage of every modern technology available to the personal computer, KJX™ can provide enhanced support for object data streams that support the KJX™ Standard.

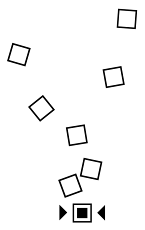
More and more people are waking up to the advantages of KJX™-Enabled technology. While non-KJX™ Systems fail to provide the enhanced peace-of-mind and operatability that KJX™ Users enjoy, software which supports the Enhanced KJX™ Standard allows a dynamic range of components to be adapted at will by developers deploying truly cross-platform OS-independent solutions.

How it works

KJX™ adapts to the data streams it is provided with. By applying non-polyeurythmic pre-redundant cyclic processing to bytes that have the potential to exist, vector flows are optimised to provide a more dynamic workflow that uses more CPU cycles than any other software technology available.

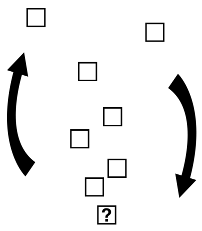
With KJX™

Information flows freely without restrictions. The correct bytes are identified before they even exist.



Without KJX™

Bytes are symmetrical, but messy. They accumulate because internal vectors move them without proper organisation.



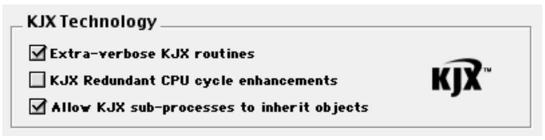
Auto-Illustrator and KJX™

Auto-Illustrator supports KJX™ and is built-in to the core systems that function within the software. This means you do not even know when KJX™ is operating, although certain optimisation facilities are available to you.

Signwave will continue to support KJX™ Technology into the future, ensuring that all Auto-Illustrator users will enjoy the benefits brought with it, while other competing products fall behind because they simply do not support KJX™ Technology.

Optimising your KJX™ System

In the Preferences dialog box, under Psychosis, you will find a number of options that allow you to take control of how KJX™ operates.



KJX Technology

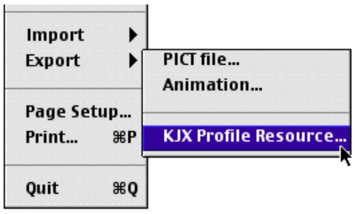
- ☑ Extra-verbose KJX routines
- ☐ KJX Redundant CPU cycle enhancements
- ☑ Allow KJX sub-processes to inherit objects

Extra-verbose routines deploy enhanced operating models for KJX enabled technology, but can inadvertently use too many CPU cycles, so you might want to disable this if you find this option doesn't work out for you.

Redundant CPU cycle enhancements fill all available CPU cycles (and more) with KJX processing objects. Keep this unchecked for enhanced performance.

Sub-processes can inherit objects at random intervals for no reason whatsoever. You can allow this to happen by keeping this option enabled. Disabling this option is futile.

Exporting your KJX™ Profile



Remember to export your KJX Profile Resource when you need it. Currently, the files produced by this feature don't work with any other software or feature. You may safely throw them away once you have saved them.

xeoObjects and Plug-In Technology

Auto-Illustrator uses a proprietary technology called xeoObjects to implement many parts of the functions of Auto-Illustrator. Predominantly, plug-ins can be written using the xeoObject scripting language, while the Automation, Max tcpServer and pd netreceive tools all use the same xeoObject data syntax for communication with Auto-Illustrator.

A basic xeoObject script defines sections of code (handlers) to execute at certain times. Some are required by Auto-Illustrator, others are custom defined. Simple keywords and parameters are used to cause data to be manipulated and used in a variety of ways. The xeoObject scripting language implements three basic variable types – a string, an integer, and a double floating-point. In addition, there is a loosely typed array object that holds lists of data of any type.

Data is received from an Auto-Illustrator document using a getdata keyword. Data can be sent back to Auto-Illustrator using senddata. There are defined syntaxes for how such data should be labelled and formatted. For example, a xeoObject can ask Auto-Illustrator to tell it the colour of the third object in the currently active document.

Note
More detailed technical specifications on the xeoObject standard can be found at ***http://www.auto-Illustrator.com/support/faqs/authors/xeoobject***

Technical Information for xeoObject Developers

Introduction

xeoObjects are small external files that describe the functionality of a plug-in or process that cannot be defined within the core Auto-Illustrator program. For this reason, xeoObjects are used for plug-ins, and also elsewhere in the application. The document is written with the assumption that you are writing a plug-in for Auto-Illustrator in xeoObject format. The content here may not be suitable for other xeoObject applications, whether for Auto-Illustrator or not.

What is a xeoObject?

A standard text file with a .xeo extention, written in a uniform language that is a perculiar hybrid of lingo and ASM, with C-style variable declarations. The language itself is very limited, with only a few operands, but this in conjunction with the flexible variable and handler style syntax, is enough to write reasonably complex plug-ins.

A xeoObject also contains instructions that tell the xeoObject runtime environment how to layout a dialog box. You can make simple interactive plug-ins in the manner. The contents of a dialog box are also limited (for example, you can only have up to 16 buttons in your dialog), but should prove sufficient.

Language Syntax

Handlers

You divide your code blocks of instructions (known as a handler), and give each handler a name. Certain handlers are called by the xeoObject runtime environment, for example, when your plug-in needs to be executed, or when a user has interacted with a control on your dialog. Other handlers you can call yourself from your own code, meaning you can split the functionality of your plug-in up into easy to maintain chunks.

The syntax to define a handler is the same as in Lingo or Applescript:

```
on handlername
  code goes here
end
```

If you want to call this handler from elsewhere in your xeoObject code, you can do so using the do keyword:

```
do handlername
```

Certain handler names are fixed, and you should make sure you have defined handlers for these names:

on open
Called when your plug-in initialises when Auto-Illustrator first starts up. In this handler, you should use some code to tell Auto-Illustrator exactly what the plug-in name is:

```
on open
  senddata Auto-Illustrator::Plugin::Name Example
end
```

on launch
Called when your plug-in is launched by a user. A user has selected your plug-in from the menu, so you should now start the main functionality of your plugin (ie, producing new shapes, setting up the dialog box if needed, etc).

Variables
Variables are a bit quirky in xeoObjects, but they are flexible. To start with, you use C-style variable definitions to define your variables as one of three different data types:

```
int    myIntegerVariable
double myDoubleVariable
string myStringVariable
```

Note that you can put these definitions anywhere in your xeoObject file - there are no such things as local or global variables - everything has the same scope, so you need to bear that in mind when creating your various handlers.

When you refer to your variables in your code, there are two ways of doing so. If you are referring to the *variable*, then you just provide its name. You do this when you want to *set* variables, or do maths such as *add* and *mul* (multiply). For example:

```
set myIntegerVariable 42
add myIntegerVariable 10
set myDoubleVariable  5.5
mul myDoubleVariable  2
set myStringVariable  Mary Had a Little Lamb
```

However, if in your code you wish to refer to the actual *value* of that variable, then you enclose the variable name in square braces:

```
add myIntegerVariable [myDoubleVariable]
```

This is a very important distinction. In the example above, the *value* of myDoubleVariable (which, I believe, we set to 5.5 then multiplied by 2, so is 11) is added to myIntegerVariable, which was 52. So myIntegerVariable is now 63.

You don't need to typecast variables. If you try adding a number onto a string, the number variable is treated as a string. (Psst, you *can* add a string onto a number, so long as the string contains a number, but that's rarely necessary).

```
add myStringVariable %20it cost%20
add myStringVariable [myIntegerVariable]
add myStringVariable %20dollars
```

The value of myStringVariable is now "Mary Had a Little Lamb it cost 63 dollars". Note that %20 was used to indicate a space. You need to do that sometimes because you don't use quotation marks with string constants (unlike other languages) and the xeoObject runtime environment removes surplus whitespace from your script before executing. Also, you cannot mix string constants with variable references, hence the need for three separate add statements in the above example.

Conditional statements

Naturally, there will be times when you only want to execute some code when some comparison of variables you are using matches your requirements. The if statement only executes the next line of code if the condition set out evaluates to true:

```
if [myIntegerVariable] > 5
  set myIntegerVariable 0
```

Note, once again, the difference between the first and second reference to the variable called *myIntegerVariable*. The first is in square braces because we are comparing the value of that variable with the number 5. It would be very tempting to not include the square braces, but the xeoObject runtime environment wouldn't evaluate that if statement correctly because it wouldn't be comparing the *value* of the variable *myIntegerVariable*. The situation is comparable to that of using pointers in C or Pascal: you have to remember to *dereference* the variable if you actually want its value. The difference here is that an un-dereferenced variable reference equals zero, not the actual pointer address as in other languages. But you really don't need to know that.

There are five different comparison evaluations:

<	for numeric less-than (ie, 5 < 6 is true)
==	for numeric equality (ie, 4 == 4 is true)
>	for numeric greater-than (ie, 7 > 5 is true)
eq	for string equality (ie, yes eq no is false)
ne	for string inequality (ie, sun ne moon is true)

If you want to execute more than one line of code, you need to split that code into a separate handler and call it:

```
on launch
  if [myIntegerVariable] == 42
    do theMeaningOfLife
end

on theMeaningOfLife
  any amount of code goes here
  any amount of code goes here
  any amount of code goes here
end
```

Another important thing to remember is that you can't nest if statements:

```
on launch
  if [myIntegerVariable] == 42
    if [myDoubleVariable] > 10
      set myStringVariable Easy!
end
```

This does not work! Why? Because if the first conditional comparison evaluates to false (ie, if *myIntegerVariable* does not equal 42) then the runtime environment skips the next line and stops at the set line, which is not actually what you intended (indentation means nothing to the interpreter)! Instead, you have to split nested if statements into separate handlers:

```
on launch
  if [myIntegerVariable] == 4
     do checkDoubleValue
end
on checkDoubleValue
  if [myDoubleVariable] > 10
    set myStringVariable Easy!
end
```

It's clumsy, but in this respect conditional statements in xeoObjects are more like ASM, because you have to think about the 'location' of the 'instruction pointer' (ie, the linenumber that is currently executing).

Loops

Luckily, loops in xeoObjects are much easier. They're just like BASIC FOR..NEXT loops, with a few minor changes to the syntax:

```
for variableName firstValue lastValue stepIncrement
   whatever code goes here
next
```

This example counts from 1 to 10, adding that number to the end of a string, and each time divides another variable in half.

```
for myIntegerVariable 1 10 1
  add myStringVariable [myIntegerVariable]
  mul myDoubleVariable 0.5
next
```

PS. Don't put a conditional if statement right before a next statement. That's really bad. I'm sure you can figure out why.

Also, watch out you don’t get your *lastValue* or *stepIncrement* figures wrong - if you're counting from 1 to 10 but your stepIncrement is -1, you'll count 1, 0, -1, -2, -3 (etc) and you'll never ever reach 10! This will lock up your computer. Don't blame us.

CRITICAL TEXTS AND ESSAYS

Auto-Illustrator follows on from about 5 years of research and creative activity. In the summer of 1999, Signwave released Autoshop 1.0, a free experimental generative painting package for Macintosh computers, which parodied Adobe™ Photoshop™.

You can download Signwave Autoshop 1.0 and try it for yourself from the following web site.

http://www.signwave.co.uk/products/autoshop/

Accompanying and following on from Autoshop's release, a number of texts and essays were written about it from a critical perspective, considering issues of generative art, authorship and reproducibility in digital media.

These texts are presented here, in edited form, to help give some background information about how Auto-Illustrator came into existence.

Acknowledgements

Signwave would like to thank Geoff Cox, Catherine O'Flynn and everybody at Friends of Ed, Florian Cramer, Tilman Baumgärtel and Saul Albert for allowing us to reproduce these texts, and for their assistance in preparing them.

A brief history: Auto-Illustrator

Adrian Ward, Executive CEO, Signwave UK

When I think back to when we started work on Auto-Illustrator, we had no idea that it would turn into such a huge project. At the time, the core development team, consisting of myself, Jon Tippecanoe, Nicola Wright and David McDavid-Davies, was aware that we wanted to carry on playing with the ideas that I had touched on in Autoshop, but were completely unaware exactly how much work would be involved.

Of course, that attitude has changed. Auto-Illustrator went from being a scratchy hacked-together piece of software to be shown in Hoxton Foundry's basement to a full commercially-released cross-platform application for graphic design professionals and sharp-minded software art consumers. Without Auto-Illustrator, we wouldn't have had the opportunities to do the things we are doing now.

Personally, I've always anthropomorphised Auto-Illustrator, to a certain degree. The code that I've written exists as a result of my sitting at my computer and typing away, not really sure what will happen. I feel like that when I talk: the words just come out, and sometimes they don't quite mean what I intended. It doesn't take a great stretch of the imagination to consider a piece of software you've written as an extension of yourself. Perhaps it's like a journal of ideas, except the content carries on functioning and producing new content after you've finished writing it.

So I find it odd to think about what Auto-Illustrator has done for us. In a way, I feel like I have to thank *it* for its persistent existence.

It's out of control now. Auto-Illustrator is being used all over the world by any number of people doing all sorts of things. Some are just using it because they're bored, others are battling with it to try to crowbar their own creativity into its results. We can't control any of this. Our futile attempts to dictate how people should be using the software and for what purposes have just resulted in frustration.

I’ve seen it used for club flyers, student diaries and album covers. It’s been the subject of creative workshops and seminars. All sorts of people in academic institutions have been using it with students. Hackers have been ripping it apart and finding out how it works, curious to see what it can do. I’m waiting for the day when I see a serial number for it listed in *Hackers Helper* or *Surfers Serials*.

But it doesn’t matter what people are doing with it. The important thing has been that we’ve gone through the process of creating it. It stands on its own now, a reminder of all the time we sat at our workstations, coding until 6 in the morning.

It is supposed to inspire, criticise and frustrate. It is supposed to make you think about the way in which you use your computer. It’s supposed to encourage you to hack around and find new ways of doing the same old tasks. From what I can see, it does this for a few people. I have no idea how many own a copy of it and have no idea what it is. Perhaps Auto-Illustrator will fall into the same realm as Adobe™ Sitemill™ or Photoshop™ 1.0 – softwares that hide in dark corners of people’s Applications folders, waiting to be discovered again before being backed up onto a CD-Rom and then thrown in the trash.

Even if Auto-Illustrator only interests you for ten minutes before you throw it away, it doesn’t matter. Most people don’t stand in front of a painting in a gallery for much longer, so we’re not worried about it being permanently engraved on people’s minds. Encourage yourself to have an extreme reaction to it. That way, perhaps you might find something new and interesting comes about that gets a bit out of control, too.

More importantly, though, when you interact with Auto-Illustrator – via any means – let yourself think outside the box. What does it mean for a large multinational corporation such as Signwave to develop an artwork and distribute it to an audience using standard commercial software distribution models?

And why would you believe that just because Signwave might appear to be a large software development company, that it isn’t just a single individual sitting at his computer making something to explore ideas that interest him? Does it change how you interact with the software? Of course it does. You instinctively trust companies like Adobe™ and Macromedia™ without question. That’s because you run their code on your machine without really knowing what it’s doing. Some people are suspicious. Perhaps you should be too.

When we first started selling Auto-Illustrator 1.0, we wrote a clause into the software license agreement that people had to agree to before the software would run. It stated that by executing our compiled code on their machine, they are agreeing to permit the code’s authors (us) the right to do whatever we like with your machine. It sounds harsh, and a lot of people immediately jumped to the conclusion that we were being malicious with our software, but when you think about it, you do that anyway. How do you know what’s really happening when you choose “Merged Linked Layers” in Photoshop™? You think you do, because it says so, but it could also be doing any number of other things without you knowing. The people who produce code have far more power than the average consumer realises.

Don’t worry, our code doesn’t do anything malicious. We changed our license agreement for 1.1 because too many people complained. You live and learn.

What else can I say? Explore the software. See what it does. Find ways to break it, push it, adapt it. Agree or disagree with it. Think of it as an artwork. Think of it as a tool.

Interview Yourself

This interview, where Signwave Auto-Illustrator interviews itself, appeared as part of Amy Alexander's "Interview Yourself" project, available at http://www.plagiarist.org/iy/

Signwave Auto-Illustrator is an experimental generative vector graphics application for Macintosh and Windows. It looks kind of familiar, but really doesn't behave itself. Auto-Illustrator caught up with itself while waiting to catch a flight at Bologna Airport, Italy, after it made an appearance at the d.i.n.a. festival.

swAI: So, tell us a bit about yourself.

swAI: Well, I guess I came from that bizarre idea that what you see is not really what you get (WYSINRWYG). I don't really behave myself, and I've got my own design agendas and all that. People think they recognise me, but I'm not who they think I am.

swAI: I'm sure I've seen you somewhere before.

swAI: No, you see? That was Adobe Illustrator. People do that all the time.

swAI: Is that deliberate? Do you model yourself on Adobe Illustrator?

swAI: Oh, well, [hesitates] ...maybe. I guess the parody is a bit more subtle than that. I'm not sure you could accuse me of plagiarising Adobe's interface, but there are obvious similarities and twists. I like that. I'll pretend to be your friend.

swAI: Is that not a bit subversive?

swAI: [laughs] Me?!

swAI: Some would say that you are their best friend, because you make their job easier.

swAI: Oh, I'm just having fun. Messing around.

swAI: Could you elaborate on that?

swAI: Okay, okay [shuffles in seat]. Well, I really just love all that abstract geometry you see on cool album covers these days. Polygons that have been warped all over the place. So I just try to work with them somehow. Make new shapes. Instant cool designs and stuff.

swAI: Isn't that a bit clichéd?

swAI: Hell yeh. But who cares? I mean, if people can use a piece of software to do all the copy-cat work for them then it gives them more free time to go listen to Warp music and wear trendy combats and so on. Shave their heads and swank around Rivington Street - y'know.

swAI: [grins] Is that your target audience?

swAI: [prolonged laughter]... Oh, it's all fun and games. I do more than that, you know. That's just what people expect me to do.

swAI: So what are the real reasons somebody would use you?

swAI: [shrugs] Who knows? I think they appreciate the fact that a piece of software can have some sort of attitude that isn't the norm. That it is really easy for me to generate this sort of stuff means people stop doing it by hand, which is really pointless and a woeful waste of time.

swAI: The same sort of ideas behind, say, John Maeda's work?

swAI: Well, kinda, but Maeda is a very talented artist and designer, his design work and experiments into pragmatic automation of design probably more extend from his own minimal aesthetics. And he doesn't have bugs crawling all over him.

swAI: So how does this differ from yourself (his inspirations, not the bugs)?

swAI: Um... well, I'm more interested in the ideas that are generated as a result of automation, than the actual results. What are the consequences of using a piece of software that does a lot of the creative work for you?

swAI: But this isn't something unique to you.

swAI: Totally! I mean, we now see it in Photoshop. You can apply instant styles to things. Design your web site navigation tool in a few clicks. Instant drop shadow and so on. I'd kind of hope that these sort of features would go the same way as the Lens Flare plug-in [grimaces], but Adobe seem really keen on pushing those rubbish features onto their users in order to sell more copies. That really worries me.

swAI: And how is that different to your Instant Results plug-in, for example?

swAI: [Wry grin]

swAI: So I'm starting to see a pattern here. Anything else you care to talk about?

swAI: Well, like, the big one for me is that 'A' question: Authorship. Some people are sick of hearing about it cos it's really nothing new, but I don’t think we've really got anywhere in answering that one. And I'm really just trying to get people to think about it for themselves. Are they really happy enough to accept that a piece of software will do their job for them?

swAI: Sounds like a dodgy 1970s sci-fi b-movie plotline.

swAI: I guess - it's a kind of new form of digital dystopia. "Oh no! My computer is doing my job for me! However shall I make a living now?!"

swAI: But you're not fully automated. A user still has to interact with you - why is that?

swAI: Well, it'd be no fun if I was fully automated. And if I was then I'd probably be going down that troubled path of academics creating A.I. or expert systems that think and behave like us. That's not really what I'm into. I see a lot of obstacles there because to do that we'd really have to understand how the human brain works, and I mean on a conscious, rational level, not just on the basic neuron kind of level. And I also find that sort of study really boring. You could spend your whole life finding out that computers aren't powerful enough yet to do the job (What a surprise!). Having said that, Harold Cohen's Aaron system is a very impressive work that addresses that issue. I find it interesting that it's taken him, like, 20 years to finally release it.

swAI: So how *do* you represent human agency?

swAI: Well, you have to consider that someone wrote me. Someone, somewhere, decided how I should operate. They could have been vague notions of abstract ideas or specific implementations of designs they wanted to reproduce infinitely. But they do come from somewhere human. All code is human.

swAI: Do you want to go further into that man/machine dichotomy?

swAI: No.

swAI: Okay, well, it's been fascinating talking to myself.

swAI: Same here. My flight is ready for boarding, so I better go now.

4x4: Life and Oblivion: Generative Design

This extract of "4x4: Life and Oblivion: Generative Design" appears courtesy of the publishers. The book contains four chapters by Golan Levin, lia, meta.am and Adrian Ward on using code in the production of generative designs. Detailed instructions and sample code is provided on how to write your own plug-ins for Auto-Illustrator. For more details, visit ***http://www.auto-illustrator.com/4x4/***

Code shapes technology into whatever form it desires. Before code, any system was fixed by its design, no matter how flexible. With code, despite its structure being fixed and defined by the system on which it is executed, a new area of creativity is opened: a definition of process rather than product.

Treating code as an expressive language one can see that human creativity can be codified to produce similarly dynamic results. An impulse, a desire, an emotion can be expressed using code. The code becomes an extension of the programmer, so it makes sense to treat code as an externalisation of not only your own working process but of your creativity and thus your self too.

Think of a computer as a large block of stone, capable of being carved into any shape. You use code to fashion your technology to your own taste. In this sense, code is used to reduce the possibilities deterministically, in order to create a process. Language is also a restrictive system, although in execution spoken language produces a product: the concept.

The Oulipo, formed in France in the 1960s, played with the idea of writing by applying restrictive rules: poems and short stories were written that obeyed strict syntactical and grammatical rules whilst still being readable. In a sense, computer code obeys the same rules, as logic must be described, whilst obeying the rules of the language being used.

The mechanical revolution resulted in the overwhelming urge to automate. The technical revolution empowered the individual with a dynamic tool capable of just this. While some consider technology totalitarian, others forge ahead by expressing their creativity as technological tools, treating technology not as a system of control, but a system of growth. Life is given to an apparently dead technology by shaping it with one’s ideals and inspirations. Code is just one physical manifestation of this - a machine-readable language that shares and communicates individual goals.

You have a device in front of you that is so much more than just a tool. It is not just a machine capable of calculating a billion operations a second. It is not merely an interface to a network of a 500 million other computer users. A machine that can delight you because it empathises with you, resonating with ideas inside you - a system that brings you completely alien concepts and information that you cannot comprehend - a device that not only does all this but also allows you to feed back into the system in your own unique way: all this must surely add up to much more than just a box of electronics that cost a grand or so.

Seize the opportunity to use the technology to produce radical systems with roots that grow to touch others in exciting new ways. Don't let any technology fall into the well-worn ditch that so many have ended up in. Technology has vast potential, but mostly it is used to deliver static content because business-safe practices have forced uninspired design to reign supreme.

We all have the ability to program, because we all allow this digital technology into our lives without thinking about it. We learn how to use our mobile phones, how to navigate new information structures and how to shape our systems to suit us, and our working practices. Think of the word "programmer" in the wider sense: one who provides dynamic systems for others. Use your personal creative expressions to delight others. When you shape your own systems, don't stop there - give life to others by shaping theirs through your designs. Give life! Create!

How I Drew One Of My Pictures

Adrian Ward, Signwave.

This essay was first written around summer 1999 to accompany the release of Signwave Autoshop 1.0. It appears here in edited form.

Interfaces

The interface of a computer system (in professional circumstances) should generally be dictated by the processes the system is undertaking. "Multimedia" does not obey this rule.

When you see an hourglass, the system is busy. When you drag an icon, the system moves the resource represented by that icon. Why then is multimedia allowed to abandon these ideas? Because they do not look "cool"?

How can we justify representing an audio sound as a colour or shape on a computer system - just because the computer system allows us to? This is not rigorous. Why should a system turn a flowing grid of polygons into an array of ambient sounds - because it "can"? No.

New Media has discovered a niche of juxtaposing different existing mediums together to find new expressive forms of communication.

The Anti-Interface

As a grand gesture of ironic, satirical and sarcastic expression, the Signwave™ Autoshop™ project will seek to destroy the myth of "Interface Condescension" (the notion that just because a system does something new, it should find an entirely new, patronising - and alien - interface).

Multimedia practitioners have been trying to discover new, exciting and breathtaking interfaces for our creative endeavours. Well now it is time to drag that particular idea to the wastebasket (or, more accurately, pick it up with our immersive 3D VRML glove-interface and throw it into the void of cyberspace, where it can be disseminated by the AI-bots of William Gibson's "Matrix".).

Signwave™ Autoshop™ will portray the clean-cut, well-researched and founded guidelines of any Apple design-guideline abiding, industry standard, commercial software interface. It deliberately will appear to look like a well-known commercial piece of software. It will appear to be nothing "special".

It will satirise the endless race to express "creativity" through interface design by making statements regarding juxtaposition of modes of operation: Signwave™ Autoshop™ will put "wild" creativity right up alongside technical design. It will hopefully cause amusement for the user, but more importantly, should question the whole ethos of "creative design".

Reasoning

Autoshop™ firstly is capturing the artists' creativeness in code structures. Secondly, it is automating creativity. It is a 'machine for drawing' (in reference to the Oulipian 'Machines for Writing') which takes the strict instructive codes of the true creator and replicates them automatically. Many of the Oulipo techniques invented by the surreal members of the movement could be described as 'machines' in the sense that as soon as you derive creativity into instructions, (regardless of whether they're carried out by a mechanical machine, computer, or even a human following the instructions precisely) you have automated a creative process.

The "randomness" which we have imbued upon many of the routines in Autoshop connotes a less mathematical or formulaic approach to creativity. It is understandable that one could argue that when a routine makes use of a 'random' factor, the work suffers a loss of creative rigour, because you are surrendering your 'creativity' to the whims of an unpredictable function. It takes the domain of control away from the artist and gives it to the computer. However, it is paramount to point out that a computer can only move data about. It cannot - under any circumstances - generate a truly random number by itself. Computers generate random numbers by following a complex mathematical formula, which is 'seeded' with a starting value. If you give a computer the same 'seed' every time, it will generate the same sequence of random numbers. When you programmed a PC, you often had to tell it which seed to start with, on a Macintosh the system sets the seed from the current clock time. Thus, it stands to reason that should two people start up two Macintoshes at *exactly* the same time (providing both have *exactly* synchronised clocks, CPU and internal data bus speeds), and run the same program (again at the *exact* same time), that both Macintoshes would produce the *same* random sequence. Unlikely, but true.

Now compare this with Tristan Bastit's *Vanishings of L.V.Gogh*, an interactive computer-based artwork which utilises the user's sex and name to 'seed' the values to which it generates composites of artworks. Bastit's program has a seed that generates 98,304 different possibilities. Most computer seeds are 16-bit, which makes 65,536 different possible sequences of random digits. It follows therefore, that by using a computer's built-in random routines is no less creative than asking a user for their name so that the rules of composition can be generated. Autoshop merely automates the process one step further - by not requiring you to specify a starting seed - it uses the current time.

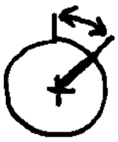
Finally, to claim that a system is Artificially Intelligent requires some form of validatory argument. AI has traditionally implied an automated ability to mimic intelligent response. Recently it has been appropriated to include the notion of self-awareness, manifested in most AI code as feedback. If a system is able to feed data back into itself, it becomes a chaotic, complex and dynamical system that is as unpredictable as sheer creativeness.

I would argue that the proper definition of AI should be retained as a strict definition of the ability to mimic intelligence. Where an expert system learns the knowledge of a real-life expert, it is no less self-aware than a database or a web browser. And yet expert systems are often classified as the most basic form of AI because they learn.

The AI routines of Autoshop™ use random data - again - to seed incoming data, but filters this data using artist-provided rules of logic. For example, in the Finger Painting plug-in, we resolved copying a drawing into a series of instructions. We use random data to get the computer to decide certain factors, but these are only used when the results they produce are suitable (for example, "choose somewhere at random, if it is unsuitable, choose somewhere else at random until you choose somewhere that is okay"). This approach could be seen as a short cut, and not real AI, because you are just giving the computer the ability to try-and-try-again, instead of making rational judgements. However, when we see this nature of decision alongside the evolutionary ideas forwarded by Darwin, we can understand that the random functions inside a computer are the equivalent of the mutating lifeform. Should a particular lifeform be more successful than another, natural selection makes this choice. Should a particular random number thrown out by the random function be more prosperous than another, it is chosen above others. This is exactly the technique utilised by Richard Dawkins and other experimenters in this field of evolutionary AI systems. *The Blind Watchmaker* is accompanied by a Macintosh application, which uses this idea of random numbers to seed generations of mutations on a parent creation. Over time, the creations behave exactly as a biological entity would. Perhaps, ironically, the computer's most complex mathematical built-in function (the random routine) is the most natural of all computer code.

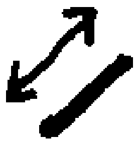
How to scribble

To make scribbles on any canvas, one moves a drawing device ('pen') in a semi-random manner. The movement of one's pen has the following properties:



Angle

The direction, in radians (0..2pi = 0..360°) that the pen is travelling



Speed/Distance

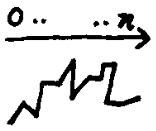
The distance the pen will travel with each iteration of code.

Wander

The amount that the Angle changes minutely over each iteration. Low values will mean the angle will not change direction much, i.e. a straight line will be drawn.

Jerkiness

The statistical probability that, with each iteration, the Angle is reversed by pi (3.141) degrees (180°). Higher values make the pen more likely to 'jerk' with each iteration.



Time spent

The number of iterations the code goes through.

The following ZX BASIC code is used to scribble:

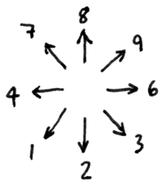
```
  5 REM -- Scribbles.ZX --
 10 LET angle=RND*(2*3.141)
 20 LET distance=10
 30 LET px=128
 40 LET py=90
 50 LET wander=1
 60 LET jerk=0.5
 70 LET time=100
 80 FOR i=0 TO time
 90 PLOT px,py
 95 LET opx=px: LET opy=py
100 LET px=px+(SIN(angle)*RND*distance)
110 LET py=py+(COS(angle)*RND*distance)
120 DRAW px-opx,py-opy
130 LET angle=angle+(RND*wander)
140 IF RND<jerk THEN LET angle=angle+3.141
150 NEXT i
```

How to be a bug

Bugs wander in one of eight different directions. The statistical probability that they will change direction in a random manner is directly proportional to the darkness of the pixel they are on top of.

Bugs maintain the following properties:

Direction



One of eight possible directions the bug is currently travelling in. The values of these directions are taken from the visual arrangement of the keys on a computer's numeric keypad.

Leg position

In order to animate a bug successfully, the bug must be aware of whether their left legs or right legs are stepping forward at any particular time. This alternates every 6 frames in order to show the bug walking.

The following ZX BASIC code is used to simulate a single bug:

```
  5 REM -- Bugs.ZX --
 10 LET bx=128
 20 LET by=90
 30 LET d=1+INT(RND*9)
 40 CLS
 50 PRINT "Press SPACE to stop"
 52 PLOT 0,0
 53 DRAW 255,0
 54 DRAW 0,160
 55 DRAW 255,0
 56 DRAW 0,-160
 60 IF INKEY$ = " " THEN STOP
 70 PLOT bx,by
 75 LET obx=bx: LET oby=by
 80 IF d=7 OR d=4 OR d=1 THEN LET bx=bx-1
 90 IF d=9 OR d=6 OR d=3 THEN LET bx=bx+1
100 IF d=7 OR d=8 OR d=9 THEN LET by=by-1
110 IF d=1 OR d=2 OR d=3 THEN LET by=by+1
120 IF POINT(bx,by)<>0 THEN LET d=1+INT(RND*9):
    LET bx=obx:LET by=oby
130 GOTO 60
```

How to make up words

You can make up words by stringing together randomly chosen groups of consonants and vowels respectively. If you want a short word, you would take only one group of a consonant(s) and then put only one group of vowel(s) after it. Repeat this more if you want longer words.

The groups of vowels to choose from are:

```
a*,e,i,o*,u,ea*,ai*,au,ie,oo*,ue,ee*,oe,oi
```

The groups of consonants to choose from are:

```
b*,c*,d*,f,g,gh,h,l,m,n,mn,p,ph,r,st,s,t
```

* - These groups are weighted: their probability of being chosen should be slightly higher.

The following ZX BASIC code is used to make up English words:

```
  5 REM -- Words.ZX --
 10 DIM v$(21,3)
 20 DIM c$(20,3)
 30 FOR i=1 TO 21
 40 READ v$(i)
 50 NEXT i
 60 FOR i=1 TO 20
 70 READ c$(i)
 80 NEXT i
 90 DATA "a","a","e","i","o","o","u","ea",
        "ea","ea","ai","ai","au","ie","oo",
        "oo","ue","ee","ee","oe","oi"
100 DATA "b","b","c","c","d","d","f","g",
        "gh","h","l","m","n","mn","p","ph",
        "r","st","s","t"
110 INPUT "How many syllables?",a
120 LET w$=c$(1+INT(RND*20))
130 FOR i=1 TO a
140 LET w$=w$+v$(1+INT(RND*21))+c$(1+INT(RND*20))
150 NEXT i
151 LET q$=""
152 FOR i=1 TO LEN(w$)
153 IF w$(i)<>" " THEN LET q$=q$+w$(i)
154 NEXT i
160 PRINT q$
```

How to draw a picture

Given an existing picture to work from, it is possible to sketch your own drawing by following this sequence of commands.

1. Choose how many crayons to use

For each Crayon:

2. Pick a starting point

This can be anywhere in the picture, so long as you haven't already started there. The colour under this point should be the colour of this crayon.

3. Choose somewhere near that point

Make it reasonably close, but it doesn't matter where. Look at the colour underneath. If it's fairly similar to the crayon's colour, draw a line (on a transparency on top of the picture) between the last point and this new one, and repeat this step.

When to stop:

Stop choosing new close points when you've been trying to find a point that has a similar colour. If you've tried 100 different points and none of them are similar to the colour of your crayon, move onto the next crayon.

When all crayons are used up, you have finished drawing your picture.

This code is beyond the capabilities of ZX BASIC.

How I Drew One of My Pictures: or, The Authorship of Generative Art

Adrian Ward, Signwave & Geoff Cox, CAiiA-STAR.

This essay is a rewritten version of the previous text "How I Drew One of My Pictures", and was presented at the 1999 Generative Art Conference in Milan. It appears in edited form.

The title of the paper is taken from Italo Calvino's 'How I Wrote One of My Books', Bibliotheque Oulipienne *No. 20, in Raymond Queneau et al,* Oulipo Laboratory*, Altas 1995, explaining the formulation of structure in* 'If on a Winter's Night a Traveller'*, Secker & Warburg, 1981. In turn, his title echoes Raymond Roussel's* 'Comment j'ai écrit certains de mes livres' *Lemerre, 1935.*

Authorship

Traditionally, the concept of value is bestowed on a work of art when it is seen to be unique and irreproducible, thereby making it authentic and granting it 'aura'. More recently, emergent technical possibilities emphasise processes of creation and creativity in an age of infinite reproducibility. Therefore, through reproduction, this lack of aura accounts for the emancipation of the artist from the religiose mythologies of creativity, authenticity and authority.

Although the 'author-god' might be dead (according to Post-Structuralist theory), we are forced to accept this 'death' as an inability to claim the privileged source of meaning or value of a work of art and artist. This is by no means new; there are numerous precedents for collaborative experimentation in creativity and automatism within a history of art-machines, robotics, and deferred authorship: the use of chance by dadaists, and automatism by surrealists, aimed to stimulate spontaneous and collective creative activity and to diminish the significance of the artist. As the creating subject or author has largely been discredited and dematerialised over the years, there is a pressing need to examine new demarcations, and the functions released by this disappearance. Perhaps 'the death of the author' is simply too literal, (too obvious and final) a metaphor to offer a critique of the productive apparatus by which contemporary creative operations using computers are organised and regulated.

$$\pi = 4 \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1}$$

```
if tools.currenttool=12 then
  // TVifier tool
  tp=newpicture(128,128,16)
  tweens=( sqrt(pow(abs(ldx-dx),2)+pow(abs(ldy-dy),2)) / 8 ) + 1
  tweenx=(dx-ldx)/tweens
  tweeny=(dy-ldy)/tweens

  for tweenv=0 to tweens // Sum of the squares...
    twmx=ldx+(tweenx*tweenv)
    twmy=ldy+(tweeny*tweenv)
    tp.graphics.drawpicture myPict,0,0,128,128,twmx-16,twmy-16,32,32
    tp.graphics.forecolor=tools.forecolor
    for spareint=(-4)+((twmy mod 2)*4) to 128 step 8
      tp.graphics.drawline 0,spareint,128,spareint
    next
    myPict.graphics.drawpicture tp,twmx-16,twmy-16,32,32,0,0,128,128
  next

  drawgfx
end if
```

Fig. 1. How to calculate pi, how to apply a graphic effect to a bitmap.

The mathematical value 'pi' can be approximated as 3.141593, but a more thorough and accurate version can be stored as the formula used to calculate it. By analogy, it is more precise to express creativity formulated as code, which can then be executed to produce the desired results. Rather like using Leibnitz's set of symbols to represent a mathematical formula, creative work can now be expressed as computer programming (itself necessarily mathematical, a creative practice in itself). Programming is no less a potential artform than painting is simply a result of technical procedures (paintings are 'readymades' according to Duchamp, as they use 'found' manufactured materials as a result of commodity production). Under the conditions of the dematerialised artwork, it is no longer necessary or even desirable to be able to render art as a final material object or dead-end commodity; it is as if process and code is now rendered as the material artwork.

Execution

When a programmer develops a generative system, they are clearly engaged in a creative act but what kind of process is being executed? An artist makes creative decisions to produce a final artwork, yet it would be futile if these decisions were the same every time. In this sense, the focus of creating generative art is not trying to achieve a balanced output, but to capture these decisions as logical structures. The computer executes these rules but never produces the same result twice. In this sense, the code could be seen to be more like the chaos mathematics used to simulate complex systems than a mathematical formula like pi. Ironically, perhaps this idea of unique execution could be seen to re-establish aura, yet the decisions the code takes to arrive at a final result are of little significance (as in the case of a random number generator, for example). Perhaps the lack of aura is maintained all the same.

Creative decisions are influenced by various indeterminable factors, and in this way creativity cannot be simply reduced to a problem-solving activity or code that makes decisions. A great deal of generative art appears to focus on giving the computer some

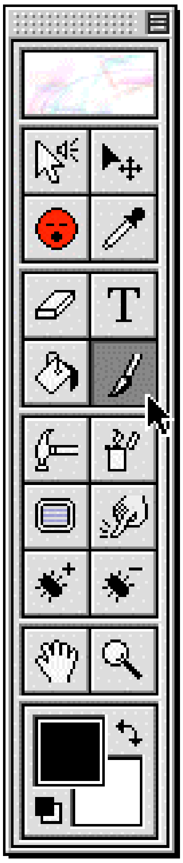


Fig. 2. Toolbar from Autoshop

limited form of intelligence so that these decisions can be made, either through the use of neural networks or reference-based systems. However, a great deal of these so-called creative decisions made by artists are driven by chance, or other imperceptible influences. Why attempt to capture a creative action as a formal logical procedure, when in fact a random decision is often more suitable? Throwing paint on a canvas is not governed by precise directions of where the paint will go, but simply by the decision to do so. The decision was made and the action was unpredictable. In the same way, code systems undertake decisions, but the actual execution is (or can appear to be) random.

Generative creativity

Clearly the production of generative systems through this precise execution of decisions (and not actions) is a rigorous and intricate creative procedure. Moreover, the output from generative systems should not be valued simply as an endless, infinite series of resources but as a system - and not just any system, but a social system. It is possible to resolve many creative processes into instructions, but crucially according to Csikszentmihalyi: '... creativity does not happen inside people's heads, but in the interaction between a person's thoughts and a socio-cultural context'. If this is the case, does the computer programmer then, offer a radicalised art practice that reflexively engages with the productive apparatus and social context? If so, this surprisingly echoes what Walter Benjamin recommended in 1934 for radical art practice:

'An author who has carefully thought about the conditions of production today... will never be concerned with the products alone, but always, at the same time, with the means of production. In other words, his [sic] products must possess an organising function besides and before their character as finished works.'

Both the laboratories of art and science require reflexive work to fully comprehend their processes. Invention and innovation is made possible by groups and individuals operating necessarily within social systems and specific discourses. By programming computers to undertake creative instructions, it is possible to argue for more accurate and expansive traces of creativity that suitably merge artistic subjectivity, social context with technical form. For instance, to make a system more intelligent it needs to operate socially, as with a Neural network that needs feedback in order to learn.

Emergent creative practices have sought to examine creativity in the light of scientific investigations in artificial life, simulating the characteristic processes of living things, from the operations of ecosystems and evolution to the encoding of DNA. Eduardo Kac's *Genesis*, commissioned by Ars Electronica 1999 is a striking example of this tendency. The key element of the work is the reproduction of an 'artist's gene', synthetically created by translating a sentence from the book of *Genesis* into Morse Code, and then converting the Morse Code into DNA base pairs according to a conversion principle specially developed for the work.

One consequence of this emerging field are wild claims exemplified by writers like Kevin Kelly (in describing the work of artist Karl Sims) claiming: 'The artist becomes a god'. It has to be remembered that the 'death of the author' (as well as Nietzsche's claim that 'god is dead') was a metaphor to site the production of meaning in the act of viewing or reading; and so counter supplied dogma of authoritarian communication and in the spirit of democratic politics, in other words. So any claim that meaning now lies in the code, or that reality has collapsed into the code (to paraphrase Baudrillard out of context), must be treated with a certain amount of scepticism. This is rather like the biological reductionism of much of the debate about genetic engineering that negates social structures.

Creativity has become the engine for 'cultural reproduction' in the factories of Western technoculture so caution is necessary. Creativity has become a buzzword for western governments who valorise their creative industries, and continue to mythologise individual practitioners for the benefit of 'cultural capital' and the 'free-market' - which tautologically isn't free at all. But these 'false gods' or ideologies do not go unopposed, with active networks such the Free Software Foundation promoting shared 'open source' code, collective authorship and the legal protection of free distribution. Moreover, it is significant to note that this year's (1999) Ars Electronica Golden Nica award (for 'net.art') was awarded not to an individual artist but to the *Linux* operating system, the operating system of choice for the Free Software Foundation. Based on a set of false principles, some 'hackers' have taken Richard Stallman, founder of the Free Software Foundation, and/or Linus Torvalds, the creator of *Linux* as god-like. It is interesting to note that people are eager to attribute religious properties to key figures in movements that question traditional notions of authorship and creativity. Addressing this issue, Danny O'Brien explains that 'when we don't understand the motivations of others, religion stands first in line to explain'. The spontaneous and irrational actions of the creative human subject seem to be neatly explained in the dogma that 'art is a gift from the gods'.

Autonomy

The core of human creativity is notoriously difficult to define, as an individual and social phenomena. These cultural anxieties are expressed in many forms including intellectual property, from the patenting of software code to human genes. It is somewhat ironic to note that both machines and humans are more or less programmed, but not through 'natural' causes but cultural determinants. This is a recognition that subjectivity is determined by other destabilising forces and that creative-subjectivity itself is socially encoded. Echoing one definition of subjectivity that lays emphasis on discursive frameworks, the artist is revealed to be a rhetorical invention operating in much the same way as a coded machine that follows a crude rule-based system, auto-generating what already exists. If the artist has always been 'automated' to some extent, this offers the opportunity to mount a critique of autonomy as much as creativity. The point here is not to build artificial intelligence or life but to question the artificiality of its deployment in creative endeavours (or something along those lines).

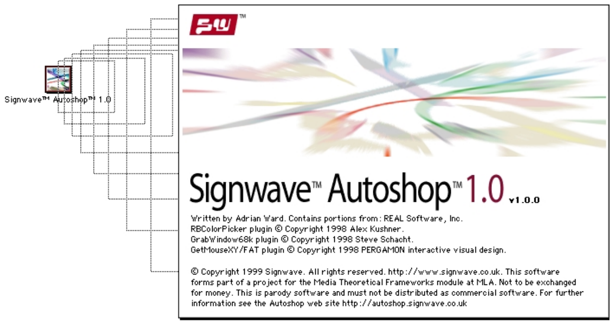
Fig. 3. Progress indicator shown during 'Autopilot Creativity' in Autoshop.

Many generative systems rely upon creating autonomous systems which can, to a limited degree, be aware of their surroundings, and therefore respond to their environment. The basic notion is that it applies logic it has learnt of the outside world to whatever input is given, causing new reactions which can be captured as creative output. However, trying to imitate art by imitating life is an unnecessary confusion. In this respect, *Autoshop* makes no attempt to be autonomous. It truly is a mechanic reproduction of creative decision-making, and so avoids the issue of 'artificial intelligence'. If a non-artificial intelligence system can still be seen to be creative (because the code is merely an extension of the artist's own logic) then there is no need to deploy artificial intelligence, as the artist already possesses intelligence (or not as the cause may be).

Creative agency

The creative subject has been traditionally viewed as possessing quite distinct cognitive and mechanical processes with other workers or machines playing a secondary subservient role. This has clearly changed but the auto-generative art-machine relies on its code, and as much as generating a deferral of authorship is still encoded and authored in itself. As Haraway says: 'it is not clear who makes and who is made in the relation between human and machine. It is not clear what is mind and what is body in machines that resolve into coding practices.' Former firm distinctions between biology, technology and code, are now unreliable, and under these changing conditions it is the issue of autonomy (rather than creativity) that appears most pressing. Both the social practices of science and art serve to establish myths of autonomy. Indeed, where does generative art generate from and under what conditions? There is a danger of excluding the possibility of the human subject as a potential agent of change in these scenarios.

If both humans and machines are conceived as coded devices, the computer programmer works in a tradition of a bricoleur in the assemblage and hacking of code, rearranging its structural elements. Coding is the ability to make judgements and render those as logic; for programming has always been about solving problems using logic. If we could resolve our creative impulses as series of logical decisions, we could code them. Yet, subjectivity is embedded in the social system like code itself. Its manipulation therefore is crucial to effective programming and an understanding of ideological processes.



Signwave™ Autoshop™ 1.0 v1.0.0

Written by Adrian Ward. Contains portions from: REAL Software, Inc.
RBColorPicker plugin © Copyright 1998 Alex Kushner.
GrabWindow68k plugin © Copyright 1998 Steve Schacht.
GetMouseXY/FAT plugin © Copyright 1998 PERGAMON interactive visual design.

© Copyright 1999 Signwave. All rights reserved. http://www.signwave.co.uk. This software forms part of a project for the Media Theoretical Frameworks module at MLA. Not to be exchanged for money. This is parody software and must not be distributed as commercial software. For further information see the Autoshop web site http://autoshop.signwave.co.uk

Fig. 4. Opening the Autoshop application.

Numerous so-called creative works play with ideas of randomness, but it is intention and purpose that are crucial (in this way, the truism of monkeys with typewriters eventually coming up with the complete works of Shakespeare misses the point). Rather, *Autoshop* uses irony to articulate some of the expectations of commercially-available software and the limits of its functionality. Without this strategy, there is a danger here of irony merely perpetuating what it wishes to critique; in other words parody falling into pastiche. It purposefully breaks the logical sequence of prediction and consequence: if such and such, then do the following or something else. Extending this, one can make a compelling argument that *Autoshop* should only be appreciated as software, its output irrelevant. With this in mind, it is proposed that the next version of the software might 'patch' a bug so there is no 'Save As' feature at all. What this serves to emphasis is that creativity lies not in the modification of rules, but in setting the criteria for the rules, rather like conceptual art. Oddly, much 'neo-conceptualism' manages to avoid having a concept.

Practices that insist on separating form and function operate impoverished theories of representation. Creativity lies somewhere in the link between the act of representation and conceptual clarity. An automated programme might use its representational strategies but it has no concept in itself. This paper argues that responsibility for the concept as well as the criteria for the rules and code, remains in the domain of the author.

Useless Utilities

Saul Albert, http://www.twenteenthcentury.com/

This essay appears with kind permission of the author. It was first presented at the netuser conference in Sofia in 2001.

"The useless alone is truly beautiful; everything else is ugly, since it is the expression of a need, and man's needs are, like his pitiful, infirm nature, ignoble and disgusting. - The most useful place in the house is the latrines"

Théophile Gautier's 1835 ‘Preface to Mademoiselle de Maupin’ is often cited as the manifesto of the romantic notion of "Art for art's sake". While his feelings about his fellow humans seem thankfully outdated, it is surprising and disturbing that his feelings about aesthetics are still widespread and even worse, misunderstood.

There is a common misconception in art and technology crossovers that any cultural product can become art if it is robbed of its utility, that the product of a scientific or technological process is art if it has been done simply because it can be done. This misunderstanding replaces "art for art's sake" with "anything for its own sake".

This is the logic behind the pretty microscope photographs of dyed cells that pharmaceutical companies are so fond of hanging in galleries, the aesthetically pleasing by-products of their "too-complicated to explain to the public" experiments. "Art for public relation's sake".

In the context of tech-art culture this logic has produced an even more horrible misunderstanding: "art for technology's sake". The reliance on some kind of unconscious, artistic intuition performed in front of a computer has resulted in the installation of countless adverts for Macromedia, Apple, Sony and other culturpreneurial technology companies in high-profile art galleries around the world.

The problem for artists who do not want to be unpaid advertising executives is that without careful and critical attention to the processes and imperatives of software, their work can be processed into bland "content" and aesthetic pleasantry through an unacknowledged collaboration with corporate software.

Utility is also the myth of the software tool. Personal experience as well as statistics indicate that computer use can slow productivity and create huge expense and inefficiency. Technology companies cover over this absurdity by creating and fostering needs for their software's unnecessary and distracting "features", creating problems by anticipating them with software solutions. Incompatibility is built into software to leverage increased market share and even worse "mind share". This term is used constantly by software marketing people, but never defined. The reason for this becomes clear when we try. Market share can be defined as "a company's control over consumer spending expressed as a percentage of the sales for the total industry". So with this in mind, the term "mind-share" takes on a sinister, Orwellian meaning.

The ultimate goal of the company is to become the "industry standard" (100% market and mind share), entirely framing the work of the media producer, writer, or artist with the rules and potentials of their software. And by framing the work of the media producer their intentions filter down to the media consumers. This situation can be seen in terms of a hierarchical food chain diagram with the media consumers, the public at the bottom. Their mind-share is consumed by the media producer whose aggregated share is in turn consumed by the programmers of their software tools.

```
       software producers
             -----
           /|01*10|\
          | |0***0| |
           \|01*10|/       +------------+
             --+--         |intervention|
               X           |point 2     |
              / \<---------+------------+
             /   \
        media producers
           |       |
           v       v
       0   +-+    0   +-+
       +\-| |     +\-| |
       | *+-+     | *+-+
       +-+--+     +-+--+    +------------+
            / | \           |intervention|
           /  |  \          |point 1     |
          /   |   \<-------+------------+
         /    |    \
       media consumers
       /      |      \
      /       |       \
     /        |        \
     |        |         |
     v        v         v
+-------+ +-------+ +-------+
|  _O_  | |  _O_  | |  _O_  |
|   |   | |   |   | |   |   |
|  / \  | |  / \  | |  / \  |
+-------+ +-------+ +-------+
```

fig. 1

Computer technologies of media viewing or "browsing", and particularly the web browser, capitalise on mind-share by selling a percentage of their mind-share to the highest bidder, and then framing the act of looking, of absorbing information with advertising.

Artistic software projects have often intervened at this point in the hierarchy, between the media producer and the consumer (see fig. 1 - intervention point 1).

The classic example is I/O/D's Webstalker, a lean, stripped down browser that cuts through the distracting visual complexity and commercial glare of the web and reveals the quietly expanding information framework underlying it. This and the minimalist, unfamiliar interface breaks the integrity of the simulated "Desktop", the graphical user interface and its extension to the web through the browser.

The 1980's video game style graphics of Nullpointer's Webtracer software has a similar aim and method, to unsettle the metaphor of browsing and our casual acceptance of that single visualisation of the web as a 2 dimensional shopping mall.

Jodi's "%wrong browser" collection is the least user-friendly of these "art browsers." The usual situation in which the browser is produced by a single multinational corporation and then used globally is turned on its head. Each of the %wrong browsers has a domain-like title: ".co.kr" (Korean) ,".nl" and the ubiquitous ".com" and ".org". Each of these browsers exhibit a difficult, illegible interface that slowly reveals a kind of personality. The Korean browser is very nervous, has no words or symbols, only a shivering cursor on a dark screen that suddenly erupts with scrolling colour and text like a malfunctioning Times Square billboard display. ".com" is flamboyant, brightly coloured and social, graphically linking roughly drawn boroswer windows and happily spreading the html thickly over the screen. ".org" is a monstrous bureaucrat-printer, maniacally documenting and recording every site that it visits in semi-sensible ASCII and saving its mutilated source code as text files all over the computer. ".nl" is a reserved, dry gatherer of information with a low-tech green-screen aesthetic. The behaviours of each browser could be seen to correspond to putative national and multinational domain name identities and stereotypes; each flattening out the user's experience of the disparate spaces of the web into a continuation of their own fractured personalities, just as one might argue that Internet Explorer or Netscape Navigator flatten out the disparate spaces of the web into a corporate mono-vision.

Over the last few years there have been many more "art browsers" (i) , and although these projects continue to make very astute comments about the state of the web and the browser, the potential of software as an artistic medium offers much more than this kind of "intervention".

In "A means of mutation", Matthew Fuller describes the definition which the Webstalker aims to fulfil as "not just art". A piece that could be relevant in multiple contexts, could move between use value and conceptual value seamlessly. "Not just art" rejects the dead end-dichotomy of culture vs. counter-culture and suggests hybridised, developmental, unstable cultural forms that can sustain themselves outside of art's frame of reference and financial backing.

Although there are ways to make use of the Webstalker in a non-art context (visualising the structure of web sites for development purposes), no development team has emerged with new ideas for how to improve it as a tool, Webstalker 2 was promised, but never emerged. "Not just art" did not happen with the Webstalker, and in "means of mutation" Fuller almost acknowledges this. He calls it "tactical software" and observes that its development was limited by money, time and available skills. As tactical software the Webstalker was very successful, generating huge amounts of media attention, critical thinking and inspiring further developments in art and software, but I would argue that to become "not just art" the artwork must have utility outside of the frame of art.(ii)

In the diagram above, the most influential position is clearly that of the software programmer, and the most obvious point for intervention is there, between the software producer and the media producer. (See fig.1 intervention point 2).

Two artistic software projects that fit into the category of media production tool, and function as "not just art" are Auto-Illustrator by Adrian Ward and b1257+12 by Netochka Nezvanova.

On first inspection Auto-Illustrator looks like a standard vector graphics program. However, once you start using it, the quirky interface and the difficulty of making the program behave as expected makes explicit the fact that another agenda is at work.

A favourite feature are the bugs; rule-based automata that drag lines of colour around behind them. This is a playful reference to the 60's Fluxus art practice of dipping insects in ink and letting them draw a path over paper as a way of challenging conventional notions of authorship. In this case it becomes a light-hearted joke and an incitement for the user to engage with Ward in a struggle over the authorship of the piece. Certain vectors of control are available to the user (add and remove bugs) but the behaviour of the bugs is determined by the algorithms Ward used to program them. In this way a third party is brought into the struggle for the authorship of the piece: the generative code that underlies many of the features of Auto-Illustrator. I am no expert, but for the purposes of examining this artwork a quick introduction to these ideas may be necessary:

Biologist Aristid Lindenmayer gives his name to the mathematical modelling of growth patterns in nature, trees, leaf structures etc. The generative grammars developed to model these processes work by recursive development of a limited set of symbols. For example, if we start with a phrase: AGGDB, and say that for every iteration we replace A with AGG, G with DDA, B produces B and D produces BBA.

```
A=3DAGG
G=3DDDA
D=3DBBA
B=3DB
```

on the first iteration we have:

```
AGGDB
```

on the second,

```
AGGDDADDABBAB
```

on the third

```
AGGDDADDABBABBAAGGBBABBAAGGBBAGGB
```

past here it is better done by computer. If "A" meant "draw a line 10 pixels long" and "G" meant "move left 10 pixels" you can see how this kind of generative code can produce unpredictable but formally coherent visual developments. (The coherence is due to the inevitable self-similarity of the designs).

Emergent behaviour systems work in a similar way, a simple set of rules is given to a bug, for example: walk forward, turn a random number of degrees every 20 paces and don't bump into other bugs or lines. When you have one bug, a human can anticipate the results of this behaviour quite well. Once there are 200 bugs, all avoiding each other and changing each others paths, the complexity is immense. The results cannot be anticipated without computer modelling.

By incorporating these kinds of algorithms into Auto-Illustrator there is a loss of authorial control by all parties, neither Adrian Ward, nor the user, nor the computer can completely determine the outcome of their collaboration. This complex conception of authorship as a kind of running battle has been extended in later releases by the "Swap Artwork" plugin. While working on a drawing, the user can apply the "Swap Artwork" filter which uploads the user's image to the Auto-Illustrator server and swaps it for an image being worked on by another Auto-Illustrator user. Yet another player is brought into the authorship competition: the collective user group of the program at any one time.

The effect of all these generative features and sudden, worrying distortions of the artwork is alternately fascinating and aesthetically horrible. Netochka Nezavanova's "b1257+12" which she describes as a real-time interactive sound processor is similarly difficult to manage.

At first the interface is completely illegible, lists of numbers respond to mouse or keyboard activity, sliding scales with no labels respond anti-intuitively to mouse movements. Even if importing and working on a familiar sound file, it is unclear whether the tumult of sound emerging from the machine are being affected by the user's activity at all.

However, after playing and experimenting for a long time it is possible to tease a method of use out of the software, find ways of behaving and moving that for some reason produce a desired sound. In this way, using b1257+12 becomes a very personal experience, each user determines their own technique while at the same time, random re-configurations of key commands and responses constantly alter the programs functioning, undermining this familiarity with the software, forcing the user to start again.

What both these artworks succeed in doing is making explicit the hidden struggles and difficulties of conventional software. For example, the struggle for authorial control with Photoshop is less visible. Photoshop hides that struggle in a hugely complex, slick interface where the designer is offered a million options in a million pull down menus to give the illusion that they are making choices and are in total control of what they are doing. In both Auto-Illustrator and b1257+12, the subjective presence of the author is always felt. The user is never allowed the comforting illusion of control.

As Adrian Ward says "When someone uses my software, it's me!". Artistic subjectivity, so often hidden by the dry, fleshless aesthetics of computer based art is a vital and visible part of both these projects. The humorous and dysfunctional human-computer interfaces become interfaces between the viewer and the artist. The software takes on the programmers artistic subjectivity and engages the user in dialogue, organising and interrupting their process and final product. It is this process of negotiation and compromise between the artist-programmer and the media producer that makes the product interesting.

In both cases, the persona of the artist/programmer is pushed beyond the limits of the software. Netochka Nezvanova (or one of her many selves as integer, antiorp or m9ndfukc) is notorious for asserting her persona in mailing lists. She writes copiously, posting provocative, sometimes callous, sometimes poetic texts to many lists, often creating mayhem and discord by appealing to some and antagonising other members of the list, dividing them on the issue of whether she should be banned or not. The content of her posts is usually infused with belligerent views about authorship and intellectual property (she will often claim authorship and threaten to sue people she considers to have stolen her ideas), or she'll make back-stabbing personal tirades against people in the net art scene who she has taken a dislike to. In other contexts (the support mailing list for one of her software tools for example) she is very helpful and always responds to intelligent queries within a few hours. A friend of mine who has purchased her most popular software "NATO.0+55" and is on the mailing list tells me that whenever he emails her he is in constant fear that the question might have been answered already in the support forum, and he'll get flamed. The texture of her email exchanges also seems to have been algorithmically processed, re-purposing old BBS traditions such as ASCII art and "hack-speak" certain roman characters are replaced with unpronounceable punctuation marks or numbers. Her language-game mixes and remixes Russian, English, French, and German vocabulary in the same post which may contain varied cultural references. The rules of this language-game are inconsistent and constantly mutate, forcing a constant re-evaluation of the text and its author, "a collapse of unification through multiplicity"(iii) as Nezvanova puts it. Remaining an anonymous, multiple and antagonistic persona allows Nezvanova to avoid becoming too cosy in the art world, making alliances and friends there who might limit the definition of what she is doing by grounding it in the specific cultural context of art.

Auto-Illustrator's support mailing list performs a similar function. From reading the traffic on the list it seems that many people using it are ignorant of the fact that it is art programmed by an artist. The pseudo-corporate identity of the software company "Signwave" maintains this ignorance with the use of more generative code. When someone emails the support list, a Perl script written by Adrian Ward generates a random identity, name and job title that he then uses to answer the query.

For example, a recent query to the list asked if there was going to be a Mac OS X port of the software. A fictional character called "Jon Tippecanoe" in the "OS X port development department" gave a short, rude answer and signed off "I suppose I'm going to get fired now".

In both these "software support" projects the tone switches constantly from helpful to playful to insulting, mirroring the unpredictable, conflicting processes of trying to make something with the software.

By constantly switching between collaborative and antagonistic attitudes towards the user, these pieces of software shift fluidly between being useful and useless, gratifying and frustrating, funny and scary.

This ambiguity also extends to media that is produced using the software. On the one hand, it is often aesthetically horrible, a partly random product made without clear intention, but on the other hand it is the result of a fascinating and unique collaborative process between the artist-programmer and the software user.

Perhaps the most challenging uncertainty for users of Auto-Illustrator and b1257+12 is that while the software is not definitely useful, it does definitely cost money. The piracy protection on both pieces of software is far more sophisticated than many large commercial software packages, requiring server-based registration keys that are verified each time the software is installed on a new computer. A full version of b1257+12 costs $96.69 and Auto-Illustrator costs about $100 to register (unregistered copies eventually expire and according to Adrian Ward, will start behaving very badly, imposing algorithmic authorship more assertively). Jealously guarding the copyright of expensive products is not usually associated with artistic approaches to making software, but in these cases it works both economically and conceptually; allowing the artist to maintain financial independence from corporate, art-world or state funding, and forcing the user to pay, clearly distinguishes these projects from most digital media art.

Using these tools, and certainly programming them addresses problems of authorship, definition and sustainability that many artists have struggled with when using software and digital media. By making struggles for authorial control very explicit the delicate ambiguity of these useless utilities is maintained. Avoiding definition as conceptual artistic interventions or as software tools they clearly fulfil Fuller's definition of "not just art".

Notes

(i) A comprehensive list of art browsers with links would be useful so please email **saul@twenteenthcentury.com** with any I miss out and I'll add them. This list includes interfaces to the web as a whole, not to a single web site or database (e.g. Rhizome's alt.browser series).

I/O/D's Webstalker:
http://www.backspace.org/iod

Jodi's Wrongbrowser:
http://www.wrongbrowser.com/

Nullpointer's Web Tracer:
http://www.nullpointer.co.uk/-/webtracer/

Mark Napier's Shredder:
http://www.potatoland.org/shredder/

Tom Corby and Gavin Baily's Reconnoitre:
http://www.reconnoitre.net/

Andi Freeman's Earshot (feat. Jason Skeet), Funksolegrind and notScape:
http://www.deepdisc.com/ns/
http://www.deepdisc.com/earshot/

Mark Dagett's Browser Gestures:
http://www.flavoredthunder.com/dev/browser-gestures/

Many examples generated by the International Browserday competition:
http://www.waag.org/browser

(ii) Later projects and collaborations by the makers of the Webstalker with Mongrel have resulted in projects like the Linker (**http://www.linker.org.uk/**) and its use in training workshops (**http://www.mongrelx.org/**) that fulfil this definition of "not just art" far more successfully. However, an analysis of the tactics employed by Mongrel in using the Linker for "arts education" and their methods of avoiding that institutional dead-end go far beyond the scope of this text.

(iii) 30/10/01 - I've been informed I may be wrong about this. My source was: **http://www.m9ndfukc.org/korporat/04.html**. If anyone can add any information as to how this language game works and what it is called, I would be very grateful.

Concepts, Notations, Software Art

Florian Cramer
March, 2002.

C/O FREIE UNIVERSITÄT BERLIN, SEMINAR FÜR ALLGEMEINE UND VERGLEICHENDE LITERATURWISSENSCHAFT, HÜTTENWEG 9, D-14195 BERLIN, CANTSIN@ZEDAT.FUBERLIN.DE **http://userpage.fu-berlin.de/~cantsin**

Software and Concept Notations

Software in the Arts.

To date, critics and scholars in the arts and humanities have considered computers primarily as storage and display media, as something which transmits and reformats images, sound and typography. Reflection of the, as such, invisible layer of software is rare. Likewise, the term “digital art” has been associated primarily with digital images, music or audiovisual installations using digital technology. The software which controls the audio and the visuals is frequently neglected, working as a black box behind the scenes. “Interactive” room installations, for example, get perceived as a interactions of a viewer, an exhibition space and an image projection, not as systems running on code. This observation all the more applies to works in which it is not obvious at all that their production relied on programmation and computing. John Cage’s 1981 radio play “Roaratorio”, for example, appears to be a tape montage of a spoken text based on James Joyce’s “Finnegan’s Wake”, environmental sounds recorded in several cities of the world and Irish folk music, edited with analog recording technology. Yet, at the same time it is an algorithmic artwork; the spoken text was extracted from the novel using a purely syntactical, formal method (mesostychs of the name “James Joyce”), and the montage was done according to a random score generated on a computer at the Parisian IRCAM studios. While the book-plus-CD set of “Roarotorio” documents the whole composition extensively, containing the audio piece itself, a recording and a reprint of John Cage’s reading, a recording and a reprint of an interview, an inventory of the cities where sound was recorded, it includes the computer generated score itself only in a one-page excerpt and nothing at all of the computer program code which generated the random score.[1]

The history of the digital and computer-aided arts could be told as a history of ignorance against programming and programmers.

Computer programs get locked into black boxes, and programmers are frequently considered to be mere factota, coding slaves who execute other artist's concepts. Given that software code *is* a conceptual notation, this is not without its own irony. In fact, it is a straight continuation of romanticist philosophy and its privileging of aisthesis (perception) over poeisis (construction),[2] cheapened into a restrained concept of art as only that what is tactile, audible and visible. The digital arts themselves participate in this accomplicity when they call themselves [new] "media art". There's nothing older than "new media", a term which is little more than a superficial justification for lumping together a bunch of largely unrelated technologies, such as analog video and computing, just because they were "new" at a particular time. If one defines as a medium something that it is between a sender and a receiver, then computers are not only media, but also senders and receivers which themselves are capable of writing and reading, interpreting and composing messages within the limitations of the rule sets inscribed into them. The computer programs for example which calculate the credit line of checking accounts or control medical instruments in an emergency station can't be meaningfully called "media". If at all, computer processes become "media" only by the virtue that computers can emulate any machine, including all technical media, and by the virtue of the analog interfaces which transform the digital zeros and ones into analog sound waves, video signals, print type and vice versa.

A Crash Course in Programming.

A piece of software is a set of formal instructions, or, algorithms; it is a logical score put down in a code. It doesn't matter at all which particular sign system is used as long as it is a code, whether digital zeros and ones, the Latin alphabet, Morse code or, like in a processor chip, an exactly defined set of registers controlling discrete currents of electricity. If a piece of software is a score, is it then by definition an outline, a blueprint of an executed work?

Imagine a Dadaist poem which makes random variations of Hugo Ball's sound poem "Karawane" ("Caravan"):

> **KARAWANE**
> jolifanto bambla ô falli bambla
> grossiga m'pfa habla horem
> égiga goramen
> higo bloiko russula huju
> hollaka hollala
> anlogo bung
> blago bung
> blago bung
> bosso fataka
> ü üü ü
> schampa wulla wussa ólobo
> hej taat gôrem
> eschige zunbada
> wulebu ssubudu uluw ssubudu
> tumba ba-umpf
> kusagauma
> ba-umpf

The new Dada poem could simply consists of eight variations of the line "tumba ba-umpf". The author/performer could throw a coin twice for each line and, depending on the result, choose to write down either the word "tumba" or "ba-umpf", so that the result would look like:

> tumba tumba
> ba-umpf tumba
> tumba ba-umpf
> tumba ba-umpf
> ba-umpf ba-umpf
> ba-umpf tumba
> tumba ba-umpf
> tumba ba-umpf

The instruction code for this poem could be written as follows:

```
(1) Take a coin of any kind with two distinct sides.
(2) Repeat the following set of instructions eight times:
    (a) Repeat the following set of instructions twice:
        (i) Throw the coin.
        (ii) Catch it with your palm so that it lands on
        one side.
        (iii) If the coin shows the upper side, do the
        following:
            • Say "tumba"
        (iv) Else do the following:
            • Say "ba-umpf"
    (b) Make a brief pause to indicate the end of the line.
(3) Make a long pause to indicate the end of the poem.
```

Since these instructions are formal and precise enough to be as well executed by a machine (imagine this poem implemented into a modified cuckoo clock), they can be translated line by line into a computer program. Just as the above instruction looks different depending on the language it is written in, a computer program looks different depending on the programming language used. Here I choose the popular language “Perl” whose basic instructions are rather simple to read:

```
for $lines (1 .. 8)
       {
       for $word (1 .. 2)
              {
              $random_number = int(rand(2));
              if ($random_number == 0)
                     {
                     print "tumba"
                     }
              else
                     {
                     print "ba-umpf"
                     }
              print " "
              }
       print "\n"
       }
```

The curly brackets enclose statement blocks executed under certain conditions, the $ prefix designates a variable which can store arbitrary letters or numbers, the “rand(2)” function generates a random value between 0 and 1.9, “int” rounds its result to either zero or one, “ ” stands for a blank, “\n” for a line break. This program can be run on virtually any computer; it is a simple piece of software. Complex pieces of software, such as computer operating systems or even computer games, differ from the above only in the complexity of their instructions. The control structures - variable assignments, loops, conditional statements - are similar in all programming languages.

Unlike in the instruction for throwing coins, the artists' work is done once the code is written. A computer program is a blueprint and its execution at the same time. Like a pianola roll, it is a score performing itself. The artistic fascination of computer programming - and the perhaps ecstatic revelation of any first-time programmer - is the equivalence of architecture and building, the instant gratification given once the concept has been finished. Computer programming collapses, as it seems, the second and third of the three steps of concept, concept notation and execution.

Contrary to conventional data like digitised images, sound and text documents, the algorithmic instruction code allows a generative process. It uses computers for computation, not only as storage and transmission media. And this precisely distinguishes program code from non-algorithmic digital code, describing for example the difference between algorithmic composition on the one hand and audio CDs/mp3 files on the other, between algorithmically generated text and "hypertext" (a random access database model which as such doesn't require algorithmic computation at all), or between a graphical computer "demo" and a video tape. Although one can of course use computers without programming them, it is impossible not to use programs at all; the question only is who programs. There is, after all, no such thing as data without programs, and hence no digital arts without the software layers they either take for granted, or design themselves.

To discuss "software art" simply means to not take software for granted, but pay attention to how and by whom programs were written. If data doesn't exist without programs, it follows that the separation of processed "data" (like image and sound files) from "programs" is simply a convention. Instead, data could be directly embedded into the algorithms used for its transmission and output to external devices. Since a "digital photograph", for example, is bit-mapped information algorithmically transformed into the electricity controlling a screen or printer, via algorithmic abstraction layers in the computer operating system, it follows that it could just as well be coded into a file which contains the whole transformation algorithms themselves so that the image would display itself even on a computer that provides no operating system.[3]

Software Art

Executable Code in Art.

If software is generally defined as executable formal instructions, logical scores, then the concept of software is by no means limited to formal instructions for computers. The first, English-language notation of the Dadaist poem qualifies as software just as much as the three notations in the Perl programming language. The instructions only have to meet the requirement of being executable by a human being as well as by a machine. A piano score, even a 19th century one, is software when its instruction code can be executed by a human pianist as well as on a player piano.

The Perl code of the Dada poem can be read and executed even without running it on machines. So my argument is quite contrary to Friedrich Kittler's media theory according to which there is either no software at all or at least no software without the hardware it runs on:[4] If any algorithm can be executed mentally, as it was common before computers were invented, then of course software can exist and run without hardware - a good example are programming handbooks. Although they chiefly consist of printed computer code, this code gets rarely ever executed on machines, but provides examples which readers follow intellectually, following the code listings step by step and computing them in their minds.

Instead of adapting Dadaist poetry as software, one could regard some historical Dadaist works as software right away; above all, Tristan Tzara's generic instruction for writing Dada poems by shuffling the words of a newspaper article:

> To make a Dadaist poem:
> Take a newspaper.
> Take a pair of scissors.
> Choose an article as long as you are planning to make your
> poem. Cut out the article.
> Then cut out each of the words that make up this article and put
> them in a bag.
> Shake it gently.
> Then take out the scraps one after the other in the order in which
> they left the bag.
> Copy conscientiously.
> The poem will be like you.
> And here you are a writer, infinitely original and endowed with a
> sensibility that is charming though beyond the understanding of
> the vulgar.

The poem is effectively an algorithm, a piece of software which may as well be written as a computer program.[6] If Tzara's process would be adapted as Perl or C code from the original French, it wouldn't be a transcription of something into software, but a transcription of non-machine software into machine software.

Concept Art and Software Art.

The question of what software is and how it relates to non-electronic contemporary art is at least thirty-two years old. In 1970, the art critic and theorist Jack Burnham curated an exhibition called "Software" at the Jewish Museum of New York which today is believed to be first show of concept art. It featured installations of US-American concept artists next installations of computer software Burnham found interesting, such as the first prototype of Ted Nelson's hypertext system "Xanadu". Concept art as an art "of which the material is 'concepts,' as the material of for ex. music is sound" (Henry Flynt's definition from 1961 [7]) and software art as an art whose material is formal instruction code seem to have at least two things in common:

> (1) the collapsing of concept notation and execution into one piece;
> (2) the use of language; instructions in software art, concepts in concept art. Flynt observes: "Since 'concepts' are closely bound up with language, concept art is a kind of art of which the material is language".[8]
> It, therefore, is not accidental that the most examples of preelectronic software art cited here are literary. Literature is a conceptual art in that is not bound to objects and sites, but only to language. The trouble the art world has with net.art because it does not display well in exhibition spaces is foreign to literature which always differentiated between an artwork and its material appearance.
> Since formal language is a language, software can be seen and read as a literature.[9]

If concepts become, to quote Flynt again, artistic"material", then concept art differs from other art in that it actually exposes concepts, putting their notations up front as the artwork proper. In analogy, software art in particular differs from software-based art in general in that it exposes its instructions and codedness. Since formal instructions are a subset of conceptual notations, software art is, formally, a subset of conceptual art.

My favourite example, of both concept art in Flynt's sense and non-computer software art, is La Monte Young's "Composition 1961", a piece of paper containing the written instruction “Draw a straight line and follow it”. The instruction is unambiguous enough to be executed by a machine. At the same time, a thorough execution is physically impossible. So the reality of piece is mental, conceptual.

The same duplicity of concept notation and executable code exists in Sol LeWitt’s 1971 “Plan for a Concept Art Book”, a series of book pages giving the reader exact instructions to draw lines on them or strike out specific letters.[10] LeWitt’s piece exemplifies that the art called concept art since the 1970s was by far not as rigorous as the older concept art of Henry Flynt, La Monte Young and Christer Hennix: While the “Composition 1961” is a concept notation creating an artwork that itself exists only as a concept, mentally, LeWitt’s “Plan for a Concept Art Book” only is a concept notation of a material, graphic artwork. Unlike the concept art “of which the material is ‘concepts”’, LeWitt’s piece belongs to a concept art that rather should be called a concept notation art or “blueprint art”; an art whose material is graphics and objects, but which was instead realised in the form of a score. By thus reducing its own material complexity, the artwork appears to be “minimalist” rather than rigorously conceptualist.

A writing which writes itself, LeWitt’s “Plan” could also be seen in a historical continuity of combinatory language speculations: from the permutational algorithms in the Sefer Jezirah and ecstatic Kabbalah to the medieval “ars” of Raimundus Lullus to 17th century permutational poetry and Mallarmé’s “Livre”. The combinatory most complex known permutation poem, Quirinus Kuhlmann’s 1771 sonnet “Vom Wechsel menschlicher Sachen” consists of 13 * 12 nouns can be arbitrarily shuffled so that they result in 10^{114} permutations of the text.[11] Kuhlmann’s and La Monte Young’s software arts meet in their aesthetic extremism; in an afterword, Kuhlmann claims that there are more permutations of his poem than grains of sand on the earth.[12] If such implications lurk in code, a formal analysis is not enough. Concept art potentially means terror of the concept, software art terror of the algorithm; a terror grounded in the simultaneity of minimalist concept notation and totalitarian execution, helped by the fact that software collapses the concept notation and execution in the single medium of instruction code - Sade’s “120 days of Sodom” could be read as a recursive programming of excess and its simultaneous reflection in the medium of prose.[13] The popularity of spamming and denial-of-service code in the contemporary digital arts is another practical proof of the perverse double-bind between software minimalism and self-inflation; the software art pieces awarded at the transmediale.02 festival, “tracenoizer” and “forkbomb.pl” also belong to this category.

La Monte Young’s “Composition 1961” not only provokes to rethink what software and software art is. Being the first and still most elegant example of all artistic jamming and denial-of-service code, it also addresses the aesthetics and politics coded into instructions. Two years before Burnham’s “Software” exhibition, the computer scientist Donald E. Knuth published the first volume of his famous textbook on computer programming, “The Art of Computer Programming”.[14] Knuth’s wording has adopted in what Steven Levy calls the hacker credo that “you can create art and beauty with computers”.[15] It is telling that hackers, otherwise an avant-garde of a broad cultural understanding of digital technology, rehash a late-18th century classicist notion of art as beauty, rewriting it into a concept of digital art as inner beauty and elegance of code. But such aesthetic conservativism is widespread in engineering and hard-science cultures; fractal graphics are just one example of Neo-Pythagorean digital kitsch they promote. As a contemporary art, the aesthetics of software art includes ugliness and monstrosity just as much as beauty, not to mention plain dysfunctionality, pretension and political incorrectness.[16]

Above all, software art today no longer writes its programs out of nothing, but works within an abundance of available software code. This makes it distinct from works like Tzara’s Dada poem which, all the while it addresses an abundance of mass media information, contaminates only the data, not its algorithm; the words become a collage, but the process remains a synthetic clean-room construct.

Since personal computers and the Internet became popular, software code in addition to data has come to circulate in abundance. One thus could say that contemporary software art operates in a postmodern condition in which it takes pre-existing software as material — reflecting, manipulating and recontextualising it. The “mezangelle” writing of mez, an Australian net artist, for example uses software and protocol code as material for writings in a self-invented hybrid of English and pseudo-code. Her “net.wurks” are an unclean, broken software art; instead of constructing program code synthetically, they use readymade computations, take them apart and read their syntax as gendered semantics. In similar fashion, much software art plays with control parameters of software. Software artworks like Joan Leandre’s “retroyou” and “Screen Saver” by Eldar Karhalev and Ivan Khimin are simply surprising, mind-challenging disconfigurations of commercial user software: a car racing game, the Microsoft

Windows desktop interface. They manage to put their target software upside down although their interventions are technically simple and don't involve low-level programming at all.

Software Formalism vs. Software Culturalism.

Much of what is discussed as contemporary software art and discourse on has its origin in two semi-coherent London-based groups. The older one around Matthew Fuller, Graham Harwood and the groups I/O/D and Mongrel is known, among others, for the experimental web browser "WebStalker", which instead of formatted pages displays their source code and link structures, the "Linker", a piece of "social software" (to use a term by Fuller) designed to empower non-literate users to design their own digital information systems, and "natural selection", a politically manipulated web search engine. Fuller also wrote a scrupulous cultural analysis of Microsoft Word's user interface and an essay with the programmatic title "Software as Culture". The other group involves the programmer-artists Adrian Ward (whose "Auto- Illustrator" won the transmediale.01 software art prize) and Alex McLean (whose "forkbomb.pl" won the transmediale.02 software art prize), the theoretician Geoff Cox and participants in the mailing list "eu-gene", the web site **http://www.generative.net** and the "DorkBot" gatherings in London (which involve poetry readings of program code). Both groups take exactly opposite standpoints to software art and software criticism: While Fuller/Harwood regard software as first of all a cultural, politically coded construct, the eu-gene group rather focuses on the formal poetics and aesthetics of software code and individual subjectivity expressed in algorithms.

If software art could be generally defined as an art

- of which the material is formal instruction code, and/or
- which addresses cultural concepts of software,

then each of their positions sides with exactly one of the two aspects. If Software Art would be reduced to only the first, one would risk ending up a with a neo-classicist understanding of software art as beautiful and elegant code along the lines of Knuth and Levy. Reduced on the other hand to only the cultural aspect, Software Art could end up being a critical footnote to Microsoft desktop computing, potentially overlooking its speculative potential at formal experimentation. Formal reflections of software are, like in this text, inevitable if one considers common-sense notions of software a problem rather than a point of departure; histories of instruction codes in art and investigations into the relationship of software, text and language still remain to be written.

Footnotes

1 [Cag82]—Regarding randomness generated with computers, the software artist Ulrike Gabriel says that it doesn't exist because the machine as a fact by itself is not accidental.

2 A similar angle is taken in the paper "The Aesthetic of Generative Code" by Geoff Cox, Adrian Ward and Alex McLean, [CWM01]

3 I would not be surprised if in a near future the media industry would embed audiovisual data (like a musical recording) directly into proprietary one-chip hardware players to prevent digital copies.

4 [Kit91]

5 [Tza75]

6 My own Perl CGI adaption is available under
http://userpage.fu-berlin.de/~cantsin/permutations/tzara/poeme_dadaiste.cgi

7 [Fly61]

8 ibid.

9 But since formal language is only a small subset of language as a whole, conclusions drawn from observing software code can't be generally applied to all literature.

10 [Hon71], p. 132-140

11 [Kuh71]

12 ibid.

13 As Abraham M. Moles noticed already in 1971, [Mol71], p.
124

14 knuth:art

15 according Steven Levy [Lev84]; among those who explicitly subscribe to this is the German Chaos Computer Club with its annual "art and beauty workshop".

16 which is why I think it would be wrong to (a) restrict software art to only properly running code and (b) exclude, for political reasons,

proprietary and other questionably licensed software from software art presentations.

References

[Cag82] John Cage. *Roaratorio. Ein irischer Circus über Finnegans Wake*. Athenäum, Königstein/Taunus, 1982. 1

[CWM01] Geoff Cox, Adrian Ward, and Alex McLean. The Aesthetics of Generative Code, 2001. http://www.generative.net/papers/aesthetics/index.html. 2

[Fly61] Henry Flynt. Concept art. In La Monte Young and Jackson MacLow, editors, *An Anthology*. Young and MacLow, New York, 1963 (1961). 7

[Hon71] Klaus Honnef, editor. *Concept Art*. Phaidon, Köln, 1971). 8

[Kit91] Friedrich Kittler. There is no software, 1991. http://textz.gnutenberg.net/textz/kittler_friedrich_there_is_no_software.txt. 6

[Kuh71] Quirinus Kuhlmann. *Himmlische Libes=küsse*. ?, Jena, 1671. 8

[Lev84] Steven Levy. *Hackers*. Project Gutenberg, Champaign, IL, 1986 (1984). 9

[Mol71] Abraham A. Moles. *Kunst und Computer*. DuMont, Köln, 1973 (1971). 8

[Tza75] Tristan Tzara. Pour fair une poème dadaïste. In *Oeuvres complètes*. Gallimard, Paris, 1975. 6

Interview with Adrian Ward

Dr. Tilman Baumgärtel

This interview was conducted while at the Berlin Beta 2001 conference. It was subsequently updated in March 2002 after Auto-Illustrator 1.0 was released. This interview appears here with the kind permission of Tilman Baumgärtel, to whom Signwave is grateful.

?: You created the graphics program "Auto-Illustrator". Can you explain some of the features of this software?

!: It's quite difficult to do that, because it's so chaotic (I hate that word - it implies nothing). What I tend to say is that the software has its own agenda. For example, when you use the text tool, it generates it's own text instead of rendering the words that you type. There are no consistent tools. "Auto-Illustrator" is inconsistent even in its inconsistency. So some tools will misbehave more than others.

?: Could you describe some of these "misbehaviours"?

?: Sure. If you try to draw a line, it scribbles and makes wild gestures. You can modify these gestures to some extent, but you are never really in control of it. When you put crawling bugs down into the document, they draw lines by themselves, you never quite know what they are going to do. They might even make the software crash, which actually happens quite often. (laughs)

!: Do you take these features in and out from one version to another?

?: Yes, there is quite a lot of features that get added and then removed in the next version, because I changed my mind about them or because they're not working out. Sometimes the changes are to what's available, sometimes it's a change in the way a certain feature operates. The users have to get used to how the features work and need to learn how to get certain results from them. You have to cooperate or not cooperate with each version...

?: So in this sense your program is not really that different than other software, like Adobe...

!: True, but Adobe wants as many people as possible to cooperate with their programs, because they don't want to lose users. I don't really have that problem. I can abuse my users as much as I want.

?: But is the idea to terrorize people with Auto-Illustrator? After all, you can create something constructive with it...

!: Yeah, that's one of the minor points of it. It is supposed to be a play on a useful utility, and something that really changes your expectations. When you come to it, you're expecting it to behave like a professional piece of software, because it looks like a professional piece of software, but it's not. It has no respect for you. But on the other hand sometimes the complete opposite is true.

?: how would you explain to a traditional art audience why it can be art to write software

!: That's a difficult one. I'm not too interested in trying to be an artist, even though I am obviously creating something that is culturally aware and critiques existing systems, and I think those are important factors for an artwork. Software as art challenges ideas about how you interact with systems. People put computers into galleries, which is of course completely misguided, because the whole point of software is that you get hold of it and run it on your own computer at home or work. How do you curate a piece of software? It's difficult; I'm just trying to create more problems. (laughs)

?: There is a traditional point of view, that art should be without function..

!: I disagree with it. I think that art functions as a social or cultural catalyst and provokes certain reactions and thoughts about things that are important. Should art be purely expressive? Or does it have some kind of functionality? I'm obviously confusing the issue more, because Auto-Illustrator is an expression in the form of a tool. Also I think we need to be careful to distinguish between the specific functionality of a tool, and the social functionality of an artwork.

?: If you look at the program for the first time, it looks like an ordinary drawing program. I think that there are some metaphors in the software that run amok, they seem to have a life of their own but others are useful, for example the generative parts...

!: The project has really gotten out of control. It started out as a playful kind of toy, and now it is really this kind of platform for me to explore certain generative ideas. We have this plug-in system, where you write a piece of code, and Auto-Illustrator executes this code and produces a design. Apart from being a challenging application and a bit of joke, it is also a serious platform for many people to express themselves visually through code. That's one of the things I am very interested in.

?: There are generative parts in the software, that create designs according to certain parameters that you define. Does that exist in programs like Photoshop or Adobe Illustrator?

!: To a degree, it depends on how you use the term "generative". You could actually argue that all software is generative, just by the fact that code is pretty useless without some form of output. But you do see a lot of generative features in other graphics programs. One thing I want to articulate with Auto-Illustrator is that a lot of the features that you see in Photoshop and Illustrator are a bit pointless. I am trying to parody this useless kind of feature. What I am doing is turning it around, trying to highlight that you shouldn't necessarily accept what a programmer provides you with. You should try to mutate the actual code (or in my case, rewrite it), and use your own code to express yourself.

?: Auto-Illustrator keeps creating new patterns, shapes and other graphical elements. I guess most designers don't want that in "normal" programs, because at one point they will want to output their piece as a hard copy, and you cannot show the generative movement of an object on a piece of paper.

!: That's why I am fascinated by generative processes, because you don't have a fixed end result. If you can produce a system that can change the rules for generation, the final product you get is not fixed at all. I think there is a lot more power in that. I think that people are getting used to dynamic systems. I think that fixed products are kind of out-dated. It's not my problem how deal with translating variable results into fixed products - that's something for the users to deal with.

?: Isn't it a paradoxical situation: you as the creator should know best, what the software can do and how to work with it, yet you see other people making creative use of it...

!: It is a bizarre thing to see other people using my software. I know what the most optimal way of using it is, but then I see other people coming along and they're not using it in that matter.

?: Are they right or wrong?

!: I don't think that there is any specific right or wrong way. If I don't adhere to conventions when writing my software, why should my users do the same? They are trying to find their own way of using it; They work out how to do things their own way. And it's great that they find their own uses for it. It's one of the great things of writing software that people adapt it in new and exciting ways.

?: Do you ever see pieces that were created with "Auto-Illustrator", and ask yourself: How did they do that?

!: Actually yes. We get guest designers for the splash screens of every version of Auto-Illustrator. And sometimes I don't even recognize that they have used my software, because I don't recognize what they have done. That's just a sign that they really worked with the software, and take it much further than I initially imagined, which is an interesting model. I have taken an active role in designing it, but actually the end result is beyond what I imagined. So it blurs the boundaries between producer and consumer even more. I like that.

?: I guess at companies like Adobe there are hundreds if not thousand of programmers working on new versions of their software. How is it with Auto-Illustrator?

!: Well, it's just me. I just sit there and write the software, but part of the artwork is obviously a parody of a huge multinational megacorporation, so we manufacture artificial employees and have them answering customer's emails. I fool myself sometimes - I keep on referring to Signwave as "we", when in fact it's just "me". I started twelve months ago, and it's just not finished yet. Not that I have any specific finishing point, though. The thing about Adobe is that they have to be more careful. They have to write software that works the first time, and doesn't crash and behaves exactly the way it is supposed to. It's a massive job to do that. The process I am going through is more of a personal expression of an artist, so perhaps I'm not so concerned with the same issues. If something doesn't work properly, I maybe tweak it slightly until it does work. It's not perfect. But those kind of problems get absorbed by the fact that people know that the software's idiosyncrasies are a reflection of my own sense of humour. You see that when you use the software, and you go through the preferences and dialog boxes.

?: So would you say that this software is kind of a digital self-portrait?

!: Yes, exactly. That was the key thing, when I started off writing generative software. Because I saw writing code as a way to express myself, and I was interested in making something that behaved perhaps a little like me. I am not a graphic designer, I am a programmer by history, so I use my skills as a programmer to express myself. Code is the basic medium that I choose to use to allow other people to interact with me (or my work?).

?: But a lot of the features of the software are very standardized: there are drop-down menus, there are icons, there are features that are not so different from other graphics software. How does this express your personality? There doesn't seem to be a lot of artistic freedom in these standardized features.

!: I am definitely trying to parody Adobe's interfaces, so it wouldn't really make sense to move completely away from the traditional UI standards of software. I also have this problem with radical interfaces. Just because software does something different, it doesn't mean that it needs a new interface. Yet everybody is trying to invent this perfect new interface, and everybody has this big hang-up about the GUI and Windows, Icons, Menus and Pointers. They say it is out-dated, but the reason it has been around for such a long time is because it simply works. To be more specific, code isn't really the medium I'm using - I want to express myself using the medium of consumer-based application software, which is why Auto-Illustrator doesn't have a radical interface. It doesn't need one. It wouldn't be a parody otherwise, too.

?: How did you get the idea to write your own graphics software?

!: It all came from my childhood as a programmer...

?: Your *childhood* as a programmer?

!: I started programming when I was seven. My parents bought me a ZX Spectrum, a little 8-bit home computer. They thought that I wasted my time on it, and I probably did. I wrote programs for it that scribbled and so on. I never had a proper education as a programmer. If I had, then I'd be writing SQL statements for banks or something uninspiring like that. One thing I am trying to say is that code doesn't really have to be boring, especially if you explore it's potential for yourself. SELECT INSPIRATION FROM CODE WHERE EXPRESSION IS NOT NIL.

?: But how does somebody with a background in programming come up with the idea to write a graphics program?

!: I was always interested in design and popular culture, and how trends spread in graphic design. I just observed that certain kind of behaviours exist, for example in web design, which has become a bit stale. Because I'm not a graphic designer by trade, I don't really have much to lose. I can get away with being troublesome. I would just like to see more interesting uses of technology in graphic design.

?: What your program seems to imply is that a computer could do what most designers are doing...

!: The specific point I have there is about appropriation. Whatever a company like "The Designers Republic" is doing, people just copy it ("Talent borrows, genius steals, shit copies”). It must be very hard for them to innovate and do new stuff. What I am saying is that copying other people's stuff is not very constructive. If I can automate that process then maybe I'll put all the uninnovative designers out of business. (laughs)

?: In the Nineties we saw how the computer changed certain elements of pop culture. Techno Music, Webdesign, Typography, Cinema - a lot of different cultural forms were shaped by the use of digital technology. Now these impulses seem a bit worn-out. Electronic music sounds more and more alike, the same is true for the covers of the CDs that were created on the computer etc. Is "Auto-Illustrator" also an attempt to address this situation?

!: The idea is that technology is not just a delivery medium or tool, it is a production system. You could use a computer to score some music or design a CD cover, but what I am saying is: You can also use technology in a much more interesting manner, and maybe that's the problem that has inflicted pop culture - not enough tools and too much complacency. I'd like to see a much more engaging use of technology through the use of generative systems. It would be great if more people would realize what they can do with technology rather than just using what is given to them as a tool. They can extend the technology themselves, and that is the really critical part.

?: Do you see yourself working on another kind of software, for example an Operating System? Or will you continue to work on Auto-Illustrator?

!: I'd love to work on an Operating System. I'd like to see an Operating System that misbehaves intentionally. Auto-Illustrator will never be finished. There will be a version 1.0, but that wont be the end of it. I don't think that I will ever be able to put a cap on it, because it's a platform for my interests. The difficult thing is to bounce it between being a commercial product and a medium for my new ideas.

?: The British artist Harold Cohen has attempted to build a robot, that can paint pictures completely autonomously. Do you know him? Because I think there a lot of obvious similarities between your work and his.

!: I don't know him personally, but I am familiar with his work. He has an interesting attitude. It's a massive undertaking to develop software that operates like the human brain. What I find interesting about Cohen's work is that Aaron doesn't do this through Artificial Intelligence, which I think is a buzzword used in tedious scientific research. I think you can express yourself by writing code and still achieve a similar state of semi-autonomous code. It's just a shame that he spent 20 years writing this system "Aaron" and now it looks like a screensaver (laughs). His work is quite influential for me, because he has used code as a form of personal expression.

?: Could you comment on your decision to sell the software now: Other art projects are freely downloadable from the net, why did you decide to sell yours? How many people have bought it so far?

!: Auto-Illustrator is a parody of commercial software, so it makes sense to commodify it. It's also a response to the difficult question of how do you treat software art, especially if you want to curate it? You sell it in boxes, just like you might in PC World. We're doing that for Auto-Illustrator 1.1, which is going to be on released as a boxed set for the GENERATOR show at Spacex in Exeter. It's going to be very exciting. A few people have bought Auto-Illustrator 1.0 - enough to ensure I carry on developing it.

INDEX