Generative Erzeugung von Design mit vvv

Diplomarbeit zur Erlangung des akademischen Grades „Magister (FH)“

Verfasser: Thomas Hitthaler

Vorgelegt am FH-Studiengang MultiMediaArt, Fachhochschule Salzburg

Begutachtet durch

Dipl.Ing. Brigitte Jellinek

Dipl.-Ing.(FH) Claudius Masuch

Wien, 20.05.2005

Hiermit versichere ich, Thomas Hitthaler, geboren am 02.06.1980 in Innichen (Italien), dass ich die Grundsätze wissenschaftlichen Arbeitens nach bestem Wissen und Gewissen eingehalten habe und die vorliegende Diplomarbeit von mir selbstständig verfasst wurde. Zur Erstellung wurden von mir keine anderen als die angegebenen Quellen und Hilfsmittel verwendet. Ich versichere, dass ich dieses Diplomarbeitsthema weder im In- noch Ausland bisher in irgendeiner Form als Prüfungsarbeit vorgelegt habe und dass diese Arbeit mit der den BegutachterInnen vorgelegten Arbeit übereinstimmt.

Wien, 20.05.2005

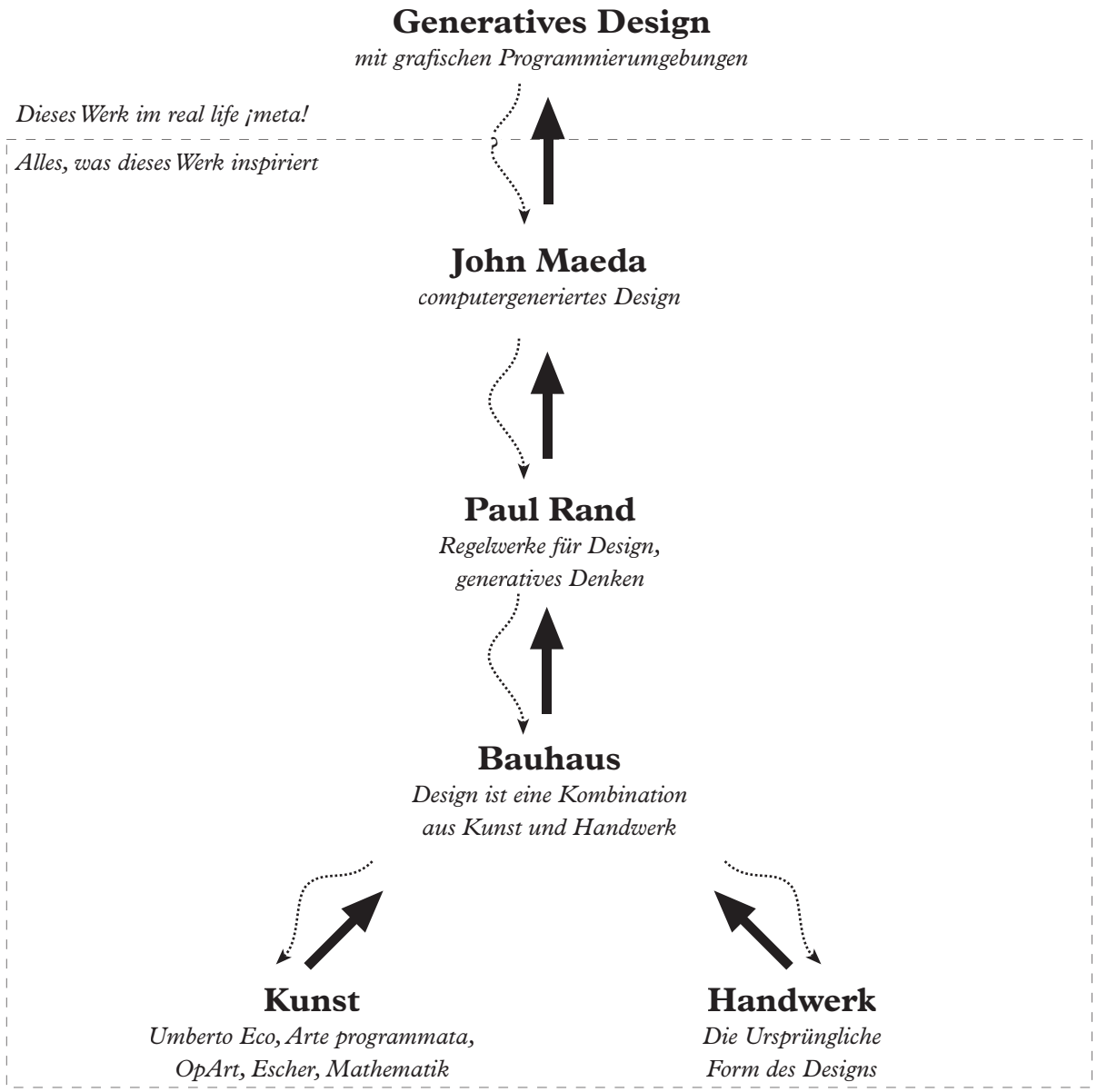
Thomas Hitthaler, Matrikelnummer : 0110055017

Wie sich während der Arbeit an dieser Diplomarbeit herauskristallisiert hat, ist die Nähe von Kunst und Design ein entscheidender Faktor bei der Diskussion von generativem Design. Ausgehend von der Kombination von Design mit mathematischen Algorithmen war John Maeda, einen Pionier des programmierten Designs, der nächste logische Schnitt für meine Recherchen. Während dieser zeigte es sich, dass sich John Maeda sehr stark von Paul Rand inspiriert fühlt. Rand trieb ab 1930 die Kombination von handwerklichem Grafikdesign mit den Freiheiten der Künste in großen Schritten voran und kann zurecht als Visionär gesehen werden. Jedoch zeigte sich Maeda insbesondere durch Rands generative Arbeits- und Denkweise beeindruckt. Und das obwohl Rand nie einen Computer für seine Arbeit eingesetzt hat.

Durch die starke Prägung von Rands Werk und Leben vom damals stattfindenden Bauhaus lässt sich eine Verbindung zu Künstlern wie M.C. Escher und den Vertretern der „Arte Programmata“ (Umberto Eco) herstellen, was die Basis dieses Werks darstellt.

Diese Kette von gegenseitigen Inspirationen ist auf der nächsten Seite als Flowchart visualisiert und verdeutlicht die Assoziationskette auf die meine Definition von generativem Design aufbaut. So konnte gezeigt werden, dass die von John Maeda vertretenen Ideen über programmiertes Design auch für visuelle Programmiersprachen gelten. Dass diese leichter erlernbar sind als eine textbasierte Syntax ließ sich dann leicht belegen.

Deswegen bedanke ich mich bei John Maeda für seine direkte und indirekte (meta) Inspiration. Weiterer Dank gebührt Sebastian Oschatz für die technische Expertise und meiner Mutter Waltraud für das Korrekturlesen. Bei Franz und Benno bedanke ich mich für deren Analysen und Kritiken. Den hilfreichen Mitarbeitern von Zanders Papier und diversen Papiergroßhändlern will ich ebenso danken. Ohne sie wäre diese Seite aus wesentlich langweiligerem Papier.


Generatives Design
mit grafischen Programmierumgebungen
Dieses Werk im real life ¡meta!
Alles, was dieses Werk inspiriert
John Maeda
computergeneriertes Design
Paul Rand
Regelwerke für Design,
generatives Denken
Bauhaus
Design ist eine Kombination
aus Kunst und Handwerk
Kunst
Umberto Eco, Arte programmata,
OpArt, Escher, Mathematik
Handwerk
Die Ursprüngliche
Form des Designs

Kurzfassung

Von der programmierten Kunst wird der Begriff des „generativen Designs“ abgeleitet, welches neue Möglichkeiten für den Designer bereitstellt um die Effizienz und Qualität seiner Arbeit zu steigern. Es werden verschiedene Beispiele für programmierte Grafik besprochen und die Techniken erklärt, die dafür eingesetzt werden können.

Anschließend wird festgestellt, dass zur Erstellung von generativem Design eine andere Denkweise, als die den Grafikdesignern vertraute, notwendig ist.

Ein mathematisches Grundverständnis und der Umgang mit einer Programmiersprache sind unerlässlich. Zudem muss der Designer in der Lage sein, seine visuellen Ideen in ein Regelwerk zu übersetzen das von einem Computer verarbeitet werden kann.

Die etablierten Ansätze für die Erzeugung von generativem Design benutzen allerdings textbasierte Programmiersprachen, die nicht an die Bedürfnisse von visuell denkenden Designern angepasst sind. Aus diesem Grund wird eine grafische Programmierumgebung vorgestellt die alle an generatives Design gestellten Ansprüche erfüllt. Zusätzlich dazu besitzt die in diesem Werk vorgestellte grafische Programmierumgebung „vvvv“ Funktionen, welche die Weiterverarbeitung von erstellten Grafiken vereinfacht und beschleunigt.

Die so geschaffenen Voraussetzungen können den Einsatz von generativen Designtools auch für Designer erleichtern, die keine Erfahrung mit Programmiersprachen haben.

Abstract

Because the term “generative design” is heavily influenced by the „programmed art“ movement, some historical examples for programmed graphics and the techniques used for their creation will be explained. This could help to extend the possibilities of the designer and to increase the quality and efficiency of his work.

But the creation of generative design demands another approach as the one common to designers. A basic understanding of mathematics and the use of a programming language are required, as well as the skill of expressing visual ideas in a way a computer can process.

The well-established methods are all text-based languages and do not suit well the needs of visually thinking designers.

This leads to the introduction of a graphical programming environment, which satisfies all demands of generative design. In addition, the discussed environment “vvvv” offers many functions to accelerate the handling of generated designs.

These prerequisites should ease the use of generative design-tools, even for those who never used a programming language.

Inhaltsverzeichnis

Generatives Design mit grafischen Programmierumgebungen und deren Eignung für Nicht-Programmierer

Einleitung

Im Zentrum der wissenschaftlichen Arbeit steht die Frage wie die grafische Programmierumgebung vvvv und andere generative Technologien in den Designprozess eingebunden werden können um programmiertes Design zu erzeugen. Es werden alternative Herangehensweisen zur Erzeugung von Designstücken vorgestellt und mit dem etablierten Designprozess verglichen. Dies geschieht in der Motivation, Wege aufzuzeigen, die einen leichteren Einstieg in die Materie erlauben, um den zur Zeit noch sehr geringen Einfluss dieses generativen Ansatzes auf die Welt der Designer zu vergrößern.

Angesichts der Tatsache, dass sich die Kunst schon lang mit einer kontrollierten Kontrollabgabe an Umwelteinflüsse oder Automatismen, die das Werk verschieden stark beeinflussen, angefreundet hat (Jackson Pollock, die OpArt, usw.) und sich dieser Methoden mehr oder minder stark bedient, wird gezeigt, dass sich die Methoden der von Umberto Eco propagierten „programmierten Kunst“ auch den Bedürfnissen von Designern anpassen lassen.

Bevor jedoch einem Programm anhand von vorgegebenen Regeln beigebracht wird, Designstücke zu erzeugen, sind Gedanken über die Fähigkeiten dieser Methode notwendig. Dabei wird zu keinem Zeitpunkt davon ausgegangen, dass generatives Design die gebräuchlichen Methoden ersetzten kann. Vielmehr soll es eine Ergänzung darstellen, die alle Designer nutzen können um die Erweiterung der Möglichkeiten, die sich durch eine Kompetenzabgabe an die Maschine ergeben, für ihre Arbeit einsetzen zu können. Eine fast beliebige Anzahl von Elementen zu komplexen Mustern zu arrangieren und endlose Variationen zu erzeugen ist für eine Maschine ein Leichtes. Der Mensch überlegt sich nur noch, wie dies geschehen soll und sucht eine Beschreibung für das gewünschte Verhalten. Die Maschine kümmert sich um die Ausführung dieser Anweisungen.

Die Auswahl der in diesem Werk besprochen Software soll die Grundlage für die Analyse von „generativem Design“ und dessen Anwendungsgebieten darstellen.

Die Einbeziehung der Kunst in dieses Werk war notwendig, da diese sehr viele Überschneidungen und Gemeinsamkeiten mit der Welt des Designs aufweist. Oder, wie es Oswald Wiener formulierte: „*Sie (eine Erkenntnistheorie) wird […] die Steuerung des menschlichen Erlebens ermöglichen, so dass sich […] ein großer Teil des heutigen Kunstmachens als […] Design herausstellen könnte.*“ (Wiener 1969).

Anzumerken bleibt noch, dass in diesem Werk aus Lesbarkeitsgründen nicht zwischen weiblichen und männlichen Wortformen unterschieden wird. Bezeichnungen wie beispielsweise „Designer“ oder „Anwender“ meinen also immer beide Geschlechter.

{1} Programmierte Grafik
Geschichte und Anwendung

Wenn in diesem Buch von „generativem Design“ gesprochen wird, bedeutet das, dass es ein Programm gibt, das Grafiken aufgrund von zuvor vom Designer programmierten Verhaltensweisen erzeugen kann. Dies ist ein anderer Zugang zum Design als der gewohnte und deshalb soll in diesem ersten Kapitel geklärt werden, wie sich generatives (oder „programmiertes“) Design erzeugen lässt, welche Anwendungsprogramme und Entwicklungsumgebungen den Zugang dazu beschleunigen können und welches Grundwissen und welche Grundeinstellung dazu benötigt werden.

In Kapitel 1.1 wird erklärt, wie man generatives Design definiert. Was die klassischen Designtools leisten und welche Ansprüche deswegen an die generativen Tools gestellt werden müssen, soll in Kapitel 1.2 erläutert werden. In Kapitel 1.3 wird gezeigt, welche etablierten Technologien verfügbar sind um programmiertes Design zu erzeugen und dass diese vom Benutzer einige Programmiererfahrung fordern. Das ist der Grund, warum in Kapitel 1.4 eine grafische Strategie vorgestellt wird, die sich auch für erklärte Nicht-Programmierer anbieten könnte.

1.1. Über generatives Design

In diesem Kapitel wird “generatives Design” definiert. Dazu werden einige Definitionen aus verwandten Forschungsgebieten herangezogen werden, da es im Designkontext keine eindeutige Definition zu geben scheint.

1.1.1. Arte programmata und OpArt

Eine sehr schöne Entsprechung, die allerdings in einem Kunstkontext steht, ist der, 1962 von Umberto Eco geprägte, Begriff „arte programmata“. Im Vorwort des Kataloges zu einer von Bruno Munari und Giorgio Soavi organisierten Kunstausstellung in Mailand schreibt er:

> *“Nelle vicende del caso può essere individuato a posteriori una sorta di programma e non sarà dunque impossibile programmare, con la lineare purezza di un programma matematico, ‘campi di accadimenti’ nei quali possano verificarsi dei processi casuali. Avremo così una singolare dialettica tra caso e programma, tra matematica e azzardo, tra concezione pianificata e libera accettazione di quel che avverrà, comunque avvenga, dato che in fondo avverrà pur tuttavia secondo precise linee formative predisposte, che non negano la spontaneità, ma le pongono degli argini o delle direzioni possibili. Possiamo così parlare di arte programmata”*
>
> *(Eco 1962, 23)*

Da keine gesicherte deutsche Übersetzung dieser Aussage Ecos gefunden werden konnte, muss eine Übersetzung des Autors genügen:

> *„In Bezug auf die vorliegenden Werke kann im Nachhinein, eine Art von Programm festgestellt werden, was die Möglichkeit beweist, mit der linearen Reinheit eines mathematischen Programmes Vorkommnisse zu programmieren, die natürliche Prozesse nachbilden. So wird ein dialektischer Zusammenhang zwischen Werk und Programm, zwischen Mathematik und Risiko, zwischen formal geplanten Prozessen und der Akzeptanz des Zufalls hergestellt, was angesichts der Tatsache, dass im Grunde alles auf präzisen formalen Kriterien beruht, die Spontaneität nicht negiert, sondern ihr Grenzen und mögliche Richtungen vorgibt.*
> *So können wir also von programmierter Kunst sprechen.“* ¬ 13

Eco sagt also, dass ein direkter Zusammenhang zwischen den bei der Ausstellung gezeigten Arbeiten und ihrer zugrunde liegenden Struktur, aus den Werken selbst hergestellt werden kann, da ihre Ausführung auf sehr präzisen formalen Vorgaben beruht, welche die Freiheit nicht einschränken, sondern ihr nur richtungsgebend sind. (vgl. Eco 1962, 23). Es gibt also einen thematischen oder szenischen Zusammenhang, der einem komplexen oder mehrteiligen Kunstwerk zugrunde liegt, aufgrund dessen ein Künstler sein Werk schaffen kann.

Zuerst werden Regeln definiert und durch die Einhaltung dieser führt das zu einem programmierten Ergebnis.

Während die „Arte programmata“ nur in Italien ein Bestandteil der Kunstszene zu sein scheint, haben Spielarten der von Eco beschriebenen Strömung (Anmerkung: Eco selbst ist kein Künstler der Arte Programmata, sondern hat nur den Begrff geprägt!) starke Popularität erfahren. Die „Op Art“ (ein 1964 von George Rickey geprägter Begriff) war zwar nie so populär wie die gleichzeitig herrschende Pop Art, schaffte es aber dennoch, 1963 ein wichtiger Bestandteil der Modebranche zu werden. Ziel von Vertretern dieser Kunstrichtung, unter anderem Josef Albers, Victor Vasarely und M.C. Escher (siehe Abb. 1), war es, die Sinne zu überlisten und Täuschungen zu schaffen.

> *“Optical Art is a mathematically-oriented form of (usually) Abstract art, which uses repetition of simple forms and colors to create vibrating effects, moiré patterns, an exaggerated sense of depth, foreground-background confusion, and other visual effects.”*
> *(Artcyclopedia 2003)*

Es werden also mathematische Verhältnisse gesucht, die gewisse Effekte erzielen und diese dann zu einem Kunstwerk verarbeitet.

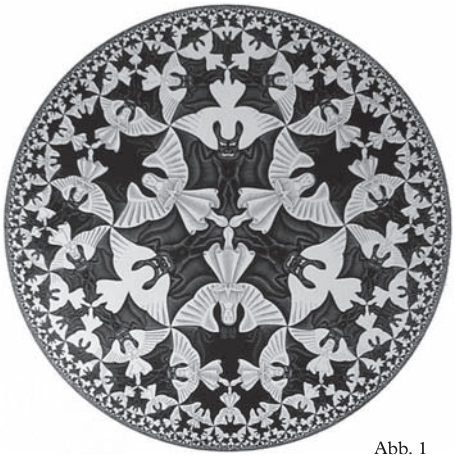


Abb. 1

1.1.2. Einführung in die programmierte Bildgenerierung

Diese, im vorigen Kapitel angesprochenen mathematischen Verhältnisse können in sogenannte „Algorithmen“ übersetzt werden. Obwohl in der ersten Hälfte des 20. Jahrhunderts eine ganze Reihe von Ansätzen entwickelt wurden, um zu einer genauen Definition des Begriffs „Algorithmus“ zu kommen, gibt es nach wie vor keine allgemein akzeptierte Definition des Begriffes. Unter anderem hat Alan Turing eine formale Maschine zur Berechnung von Algorithmen (die Turing-Maschine) konstruiert, welche die folgende Definition eines Algorithmus ermöglicht:

> *„Eine Berechnungsvorschrift zur Lösung eines Problems heißt Algorithmus genau dann, wenn eine zu dieser Berechnungsvorschrift äquivalente Turingmaschine existiert, die für jede Eingabe stoppt.“*
>
> *(Turing 1936)*

¬ 15

In Verbindung mit der Church-Turing-These (*„Jedes intuitiv berechenbare Problem kann durch eine Turingmaschine gelöst werden.“*, Church 1932) und der Tatsache, dass sich eine Turingmaschine von einem Computer simulieren lässt, kann ein Programm geschrieben werden, das Algorithmen berechnen kann. Turings Einschränkung, dass diese Berechnungsvorschrift auch zu einem Endergebnis führen muss, ist damit allerdings noch nicht sichergestellt. Der Begriff der Berechenbarkeit ist dadurch definiert, dass ein Problem nur berechenbar ist, wenn es eine Abbruchbedingung für ein gewisses Problem gibt, d. h. wenn eine entsprechend programmierte Turing-Maschine das Problem in endlicher Zeit lösen kann. Sonst würde eine Endlosschleife entstehen, die nie ein Ergebnis zurückliefern würde. Der Positivist Wittgenstein sagte in Bezug auf die menschliche Kreativität etwas ganz ähnliches: *“Wenn sich eine Frage überhaupt stellen lässt, so kann sie auch beantwortet werden.”* (Wittgenstein 1921).

Legt man diese Aussage auf ein Problem im Kunst- und Designkontext um, bedeutet dies, dass jedes Werk, das sich explizit in ein formales Regelwerk übersetzten lässt, auch als maschinenlesbarer Algorithmus beschrieben werden kann. Für generatives Design im Sinne dieses Werks, scheint demnach die von Peter Weibel gelieferte Definition passend:

> *„Unter einem Algorithmus versteht man eine Entscheidungsprozedur, eine Handlungsanweisung, die aus einer endlichen Menge von Regeln besteht, eine endliche Folge von eindeutig bestimmten Elementaranweisungen, die den Lösungsweg eines spezifischen Problems exakt und vollständig beschreiben."*
>
> *(Weibel 2004)*

Der erste für einen Computer gedachte Algorithmus wurde 1842 von Ada Lovelace, in ihren Notizen zu Charles Babbages *Analytical Engine* (vgl. Stein 1984), festgehalten. Sie gilt deshalb als die erste Programmiererin. Weil Charles Babbage seine *Analytical Engine* nicht vollenden konnte, wurden Ada Lovelaces Algorithmen nie darauf implementiert.

Als Beispiel für die Illustrierung des Begriffs Algorithmus soll der Euklidische Algorithmus dienen, der bereits um 300 v. Chr. beschrieben wurde und noch immer für die Ermittlung des größten gemeinsamen Teilers zweier natürlicher Zahlen A und B Verwendung findet:

> *1. Sei A die Größere der beiden Zahlen A und B (entsprechend vertauschen, falls dies nicht bereits so ist)*
>
> *2. Setze A = A – B*
>
> *3.Wenn A und B ungleich sind, dann fahre fort mit Schritt 1, wenn sie gleich sind, dann beende den Algorithmus: Diese Zahl ist der größte gemeinsame Teiler.*
>
> *(vgl. Euklid)*

Algorithmen können sehr viele Rechenschritte erfordern um zu einem Ergebnis zu gelangen und oft sind tausende Berechnungsschritte notwendig bevor ein Wert zurückgeliefert wird. Diese Rechnungen stur durchzukalkulieren war früher die Aufgabe von Studenten und Gehilfen, da zur Erfüllung ihrer Aufgabe nur ein minimales Grundverständnis der Mathematik benötigt wird. Die große Herausforderung ist die Konstruktion eines sinnvollen Algorithmus, nicht dessen Berechnung.

Die sich rasant entwickelnde Computertechnologie, angefangen bei mechanischen Rechenmaschinen über Relais und Elektronenröhren bis hin zu Transistoren und deren ständige Verkleinerung, bildete stets die Grundlage für die Berechnung immer ausgefeilterer und komplexerer Algorithmen in immer kürzerer Zeit und mit größerer Präzision. Der Mensch hat so die Abarbeitung von Entscheidungsverfahren in eine Maschine ausgelagert.

Für mehr Information zu Algorithmen und deren Berechnung empfiehlt sich die Lektüre von „Einführung in die Automatentheorie, Formale Sprachen und Komplexitätstheorie“ (Hopcroft 2002).

Um nun den Bogen zurück zum ursprünglichen Thema, der Anwendung von Algorithmen in Kunst und Design zu spannen, wird eine wichtige Gemeinsamkeit von Op Art und arte programmata herangezogen: beide stellen Regeln auf, die zu befolgen sind, um ein gewünschtes Resultat zu erzielen. Und beide verwenden mathematische Algorithmen dazu.

Die von den mathematischen Berechnungen gelieferten Ergebnisse wurden von den Künstlern dieser Kunstströmungen meist händisch auf ein Medium (z.B.: Leinwand) übertragen. Für die in diesem Werk verfolgten Zwecke wird jedoch, sowohl für die Berechnung als auch für die Darstellung, ein Computerprogramm verwendet. Es sollen Regelsysteme mit Hilfe von Algorithmen aufgestellt und diese dann in den Computer eingegeben werden. Alles Weitere soll frei von menschlicher Intervention vonstatten gehen und vollautomatisch ablaufen.

Während es Leonardo Da Vinci 1509 noch ziemlich schwer hatte, die festgelegten Proportionen seines „Uomo Vitruviano“ (Abb.2) selbst zu berechnen und einzuhalten, könnte er heute einen Computer so programmieren, dass dieser, ohne Leonardos zutun, sicherstellen würde, dass die zuvor definierten Regeln eingehalten werden. Alles, was außerhalb dieses Regelsystems liegt, könnte er dann noch auf gewohntem (dem händischen) Weg ergänzen.

Genauso wie es der Computer erlauben würde, den, nach Leonardo Da Vincis Vorstellungen, perfekten Menschen automatisch zeichnen zu lassen, kann ein Computerprogramm einem Designer helfen, ein gewünschtes Design zu erzeugen.

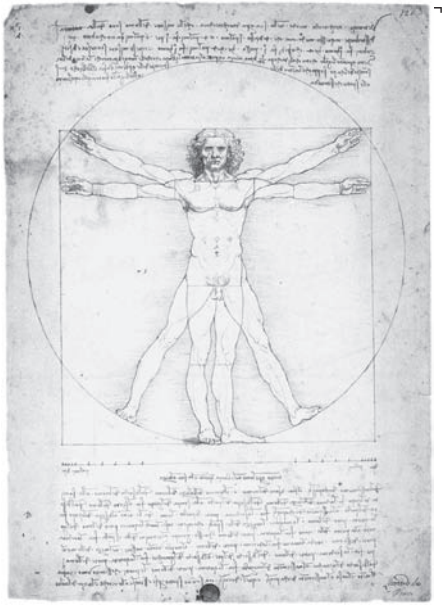


Abb. 2

Allerdings nur, wenn sich das zu erzielende Resultat so formulieren lässt, dass es der Computer umsetzen kann. Ist dies geschehen, braucht sich der Designer nur noch das für seinen Zweck schönste Design auswählen und dieses dann entweder unverändert übernehmen oder beliebige Änderungen mit anderer Software vornehmen. Jedes einzelne Element oder jede beliebige Gruppe von Elementen kann, bei entsprechender Vorarbeit, direkt als Ebenen in ein Bildbearbeitungsprogramm importiert werden. Hier bieten sich dem Designer wieder alle gewohnten Möglichkeiten des klassischen Designs mit dem Computer, aber er muss nicht mit einem leeren Dokument starten, sondern hat schon passendes Ausgangsmaterial, dem er nur noch den letzten Schliff geben muss. Selbstverständlich hat er auch die Möglichkeit, im Zuge seiner Nachbearbeitung das gesamte Design so zu verändern, dass es mit dem automatisch generierten Ausgangsmaterial kaum noch Ähnlichkeiten aufweist. Jedem ist hierbei selbst überlassen inwieweit, wenn überhaupt, eine weitere Bearbeitung des generierten Materials notwendig ist. Die automatisch erzeugten Design-Rohlinge können auch nur zur Ideenfindung benutzt werden und daraus dann ein endgültiges Design entwickelt werden, das den generierten Vorlagen mehr oder weniger ähnlich sein kann.

1.1.3. Generatives Design im Sinne des beiliegenden Bildbandes

Auf den ersten Blick scheint es sehr viele Beispiele für algorithmische Kunst zu geben, doch keinerlei Entsprechungen im Designkontext.

Bei einer etwas genaueren Betrachtung der Aufgabengebiete und der geschichtlichen Entwicklung von Design lassen sich jedoch einige Parallelen zu generativem Design finden. So war das, einst nur von Druckern und Textsetzern ausgeübte, Design-Handwerk frei von künstlerischen Idealen und verfolgte nur kommerzielle Ziele. Die Arbeit des Textsetzers war den strengen Regeln und Einschränkungen des Bleisatzes unterworfen. Erst das Bauhaus ermöglichte um 1920 die Emanzipation des Designs.

Von der Idee getrieben, die ökonomischen Barrieren zu eliminieren und während der Depression in Deutschland allen ein besseres Leben zu ermöglichen, hinterfragten die Vertreter dieser Strömung (u.a. Walter Gropius und Ludwig Mies van der Rohe) die althergebrachten Konventionen. Die vom Bauhaus eingeführte Verschmelzung von künstlerischen Idealen mit dem sehr mechanisch betriebenen Handwerk führte, auch durch Einflüsse von Kubismus und Konstruktivismus, zu einer neuen Bildsprache, die mehr war als ein bloßes Handwerk, aber trotzdem viel zweckgebundener als die Kunst.

Begriffe wie Ästhetik und die Diskussion um das Verhältnis zwischen Design und Kunst erlangten große Bedeutung. Was vielen Designern auf diesem Weg verloren ging, ist die Fähigkeit den Designprozess zu beschreiben. Ein Handwerker vermag jeden Schritt seiner Arbeit zu erklären, der Designer erlangte jedoch, dank der Kunst, die Freiheit, sein Werk unkommentiert und undokumentiert zu lassen. Es gibt jedoch Bereiche, in denen ausführliche Erklärungen unerlässlich sind: bei Design Manuals. Diese müssen alle Regeln und Richtlinien enthalten um die korrekte Umsetzung einer Corporate Identity, auch von wenig qualifizierten Mitarbeitern, sicherzustellen.

Und wenn diese Regeln gut durchdacht sind, können sie das Bestehen eines Erscheinungsbildes für Jahrzehnte ermöglichen. Als Beispiel soll das, 1956 von Paul Rand entwickelte, IBM-Erscheinungsbild und das dazugehörige Logo dienen. Durch einen modularen Aufbau, der es ermöglicht das Logo den unterschiedlichsten Situationen anzupassen, wurde ein zeitloses Markenzeichen geschaffen, das nach 50 Jahren immer noch eingesetzt wird (vgl. IBM, 2004) Die Mutationsmöglichkeiten des Logos sind im Design Manual exakt vorgeschrieben, so dass sie auch einem Computerprogramm als Regelwerk eingegeben werden könnten, um jedes angefertigte Schriftstück auf Regelkonformität zu überprüfen oder sogar um automatisch Designstücke herzustellen. Auch wenn sich ein Design Manual wahrscheinlich nie vollständig als ein von einem Computer verarbeitbares Regelwerk umsetzen lassen wird, so sind zumindest Teilaspekte als Algorithmen programmierbar.

Sobald es eine exakte Beschreibung eines Handlungsablaufes gibt, kann es auch ein Computerprogramm geben, das die beschriebene Aufgabe ausführen kann. Dazu muss sich der Designer allerdings Gedanken über die Struktur dieses Handlungsablaufes machen, einen entsprechenden Algorithmus konstruieren und diesen dann in ein ausführbares Programm verwandeln, das dann das gewünschte Ergebnis zurückliefert. Auch hierfür kann Paul Rand als Vorbild betrachtet werden, da auch er während des Designprozesses stets mehr geschrieben als gezeichnet hat (vgl. Hefland 2001,147f): So fertigte er für seine Kunden stets detaillierte Berichte über seinen Arbeitsprozess und dessen Fortschritte an. Seine Fähigkeit Ideen verbal auszudrücken, machen ihn zu einem Musterbeispiel für einen generativ denkenden Designer. Laszlo Moholy-Nagy, ein Pionier der Typografie und Professor am Weimarer Bauhaus schrieb über Rand: *"an idealist and a realist using the language of the poet and the businessman. He thinks in terms of need and function. He is able to analyze his problems, but his fantasy is boundless."* (Moholy-Nagy 1933)

Um nun zu einer exakten Definition von generativem Design zu gelangen, werden die Definitionen von „Arte Programmata“ und „Op Art“, unter Zuhilfename des Begriffs „Algorithmus“, verwendet und um einige für den Designkontext wichtige Aspekte erweitert. Was dabei von den kunsttheoretischen Definitionen übernommen wird:

* Ein formales Regelwerk, das befolgt wird um ein Resultat zu erzielen.
* Die Verwendung von Algorithmen um die Regeln zu definieren.
* Die „Symmetrie“ zwischen den Regeln und dem Werk.
* Die Freiheit des Designers die Regeln nicht als Einschränkungen sondern als Entscheidungshilfe zu sehen.

Einschränkungen, die vorgenommen werden:

* Es wird eine vorgefertigte Software benötigt, die einfachste Bedienung bei größtmöglichem Funktionsumfang bieten muss.
* Wir befinden uns nicht im Kunstumfeld und müssen uns deswegen auf reproduzierbare Ergebnisse verlassen können. D.h., bei einer Eingabe von definierten Ausgangswerten muss immer das selbe Ergebnis zurückgeliefert werden.
* Die Kontrollierbarkeit jedes Programmaspektes soll gewähreistet werden um die Beeinflussbarkeit jedes einzelnen Objektes zu ermöglichen.
* Es sollen keine Texturen, importierte 3D Modelle oder vorproduzierte Medien verwendet werden.
* Die Objektpalette soll sich auf Grundformen beschränken und Komplexität soll durch Anordnung dieser erreicht werden.

Die Punkte, die als Einschränkungen angeführt sind, gelten nur in Bezug auf den beiliegenden Bildband und stellen keine allgemein akzeptierten Prinzipien dar. Da nähere Ausführungen zu Thema Farbe, 3D Animation und Programmierung den Rahmen gesprengt hätten und für die Produktion des Bildbandes nicht relevant waren.

Ein wichtiger Punkt in Bezug auf die benötigten formalen Regelwerke ist die Feststellung, dass sich generatives Design auf einer anderen Stufe abspielt, als das Klassische. Der Designer denkt nicht mehr darüber nach, wie er einzelne Objekte manuell positionieren soll, sondern überlegt, welche Befehle und Algorithmen ein Objekt generieren können, das seinen Vorstellungen entspricht.

Es ist daher angebracht von „Meta-Design“ zu sprechen. Auf dieser höheren Ebene sind ganz andere Strategien notwendig um Design zu erzeugen. Der gewohnte Vorgang, eine leere Arbeitsfläche mit den benötigten Objekten (Bilder, Texte, Ornamente, usw.) zu füllen und diese dann so lang zu manipulieren bis ein ansprechendes Ergebnis erreicht ist, ist bei der Programmierung eines generativen Designs so nicht anwendbar, was auch den Vergleich mit nicht-generativem Design verkompliziert.

Der Designer ist demnach gezwungen, seine Vorstellungen in formal eindeutige Beschreibungen zu übersetzen und die Umsetzung des entstandenen Regelwerkes anschließend der Maschine zu überlassen.

Die in dieser Arbeit propagierte Definition des „generativen Designs“ ist stark auf die Form fixiert, was durch die selbstaufgelegte Beschränkung auf Grundformen und den Verzicht auf importierte (d.h. vorgefertigte) Daten zum Ausdruck gebracht wird. Dies soll jede Ablenkung durch nicht-generative Elemente (Bilder als Texturen oder andere importierte Objekte) verhindern.

Des Weiteren werden sich die generierten Beispiele auf Graustufen beschränken, da Farbe eine weitere Ablenkung von der Form sein kann.

Unsere Definition lautet demnach:

> *Generatives Design ist das Ergebnis einer, mit Hilfe von Algorithmen auf einer Meta-Ebene definierten **Handlungsanweisung**, welche von einem bestimmten Computerprogramm in beliebig viele nur aus Grundformen aufgebaute Designstücke umgesetzt werden kann.* ¬ 24

Diese Definition beachtend, werden nun Methoden analysiert, welche die Erstellung von generativem Design erlauben sowie Werkzeuge gewählt, die die gestellten –und noch zu stellenden– Anforderungen bestmöglich erfüllen.

1.2. Die Ansprüche an generatives Design

John Maeda, einer der Pioniere auf dem Gebiet des generativen Designs, stellt in seinem Buch „Design by Numbers“ die von ihm entwickelte Programmiersprache DBN (Design by Numbers) vor. Diese Sprache erlaubt über einen textbasierten Editor das Zeichnen von Punkten und Linien und stellt zudem einfachste Funktionen bereit, um diese Objekte nach definierten Regeln automatisch zeichnen zu lassen. Anhand dieser Programmiersprache führt Maeda, der die Professur für Media Arts & Sciences am MIT Media Lab innehat, den Leser in die grundlegenden Konzepte von computergeneriertem Design ein.

Dabei sieht Maeda den Computer als Werkzeug um Ideen zu verwirklichen, die mit analogen Werkzeugen wie z.B. einem Pinsel, nicht umsetzbar wären. Im Zuge seiner Ausführungen hebt er ganz klar hervor, dass es für ihn zwei verschiede Arten von digitalen Werkzeugen gibt: Solche, die analoge Methoden nur simulieren und solche, die vollkommen neue Möglichkeiten eröffnen.

> *„Painting with a mouse on the computer screen has a high entertainment value, but […] drawing a stroke with a pen is no different from drawing a stroke with a mouse. The real challenge is to discover the intrinsic properties of the new medium and to find out how the stroke you "draw" via computation is one you could never draw, or even imagine, without computation." (Maeda, 2001, 175)*

Die wahre Herausforderung bei der Benutzung von digitalen Designwerkzeugen ist es also, jene Möglichkeiten des neuen Mediums zu nutzen, die über die Möglichkeiten der analogen Werkzeuge hinausgehen. Da Computer sehr viele Befehle sehr schnell ausführen können schlussfolgert Maeda, dass „computation“ (d.h. Berechnung, Errechnung) die natürlichste Methode ist um Design mit dem Computer zu erzeugen:

„Drawing by hand, using pencil on paper, is undisputedly the most natural means for visual expression.When moving on to the world of digital expression, however, the most natural means is not pencil and paper but, rather, computation."

(Maeda, 2001, 251)

Dabei geht er sogar noch weiter indem er schreibt dass die Kombination von traditionellen Kunstformen mit der Computertechnologie keine wahre digitale Kunst sein kann, sondern nur eine digital-verstärkte, aber noch immer analoge Kunst. *„True digital art embodies the core characteristics of the digital medium, which cannot be replicated in any other"* (Maeda, 2001, 251) Das Ziel von „Design by Numbers" ist es, mit Hilfe von DBN ein grundlegendes Verständnis der Formgenerierung mit einer Programmiersprache zu vermitteln. Dabei ist für Maeda der Prozess des Programmierens und die Eleganz des Programmcodes ebenso wichtig, wie das erreichte Ergebnis.

Diesen Ansatz soll auch der in diesem Werk verwendete Begriff "generatives Design" verfolgen. Da sich die Methoden von generativem Design stark von denen des klassischen Designs (welches analoge Techniken nur mit digitalen Werkzeugen simuliert) unterscheiden, wird der Vergleich der zwei Welten auf der Ebene des Funktionsumfanges der verfügbaren Werkzeuge stattfinden . Welche Möglichkeiten stehen dem Designer jeweils zur Verfügung und auf welche Funktionen muss dieser eventuell verzichten?

Da die Firma Adobe plattformübergreifend eine Rolle als Marktführer (vgl. Cnet, 2005) eingenommen hat, dient deren Software, insbesondere Photoshop CS als pixelbasiertes Bildbearbeitungsprogramm und Adobe Illustrator CS als vektorbasiertes Illustrationsprogramm, als Vergleichsgrundlage. Zur Einführung in Illustrator und Photoshop empfehlen sich die „Photoshop Cs Bible" (McClelland 2004) und „Classroom in a Book: Adobe Illustrator CS" (Adobe 2003).

Um die Leistungsfähigkeit von Software für die Erstellung von generativem Design zu messen, soll die Betrachtung der Werkzeuge in 5 Themengebiete gegliedert werden. Dies ermöglicht die Betrachtung von Gemeinsamkeiten und Unterschieden zwischen generativem- und klassischem Design:

* Unterstützte Ausgabemedien und die erreichte Qualität
* Der Funktionsumfang der Werkzeuge
* Die Benutzbarkeit der Werkzeuge
* Nachträgliche Veränderbarkeit eines erstellten Designs
* Erweiterbarkeit der Werkzeuge

Das Wichtigste hier ist, dass das verwendete Tool mehr als nur eine Aufgabe erfüllen muss. Eine Software, die nicht modifizier- und erweiterbar ist und daher nur ein Design (in Variationen) liefert oder nur ganz bestimmte Formen darstellen kann, ist für die in diesem Werk propagierten Zwecke nicht geeignet.

Es gibt viele Beispiele für „Spezialdesigngeneratoren“, wie die „n_Gen Design Machine“, programmiert von Move Design (www.movedesign.com). Diese erlaubt die Eingabe einer Überschrift, Unterüberschrift und von Fließtext und generiert daraus ein Designstück (Flyer, Poster, Webseite oder CD Hülle) nach einem vorgegebenen Design-Template (Abb. 3-6).

Abb. 3

Abb. 4

Abb. 5

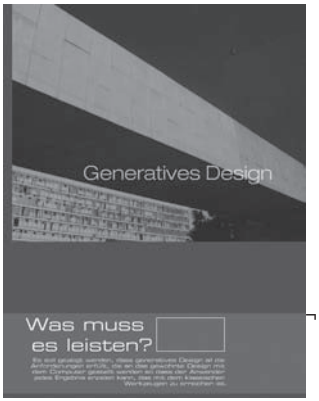


Abb. 6

Allerdings hat der Benutzer keine anderen Einflussmöglichkeiten, außer sich für eine Stilvorgabe (modern, klassisch, technisch, formalistisch…) zu entscheiden und sich daraus ein mehr oder weniger zufälliges Design nach dem anderen generieren zu lassen bis ein akzeptables Resultat erzielt wird. Aus diesem Grund scheidet dieser und viele ähnliche Template-basierte Ansätze von vornherein aus.

Generatives Design soll aber auch nicht alle Aufgaben des klassischen Designs übernehmen. Vielmehr muss herauskristallisiert werden, in welchen Fällen eine Entscheidung zugunsten generativer Ansätze von Vorteil ist und in welchen Fällen dies eine Verkomplizierung des Designprozesses mit sich bringt.

1.2.1. Unterstützte Ausgabemedien und die erreichte Qualität

Die erste Qualität muss das Ausgabemedium selbst sein. Die Ausgabe auf Papier stellt andere Ansprüche als die Wiedergabe auf einem Bildschirm. Die verwendete Software muss Schnittstellen besitzen um ein im Computer erzeugtes Design in hoher Qualität auf Papier bringen zu können. Für eine Publikation im Web muss ein Standard existieren, der die Darstellung der erzeugten Inhalte, auf so vielen Plattformen wie möglich, garantiert.

Photoshop und Illustrator sind auf den Printbereich spezialisiert und bringen daher alle Voraussetzungen mit um Motive in beliebiger Größe erstellen und bearbeiten zu können. Auch sonst werden die Adobe Produkte allen Anforderungen einer hochqualitativen Ausgabe wie Farbseparierungen, Rasterung, Schnitt- und Passmarken gerecht.

Für die Darstellung im Web stehen hier das JPG und GIF Format zur Verfügung, Formate wie SVG und PNG lassen sich exportieren, was eine universelle Darstellbarkeit garantiert.

In diesem Punkt können die generativen Tools grundsätzlich nicht mithalten, da diese ihren Fokus ganz klar auf die Ausgabe über den Bildschirm haben und deshalb nur selten Funktionen für den Export von hochauflösenden Bilddaten besitzen. Auch wenn vektor-basierte Tools wie Macromedia Flash den Export von auflösungsunabhängigen Pfaden, die in beliebiger Größe gedruckt werden können, unterstützen, bieten generativen Tools diese Funktion nicht standardmäßig an. Dies bedeutet, dass für das Finishing von generativen Designs praktisch immer auf ein Bildbearbeitungsprogramm zurückgegriffen werden muss.

1.2.2. Der Funktionsumfang der Werkzeuge

Die zweite Anforderung an ein Designwerkzeug ist, dass es Funktionen zur Verfügung stellt um Objekte zu erstellen, Texte zu platzieren und um Pfade zu zeichnen (z.B. Bezier Funktion). Photoshop und Illustrator bieten eine Fülle an Funktionen, die die Erstellung und Manipulierung von Objekten ermöglichen.

Alle in Kapitel 1.3 vorgestellten generativen Tools erfüllen ebenso die Voraussetzungen, d.h. sie besitzen die oben aufgeführten Funktionen, die je nach Spezialisierung der Software unterschiedlich ausgeprägt sind. So sind die Vektorfunktionen bei SWF und dem SVG Format sehr ausgeprägt, während sich z.B. processing (siehe Kapitel 1.3.3) gut für eine dreidimensionale Objektgenerierung eignet.

Vor allem der individuelle Stil des Designers, in Kombination mit dem angepeilten Designziel, ist ein Auswahlkriterium für das Werkzeug.

1.2.3. Die Benutzbarkeit der Werkzeuge

Der dritte und wohl wichtigste Punkt ist die Bestimmbarkeit des gewünschten Ergebnisses und das Maß an Kontrolle die über das Programm ausgeübt werden kann. Der Designer muss im Stande sein, genau zu definieren, wie und wo Elemente zu platzieren sind und wie diese farblich und strukturell beschaffen sein müssen. Ein Programm, das es dem Designer nicht ermöglicht jedes Element seines Designs zu beeinflussen, sei es durch Mängel im Interface oder wegen einer Fixierung auf unveränderbare Templates, ist ungeeignet, da es die Freiheit des Designers einschränken würde.

Grundsätzlich ist das Handling der generativen Tools viel schwieriger zu erlernen als der Umgang mit Photoshop, da die meisten generativen Funktionen Programmiertechniken erfordern. Allerdings bieten die generativen Tools auch Funktionen um den genauen Ablauf eines Programms zu überprüfen (Tracking) und Fehler aufzuspüren (Bugfixing).

Dadurch, dass jede gewünschte Eigenschaft explizit programmiert werden muss, ist der Designer beim generativen Ansatz jederzeit versichert, dass das Programm nur das macht, was auch vorgegeben ist. Dem ist allerdings entgegen zu halten, dass kein Photoshop User so viel über die innere Struktur des Programmes wissen muss um es beherrschen zu können, da es auf vollkommen verschiedenen Bedienprinzipien beruht.

1.2.4. Nachträgliche Veränderbarkeit eines erstellten Designs

Dieser Punkt prüft, inwieweit es möglich ist gewisse, im Designprozess getroffene Entscheidungen zu revidieren und Objekte nachträglich zu manipulieren.

Die UnDo Funktion ist ein einfacher Mechanismus um Veränderungen rückgängig zu machen. Dabei wird eine Liste erstellt auf der jeder verwendete Befehl notiert wird und bei Bedarf kann der Designer zu einem früheren Stadium seiner Arbeit zurückspringen. Allerdings geht so die ganze Arbeit, die nach dem fehlerhaften Schritt gemacht wurde, verloren.

Gewisse Programme, wie etwa „Maya“ von Alias (vgl. Alias 2005) verfügen über viel ausgeklügeltere Funktionen, mit der jeder Parameter jedes gemachten Arbeitsschrittes jederzeit verändert und sogar gelöscht und verschoben werden kann ohne die später erteilten Befehle zu löschen.

Generatives Design muss im Optimalfall 100% variabel sein so dass jederzeit jedes Element beeinflussbar bleibt. Dies wird gefordert, um leicht Variationen eines Designs erzeugen zu können. Photoshop legt einzelne Ebenen an, die nachträglich manipuliert werden können, genauso wie es Illustrator erlaubt, jeden einzelnen Punkt und jedes Objekt, bzw. Gruppen von Objekten zu manipulieren.

Die generativen Tools gehen hier viel weiter. Sie können beliebige Werte (Farben von Objekten, Transformationen) als Variablen definieren, die jederzeit beeinflusst werden können. So kann z.B. eine einzige Variable die Größe eines Objekts und gleichzeitig die Helligkeitswerte von 5 anderen Objekten kontrollieren. In diesem Punkt sind die generativen Ansätze den klassischen Tools weit überlegen.

1.2.5. Erweiterbarkeit der Werkzeuge

Dieser letzte Punkt beschreibt die Erweiterbarkeit eines Programmes. D.h., die Integration von neuen Funktionen durch den Anwender, um so Ergebnisse erreichen zu können, die dem Programm ursprünglich nicht möglich gewesen wären. Plugins sind eine Möglichkeit um dies zu gewährleisten, aber es gibt auch Open Source Software die mit den dazu nötigen Programmierkenntnissen zu 100% personalisiert werden kann. Dieser Punkt wird immer dann sehr wichtig, wenn bei einem Programm eine dringend benötigte Funktion abgeht.

Während die Adobe Produktpalette eine Erweiterung nur über meist sehr teuere Plugins von Drittherstellern erlaubt, bieten die generativen Ansätze gewöhnlich mehr Möglichkeiten um benötigte Funktionen zu integrieren. Die in Kapitel 1.4 vorgestellten Tools besitzen alle Schnittstellen um über diverse Protokolle (TCP, UDP, NetSend, OSC, MIDI, DMX, RSh, usw.) mit anderen Programmen Daten auszutauschen. So können Bild- oder Audioanalysedaten in einem geeigneten Programm berechnet werden und dann über eine Schnittstelle an ein generatives Tool übermittelt werden, wo diese Daten dann auf Objekte angewendet werden können.

Auch die Integration von Daten aus Online Quellen (RSS Newsfeeds, Wetterdaten und anderen Datenbanken) und direkt an das System angeschlossener Hardware (Sensoren, Interfaces, usw…) ist möglich. Also liegt auch hier der Vorteil beim generativen Ansatz

1.3. Etablierte Ansätze im generativen Design

Die Frage, die es nun zu klären gilt ist, welche Werkzeuge uns zur Verfügung stehen und wie diese benutzt werden um die im Werkteil gezeigten Objekte generieren zu können. Deshalb sollen in diesem Kapitel einige etablierte Ansätze besprochen werden, die zur Erzeugung von generativem Design Verwendung finden können. Wie schon festgestellt wurde muss nicht gezeigt werden, dass sie die klassischen Designtools vollständig von generativen Tools ersetzt werden müssen. Die einfache Tatsache, dass der generative Ansatz für gewisse Aufgaben effektiver ist, reicht vollkommen aus, um den Einsatz dieser Methoden in vielen Situationen zu rechtfertigen. Zum Beispiel: die im Werkteil gezeigten Grafiken sind, vor allem in diesem Variantenreichtum und in dieser Geschwindigkeit, nicht mit Photoshop, Illustrator und ähnlichen Programmen erzeugbar. Es muss noch vorweggenommen werden, dass der generative Einsatz nicht ohne eine Syntax auskommt, die bestimmt wie das Programm Anweisungen entgegennehmen kann. Die am weitesten verbreitete Form dieser Syntax ist die textbasierte. Anweisungen werden Zeile um Zeile eingetippt, in ein ausführbares Programm übersetzt und dann ausgeführt.

Das Design wird also erst in jenem Moment erstellt, in dem das Programm aufgerufen wird. Vorher ist das Design zwar schon vorhanden, aber nur implizit, als eine Menge an Regeln, die erst angewendet werden müssen um ein Bild zu erzeugen. Der Umgang mit einer generativen Programmierumgebung erfordert demnach einiges Umdenken. Der Designer produziert sein Werk nicht mehr selbstständig, er schreibt ein kleines Programm, das diese Arbeit für ihn erledigt. In diesem Programm schlummert dann die Funktion, das gewünschte Design (oder beliebige Variationen desselben) zu generieren.

Im Zuge der Recherchen zeigte es sich, dass vor allem 4 Technologien weite Verbreitung finden und diese sollen nun vorgestellt werden und auf ihre Eignung, generatives Design zu erzeugen, geprüft werden.

1.3.1. Flash und Action Script

Macromedia Flash ist eine proprietäre, integrierte Entwicklungsumgebung zur Erzeugung von Animationen in einem auf Vektorgrafiken basierenden Grafik- und Animationsformat (SWF).

Die 1992, aus dem Zusammenschluss der Firmen MacroMind und Authorware, hervorgegangene Firma Macromedia kaufte sich 1996 das von FutureWave entwickelte Animationsprogramm FutureSplash-Animator und benannte dieses in „Macromedia Flash 1.0“ um. Dessen Funktionsumfang war allerdings noch stark begrenzt. 1999 stellte die Integration der Programmiersprache ActionScript (vgl. Lott 2003) in Flash Version 4 erste Kontrollstrukturen, wie if-Anweisungen und Schleifen zur Verfügung, mit deren Hilfe die Entwicklung von wesentlich komplexeren Applikationen (Trainingssoftware, Online Spiele und auch schon einfaches generatives Design) ermöglicht wurde.

ActionScript wurde seitdem stets erweitert und an Programmiersprachen wie JavaScript angepasst, was den Umstieg für Entwickler, die mit anderen Programmiersprachen zu tun hatten, wesentlich erleichtert. Im März 2002 erschien die MX-Version mit neuer Zeichnen-API und wesentlich erweiterten ActionScript-Programmierfunktionen, die das Erstellen von dynamischen Formen ermöglichen. Zusätzlich wurde das Objekt- und Ereignismodell erweitert. Mit ActionScript 2.0, das mit Flash MX 2004 im Oktober 2003 eingeführt wurde, schaffte Flash endlich den Sprung zu einer wirklich umfassenden Programmierbarkeit, was es zu einem guten Kandidaten zur Erzeugung generativer Formen macht.

Macromedia beschreibt sein Tool folgendermaßen:

> *"Macromedia Flash MX 2004 allows designers and developers to integrate video, text, audio, and graphics into immersive, rich experiences that deliver superior results for interactive marketing and presentations, e-learning, and application user interfaces. Flash is the world's most pervasive software platform, used by over one million professionals and reaching more than 97% of Internet-enabled desktops worldwide, as well as a wide range of devices."*
> *(Macromedia 2005a)*

Durch die weite Verbreitung des SWF Plugins, das laut einer von Macromedia in Auftrag gegebenen Studie der NPD Group, auf 97% (vgl. Macromedia 2005b) der Rechner weltweit installiert ist, wird sichergestellt, dass praktisch jeder Internetnutzer die mit Flash erstellten Inhalte betrachten kann.

Die große User Gemeinde (laut Macromedia über eine Million Anwender) ist ein weiterer großer Vorteil von Flash. Die Tatsache, dass sich Flash mit seiner Bedienstrategie sehr an gewohnten Windows-Konventionen orientiert ist mit Sicherheit ein wichtiger Grund für die weite Verbreitung, da gewohnte Strategien zur Benutzung von Designsoftware zum Teil beibehalten werden können.

Dieser Ansatz funktioniert allerdings nur bei der grafischen Erstellung von Formen und Objekten, nicht aber um Anweisungen zu deren Verhalten und Positionierung zu geben. Dafür muss auf Action Script zurückgegriffen werden, was den Designer zwingt, seine Anweisungen textbasiert zu programmieren.

Um ein Beispiel zu geben, wie ein einfaches Action Script Programm aussieht:

```
_root.createTextField("mytext",1,100,100,300,100);
mytext.text = "Hello World!";
```

Allerdings richtet sich Flash an zwei vollkommen unterschiedliche Zielgruppen gleichzeitig: Designer und Programmierer. Das Programm bietet sogar zwei vollständig angepasste Interface-Konfigurationen an, von denen eine (Designer oder Developer) beim ersten Aufruf des Programmes ausgewählt werden muss. Dies zeigt, dass in Agenturen nach wie vor eine große Distanz zwischen den Rollenmustern von Programmierern ("technisch versiert aber unkreativ") und Designern ("kreative Köpfe, die von der Technik wenig Ahnung haben") besteht. Dies kann daran liegen, dass sich die Designer nur schwer mit der textbasierten Programmierung anfreunden können, da diese im absoluten Gegensatz zu deren bildhafter Denkweise steht.

1.3.2. SVG und SMIL

SVG (Scalable Vector Graphics) ist ein freies Konkurrenzformat zu SWF. Es wird vorwiegend von der Firma Adobe vorangetrieben und ist eine Sprache zur Beschreibung zweidimensionaler Vektorgrafiken in XML. SVG unterstützt mittels SMIL (Synchronized Multimedia Integration Language) auch Animationen. Es wurde im September 2001 zur W3C-Empfehlung (World Wide Web Consortium, www.w3.org) erhoben und wird bereits von einigen Browsern, wie Mozilla Firefox, der von www.mozilla.org/projects/svg heruntergeladen werden kann, nativ unterstützt. Für andere Browser ist dafür ein Plugin notwendig, wie z.B. der SVG Viewer von Adobe. Macromedia gibt auf seiner Seite an, dass das SVG Plugin nur auf 13,5% aller Computer installiert ist, was eine Verbreitung von SVG-Grafiken über das Web (noch) unattraktiv macht.

Alle grafischen Objekte in SVG bauen auf einfachen grafischen Grundelementen auf. Komplexere Objekte sind dabei aus mehreren einfachen Objekten zusammengesetzt, wobei der Pfad das eigentliche Grundelement in SVG ist. Aus ihm können alle anderen Objekte (Kreise, Rechtecke, Polygone etc.) aufgebaut werden. Da das aber teilweise sehr umständlich ist, hat das Team das die Spezifikationen ausarbeitet, diese häufigen Formen mit jeweils eigenen Beschreibungen versehen.

Das W3C beschreibt SVG so:

> *"SVG is a language for describing two-dimensional graphics in XML. SVG allows for three types of graphic objects: vector graphic shapes (e.g., paths consisting of straight lines and curves), images and text. Graphical objects can be grouped, styled, transformed and composited into previously rendered objects. The feature set includes nested transformations, clipping paths, alpha masks, filter effects and template objects. SVG drawings can be interactive and dynamic. Animations can be defined and triggered either declaratively (i.e., by embedding SVG animation elements in SVG content) or via scripting. Sophisticated applications of SVG are possible by use of a supplemental scripting language which accesses SVG Document Object Model (DOM), which provides complete access to all elements, attributes and properties."*
>
> *(SVG Working Group 2003)*

Wie das obrige Zitat belegt, hat SVG in der aktuellen Spezifikation 1.0 einen großen Nachteil: die Beschränkung auf 2D Grafik. Allerdings ließe sich auch in zwei Dimensionen äußerst komplexes und optisch ansprechendes generatives Design erzeugen. Da SVG jedoch nur eine Beschreibungssprache ist, für die es noch keine, mit Flash und anderen generativen Tools vergleichbare, Entwicklungsumgebung gibt, fällt ein Einstieg in die sehr formelle Welt von SVG nicht leicht. Auch wenn es Vektorgrafik-Programme wie Sodipodi und Inkscape (beide Freeware) gibt, die SVG als ihr natives Datenformat verwenden und mit der Bedienung von Adobe Illustrator vergleichbar sind, fehlt ihnen jede Möglichkeit, komplexere Verhaltensweisen zu programmieren, weswegen diese Tools und damit SVG, für die in diesem Werk verfolgten Ziele ausscheiden.

1.3.3. Java und processing

Java ist eine objektorientierte, plattformunabhängige Programmiersprache, die von Sun Microsystems entwickelt wird. Üblicherweise benötigen Java-Programme zur Ausführung eine spezielle Umgebung (Java Virtual Machine) um lauffähig zu sein (vgl. Gosling 2000). Der Vorteil ist, dass nur diese Umgebung an verschiedene Computer und Betriebssysteme angepasst werden muss. Sobald dies geschehen ist laufen auf der Plattform alle Java-Programme ohne Anpassungsarbeiten.

Die Urversion von Java wurde 1992 im Auftrag von Sun entwickelt. Das ursprüngliche Ziel war nicht lediglich die Entwicklung einer weiteren Programmiersprache, sondern die Entwicklung einer vollständigen Betriebssystemumgebung, inklusive virtueller CPU, für unterschiedlichste Einsatzzwecke. Man entschied sich für den Namen Java nach dem Namen einer starken Kaffee-Sorte, die speziell für Espresso Verwendung findet (Java-Bohne) und von den Entwicklern bevorzugt getrunken wurde.

Processing ist eine Erweiterung von Java mit einer stark vereinfachten und verkürzten Syntax und von den Initiatoren des Projekts (Ben Fry und Casey Reas) als eine Programmierumgebung zum Erlernen der Grundprinzipien des Programmierens im Kontext der elektronischen Künste konzipiert. Processing wird vom MIT Media Lab (in der von John Maeda geleiteten Gruppe Aesthetics and computation) in den USA und am italienischen Institut für interaktives Design IVREA entwickelt.

Es ist ein offenes, sich schnell weiterentwickelndes Projekt, das auf der Projekthomepage (proce55ing.org) so beschrieben wird:

> *"Processing is a programming language and environment built for the electronic arts and visual design communities. It is created to teach fundamentals of computer programming within a visual context and to serve as a software sketchbook. It is used by students, artists, designers, architects, and researchers for learning, prototyping, and production."*
> *(Fry | Reas 2004)*

Durch seine Spezialisierung auf den Kunst- und Designkontext und die breite Palette an Erweiterungsmöglichkeiten, die sich durch die zugrunde liegende Java Technologie ergeben, machen Processing zu einem ausgezeichneten Werkzeug um damit generatives Design zu erzeugen. Es existiert eine große Community (04.05.05: 2583 registrierte User), die in diversen Foren (fast 4000 Themen mit insgesamt über 14.000 Postings) sehr guten Support (processing.org/discourse) und Hilfe bei Fragen bietet. Zudem ist es eine kostenlose Technologie, die noch dazu auf jeder Plattform und jedem Betriebssystem mit einer Java Virtual Machine ausgeführt werden kann.

Allerdings kommt der Anwender auch bei diesem Ansatz nicht um das Erlernen einer textbasierten Programmiersprache herum.

1.4. Ein grafischer Ansatz

Dieses Kapitel bespricht die, sich zur Zeit sehr schnell verbreitenden, visuellen Multimedia-Programmiersprachen (von nun an „grafische Programmierumgebungen“ genannt), welche eine gänzlich andere Strategie verfolgen als die 3 besprochenen Ansätze mit ihrer textbasierten Syntax:

> *“A visual language manipulates visual information or supports visual interaction, or allows programming with visual expressions.”*
>
> *(Golin 1990, 141ff)*

Diese, von E. J. Golin in seinem Artikel “The Specification of Visual Language Syntax” gemachte Aussage sagt also, dass eine visuelle Sprache visuelle Informationen manipuliert, eine visuelle Interaktion unterstützt oder programmieren über visuelle Ausdrücke ermöglicht.

B.A. Myers hingegen definiert eine grafische Programmiersprache als ein Programm, das in zwei oder mehr Dimensionen geschrieben wird: *“Any system where the user writes a program using two or more dimensions.”* (Myers 1990, 97ff). Damit spielt er auf die Positionierung von Funktionsblöcken auf den Arbeitsflächen von visuellen Entwicklungsumgebungen an.

Eine noch präzisere Definition liefern uns M. Burnett, A. Goldberg und T. Lewis in ihrem Buch ”Visual Object-Oriented Programming: Concepts and Environments”:

> *“A visual language is a set of spatial arrangements of text-graphic symbols with a semantic interpretation that is used in carrying out communication actions in the world.”*
>
> *(Burnett 1994)*

Eine visuelle Sprache ist ein räumliches Arrangement von beschrifteten grafischen Symbolen, die semantisch interpretiert werden müssen, um innerhalb der Programmierumgebung miteinander kommunizieren zu können. Auf diese 3 Definitionen bezieht sich jede Erwähnung einer “grafischen Programmiersprache” in diesem Werk. Dieser Begriff ist nicht mit integrierten Entwicklungsumgebungen (IDEs) wie Microsoft Visual Studio oder Eclipse (www.eclipse.org) zu verwechseln, die grafische Oberflächen für textbasierte Programmiersprachen sind.

Alle grafischen Programmierumgebungen benutzen so genannte „Knoten“ (engl. „nodes“), um Funktionen der Programmiersprache darzustellen. Diese Funktionsblöcke, die nach Belieben auf einer Arbeitsfläche verteilt werden können, erfüllen die unterschiedlichsten Aufgaben, die, je nach verwendeter Software, unterschiedliche Anwendungsgebiete abdecken. Es gibt Spezialsoftware für Klangerzeugung und Audioanalyse, Prototyping Systeme für die Spieleindustrie, ein Tool, das sich ganz auf die Bildanalyse konzentriert und auch stark visuell orientierte Technologien, die sich für generatives Design eigenen könnten. Eine weitere Gemeinsamkeit dieser grafischen Programmierumgebungen ist, dass alle Knoten nach genau festgelegten Regeln miteinander verbunden werden können, was den Knoten erlaubt, untereinander Daten auszutauschen. Diese Daten können einfache Zahlenwerte, aber auch Media Streams (Audio|Video), 3D Modelle oder Sensordaten sein. Je nach Art des Knotens werden die Daten nach der Ankunft verarbeitet und in irgendeiner Form an einen oder mehrere verbundene Knoten weitergereicht.

Dies sind die beiden Grundeigenschaften von datenflussorientierten Programmierumgebungen: Verhaltensweisen werden programmiert indem Funktionsblöcke auf einer Arbeitsfläche arrangiert und verknüpft werden. Ein sehr grafischer Ansatz, da der Datenfluss zwischen den Funktionsblöcken sehr gut nachvollziehbar ist.

Zur Illustration der erklärten Konzepte sollen folgende zwei Screenshots dienen.

Links puredata | gem (Abb.7), rechts max | MSP (Abb. 8).

Eine genauere Vorstellung der beiden Softwarepakete folgt im Kapitel 1.5.

Abb. 7

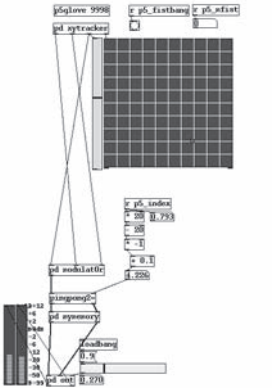
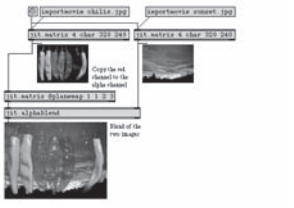


Abb. 8



Grafische Programmierumgebungen stellen, wie die drei zuvor besprochenen Ansätze, nur Werkzeuge dar, um dem Computer, in Grenzen, beizubringen wie er ein gewünschtes Design zu produzieren hat. Die erzeugten Grafiken können den Ergebnissen der anderen besprochenen Technologien zum verwechseln ähnlich sehen. Der unterschiedliche Ansatz macht sich nur auf der Ebene des Schaffenden bemerkbar, nicht aber auf der Ebene des generierten Bildes.

Der frappierend andere Ansatz gegenüber den textbasierten Programmierumgebungen ist der, dass Zusammenhänge zwischen Knoten auf einer visuellen Logik beruhen, die das Gehirn auf einer anderen, viel direkteren Ebene interpretieren kann. Während das Vokabular einer typografischen Programmiersprache ziemlich genau bekannt sein muss, um den Datenfluss im Programm oder auch nur die Verbindungen zwischen zwei Programmfunktionen herauslesen zu können, kann aus der grafischen Verknüpfung zwischen zwei Knoten einer grafischen Programmiersprache sofort gefolgert werden, dass diese Knoten in irgendeinem Verhältnis zueinander stehen. Dieses intuitive Verständnis erleichtert die Kontrolle des Datenflusses auch bei komplexeren Programmen mit vielen Knoten und Verknüpfungen wesentlich. Andrew Bregel, vom MIT Media Lab, schrieb 1996 in einer Abhandlung über eine vom MIT entwickelten grafischen Programmiersprache für Handheld-Computer:

> *„Using graphical representations of objects, you can more concretely show object orientation […], eliminate annoying syntax […] and better visualize the pathways that your program is following. Parallelism can also be made more explicit […] Graphical programming can also use metaphors from real life to make programming easier. […] Graphical programming also allows for easy sharing of programs. […] Another advantage is easy browsability. Looking at a picture of a program, a user might more easily be able to discern its meaning, rather than looking at a large textual program that is composed of many code files. […] Perhaps one of the best advantages is the use of visual cues in graphical languages. Connections between objects can be made more explicit through the design and graphical representation of the constructs."*
>
> *(Bregel 1996, 9f)*

Bei der grafischen Repräsentation kann die lästige Syntax eliminiert, und der Datenfluss im Programm besser visualisiert werden. Metaphern aus dem real-life machen das Programmieren einfacher und ermöglicht den leichten Austausch von Programmen zwischen Programmierern. Ein grafisch visualisiertes Programm lässt sich einfacher begreifen als ein langer Quellcode. Die visuellen Hilfen, wie Linien als Verbindungen zweier Funktionen, sind dabei wohl der größte Vorteil. So lassen sich die von Bregel aufgelisteten Vorteile von grafischen Programmierumgebungen zusammenfassen. Der nächste Absatz listet die von ihm definierten Nachteile auf:

> *"Some graphical languages are graphical to the core, which leads to frustration for sophisticated programmers who want to concisely express a statement that might be better represented using text. Screen real estate is also a limiting factor. This is called the "Deutsch Limit." "The problem with visual programming is that you can't have more than 50 visual primitives on the screen at the same time." […] In order for icons and graphics to be understandable they need to be big enough to see or have a textual label. Some languages also depict function calls by lines between clusters of graphics. If there are too many functions on a "page," the code becomes messy and hard to follow."*
> *(Bregel 1996, 10)*

Das „Deutsch Limit“ ist ein von Fred Lakin geprägter Begriff, der das Maximum an Symbolen bezeichnet, das sich in einer grafischen Programmierumgebung überschaubar darstellen lässt. Lakin und Peter Deutsch unterhielten sich über grafische Programmierumgebungen, als Deutsch plötzlich fragte:

> *“Well, this is all fine and well, but the problem with visual programming languages is that you can’t have more than 50 visual primitives on the screen at the same time. How are you going to write an operating system?”*
>
> *(Baeza-Yates, 2005)*

Das Problem mit grafischen Programmierumgebungen ist also, dass nicht mehr als 50 primitive visuelle Objekte auf einem Bildschirm Platz finden, was Deutsch zu der Frage veranlasste: „Wie soll man damit ein Betriebssystem programmieren?“ Diesen Vorteil der größeren Informationsdichte von textbasierten gegenüber grafischen Programmierumgebungen führt Bregel als Nachteile an. Ob es eine ähnliche Beschränkung wie das Deutsch Limit auch für textbasierte Programmierumgebungen gibt, bleibt allerdings offen.

Die meisten grafischen Programmierumgebungen erlauben tatsächlich keine Programmierung über eine textbasierte Syntax, was für viele fortgeschrittene Programmierer eine Einschränkung darstellt. Durch die Fokussierung der Zielgruppe dieses Werkes vor allem auf Nicht-Programmierer, sollten sich daraus aber keine Nachteile ergeben.

Dass bei sehr vielen Symbolen ein Programm sehr schnell unübersichtlich werden kann, ist jedoch ein ernstzunehmendes Problem, das sich allerdings durch eine Gruppierung von vielen Knoten und deren Verknüpfungen zu einem einzigen Symbol umgehen lässt.

Dies ist ein wichtiges Feature, das jede ernstzunehmende grafische Programmiersprache besitzen muss. Jede Gruppe von Knoten und deren Verknüpfungen muss auch als Modul verwendet werden können, das als einfacher Knoten mit definierten Ein- und Ausgängen, beliebig oft, in unterschiedlichen Situationen, wiederverwendet werden kann. Dieses Konzept der Module oder „Unterprogramme“ findet sich¬ selbstverständlich auch in modularen, textbasierten Programmiersprachen.

Der grafische Ansatz ist allerdings weit weniger abstrakt.

1.5. Zwei Beispiele für grafische Programmierumgebungen

In der Welt der grafischen Programmierumgebungen sind zwei Softwarepakete sehr weit verbreitet: Pure Data (pd) und Max|MSP. Beide wurden von Miller Puckette entwickelt, wobei Max ein kommerziell vertriebenes Tool ist, während pd Open Source ist. Beide werden nun vorgestellt.

1.5.1. Max|MSP & Jitter

Die ältere der beiden Softwarepakete war ursprünglich nur auf Apple Macs lauffähig, wurde aber kürzlich auf den PC portiert. Auf der Homepage des Herstellers (www.cycling74.com) wird Max so beschrieben:

> *"Max|MSP is a graphical environment for music, audio, and multimedia. In use worldwide for over fifteen years by performers, composers, artists, teachers, and students, Max|MSP is the way to make your computer do things that reflect your individual ideas and dreams." (Cycling '74 2004)*

Max|MSP stellt das Grundprogramm dar und Jitter ist eine Erweiterung dazu, die vor allem Funktionen zur Integration von Video Dateien und 3D Grafiken enthält. Vor allem zur Manipulation von Matrizen (zwei- oder mehrdimensionale Arrays), die den Zugriff auf jeden einzelnen Pixel eines Videosignals ermöglichen, ist Jitter bestens geeignet. Diese Matrizen können aber auch 3D Geometrie, Text, Tabellen, Partikelsysteme oder Audio Dateien enthalten. Jitter bietet Support für Nurbs-Geometrie, so dass Objektkanten statt Geraden auch Bezier-Kurven sein können. Ausgestattet mit ausgereiften mathematischen Funktionen zur Berechnung von Linearer Algebra, zellulären Automaten, Lindenmeyer Systemen, etc., bietet es viele Funktionen um generative Formen und Objekte zu erzeugen.

1.5.2. Pure Data & GEM

Pure Data ist die freie Konkurrenz zu Max|MSP und stammt vom selben Programmierer. Miller Puckette und sein Team wollten die Strategien, die bei der Programmierung von Max|MSP und anderer Software aus der selben "Familie" (Max|FTS, ISPW Max, jMax, u.a.) entwickelt wurden, offener gestalten und den Umgang mit verschiedensten Datentypen vereinfachen. Da pd von Anfang an als Open Source Software konzipiert war, gestaltet sich die Erweiterung um neue Objektklassen, bei ausreichender Programmiererfahrung, als relativ einfach, was der Grund ist warum der Kern von pd relativ schlank gehalten wurde und nur grundlegende Funktionen enthält. Zusätzliche Funktionen können als Plugins leicht hinzugefügt werden und dank der offenen Architektur gibt es sehr viele Programmierer, die selbst die ausgefallensten Objekte programmiert haben.

Eine der wichtigsten Ergänzungen zu pd ist GEM, das ähnliche Funktionen wie Jitter für Max|MSP enthält. Das GEM Projekt wird von Intel Research Council unterstützt und ist Teil des "The Global Visual Music" Projektes, initiiert unter anderem von Miller Puckette. Die Idee stammt von Mark Danks, der auch die erste Version geschriebn hat und zur Zeit wird GEM am Institut für Elektronische Musik und Akustik in Graz weiterentwickelt, auf deren Seiten folgendes über GEM geaschrieben wird:

> *"GEM is the Graphics Environment for Multimedia. It was originally written by Mark Danks to generate real-time computer graphics, especially for audio-visual compositions. Because GEM is a visual programming environment, users do not need any experience in traditional computer languages. GEM is a collection of externals which allow the user to create OpenGL graphics within Pd, a program for real-time audio processing by Miller Puckette (of Max fame).*

There are many different shapes and objects, including polygonal graphics, lighting, texture mapping, image processing, and camera motion. All of this is possible in real-time without any previous programming experience. Because GEM is an add-on library for Pd, users can combine audio and graphics, controlling one medium from another."
(Zmölnig, 2002)

Die Fähigkeit, hochqualitative 3D Grafiken zu erzeugen, eignet die Kombination pd und GEM auch für generatives Design. Vor allem für sehr spezielle Projekte die ungewöhnliche Funktionen benötigen, ist pd zu empfehlen.

Während der Recherchen zeigten sich keine wesentlichen Vor- oder Nachteile von einzelnen Paketen. Jedes ist für sich sehr leistungsfähig und bietet genügend Funktionen, um die kreative Freiheit für generatives Design zu garantieren.

Es dürfte also eine Geschmacks- und Budgetfrage sein, ob man sich für das vollkommen freie pd|GEM, das relativ teuere Max|Jitter ($850) oder eine vergleichbare grafische Programmierumgebung entscheidet.

{2} Generatives Design mit vvvv

Um nun weitere Eigenschaften von generativem Design mit grafischen Programmierumgebungen erörtern zu können, soll eine Programmierumgebung ausgewählt werden und diese näher betrachtet werden.

Zur Erstellung der Beispiele in dieser Arbeit wurde auf die Entwicklungsumgebung „vvvv“ der Firma meso zurückgegriffen, da diese stark visuell ausgerichtet ist und trotz des sehr überschaubaren Funktionsumfangs (etwas mehr als 300 Knoten) und des spartanischen Interfaces, alle benötigten Funktionen bereitstellt. Durch die direkte Integrierbarkeit von Macromedia Flash Animationen und anderen externen Datenquellen, sowie die Möglichkeit eines hochauflösenden Bildexports, ergeben sich weitere Vorteile im Designkontext. Zudem ist die Software für die nicht-kommerzielle Nutzung kostenlos und bietet dabei trotzdem einen guten Support.

Da somit alle definierten Anforderungen erfüllt werden, soll vvvv als Repräsentant der grafischen Programmierumgebungen, in Kapitel 2 näher betrachtet werden.

vvvv ist eine Eigenentwicklung der Firma meso und entstand als firmenintern klar wurde, dass die verfügbaren Softwareprogramme auf dem Markt nicht ausreichten um die aufwendigen Projekte zu realisieren, die meso damals plante. Leider gibt es dazu noch keine publizierten Schriften und das Produkt selbst befindet sich noch im Beta-Stadium. Auf der Webseite vvvv.meso.net findet der interessierte Leser aber eine große Menge an Informationen und auch das Programm selbst zum kostenlosen Download.

Anfangs noch textbasiert entwickelten sich viele Funktionen von vvvv (umgangssprachlich auch kurz v4 oder 4v genannt) mit der Zeit weiter und schließlich wurde der Applikation eine grafische Benutzeroberfläche gegeben und nach einigen weiteren internen Beta Versionen wurde entschieden, vvvv der Öffentlichkeit für nicht-kommerzielle Zwecke kostenlos zu Verfügung zu stellen. Von den Entwicklern, wird ihre Software so beschrieben:

> *"vvvv is a toolkit for real time video synthesis and connecting physical devices. We use the word "synthesis" because vvvv generates or "synthesizes" video through the usage of moving graphical objects."*
>
> *(meso 2005a)*

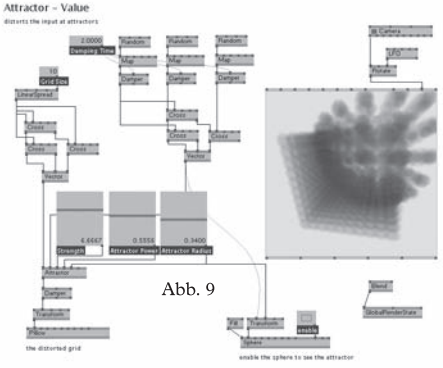


Abb. 9

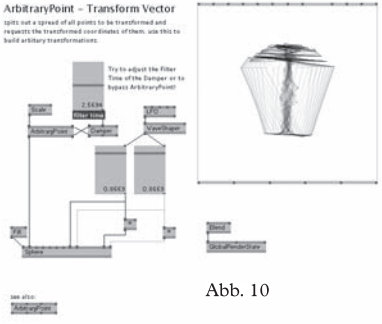


Abb. 10

Auf der Webpage findet sich auch eine Liste mit den Funktionen (“Core Features”) von vvvv, wobei nur die relevantesten Punkte hier besprochen werden:

* Sophisticated objects for high level animation and problem solving
* Simple handling of a multitude of logical objects (Spreads)
* Runtime graphical programming for easy prototyping and testing
* High-Speed, High-Quality, High-Resolution DirectX9.0 based rendering
* Runs on standard windows xp platforms. DirectX9.0 graphics card recommended.

[...] (vgl. meso 2005b)

Die große Palette an Funktionen und die einfache Handhabung von vielen Objekten gleichzeitig (siehe „Spreads“ Kapitel 2.3) sind in dieser Auflistung die wichtigsten Punkte um vvvv zur Erstellung von generativem Design einsetzen zu können. Punkt 3 beschreibt die unterbrechungsfreie Arbeitsweise von vvvv.¬ 54 „Runtime“ bedeutet, dass es keinen Unterschied zwischen einem Programmier- und einem Ausführungsmodus gibt. Jede Veränderung im Patch bewirkt sofort eine Veränderung im Verhalten da die mit vvvv erstellten Programme nicht erst in eine ausführbare Datei übersetzt werden müssen um lauffähig zu sein. Es gibt nur den einen Modus in dem das Programm sowohl läuft als auch verändert werden kann. Dies vereinfacht das Erstellen und Testen („prototyping & testing”) von Patches erheblich. Punkt 4 belegt das hochqualitative und auflösungsunabhängige Rendering, was auch die Produktion von großformatigen Designs ermöglicht. Der letzte Punkt stellt die grundsätzliche Benutzbarkeit von vvvv sicher, da es auf allen Standard-Rechnern mit Windows XP läuft und keine exotischen Anforderungen an die Hardware stellt.

2.1. Die zugrunde liegende Funktionsweise

Wie aus den Abbildungen 9 und 10 ersichtlich, besteht ein vvvv-Programm aus einzelnen, mit Namen versehenen Blöcken („Knoten“ genannt) und Verbindungen zwischen diesen. Auf der Oberseite des Knotens befinden sich alle Eingänge, auf der Unterseite alle Ausgänge. Jeder Knoten, von denen es bisher über 300 Stück gibt, bearbeitet die Daten mit denen er gefüttert wird auf eine vorgegebene Art und Weise und reicht das Ergebnis über seine Ausgänge weiter an andere Knoten.

Es gibt sehr einfache Knoten, wie etwa die Addition von 2 Werten (2 Eingänge, ein Ausgang) aber auch komplexe Knoten mit dutzenden Ein- und Ausgängen. Die Ein- und Ausgänge werden als „Pins“ bezeichnet. Durch diese Pins fließen alle Datenströme und auch wenn einzelne Knoten, an sich betrachtet, nicht besonders mächtig sind, lassen sich durch eine kreative Kombination dieser Knoten äußerst komplexe Vorgänge berechnen und darstellen.

Einige Beispiele, für Anwendungen für die vvvv eingesetzt werden kann:

* Analyse von Daten (Bilderkennung, Audioanalyse, Sensoren…)
* Rendering und Shading von importieren oder live erstellten 3d Objekten
* Kontrolle von angeschlossenen Geräten wie Motoren, Lichtern und auch anderen Computern
* Erzeugung von komplexen Formen mit Hilfe von mathematischen Algorithmen

2.2. Die Generierung von primitiven Objekten

Da der Schwerpunkt dieses Werks auf der Generierung von Design liegt, wird hier nur auf den letzten Punkt eingegangen. vvvv soll nun also dazu verwendet werden, große Objektmengen in regelbasierte Strukturen zu verwandeln. Doch bevor diese Erzeugung von komplexen Formen in Angriff genommen werden kann, muss erst geklärt werden, woraus diese Gebilde überhaupt bestehen sollen.

vvvv bietet ein breites Spektrum an Funktionen zur Erzeugung von primitiven Objekten an, welche sich in die Kategorien „Punkt“, „Linie“, „Fläche“ und „Körper“ einteilen lassen. Es ist hier unabdingbar, etwas über die Dimensionalität zu sprechen, insbesondere in Bezug auf die Positionierung, Darstellung und der eigentlichen Konstruktion eines Elements.

2.2.1. Punkte

Ein Punkt ist ein 0-dimensionales mathematisches objekt, das in einem n-dimensionalen Raum mit n Koordinaten definiert werden kann (“A point 0-dimensional mathematical object, which can be specified in n-dimensional space using n coordinates.” (Wolfram Research 2005a)). Im Allgemeinen leichter verständlich scheint Euklids Definition: *„Was keine Teile hat, ist ein Punkt.* “ (Euklid 2003).

Sobald jedoch ein definiertes Ausgabemedium vorhanden ist (Papier, Bildschirm), ist ein Punkt einfach nur das kleinste darstellbare Objekt dieses Ausgabemediums. Also ein Punkt auf dem Papier oder ein Pixel auf dem Bildschirm.

In vvvv gibt es nur einen Knoten, „Point (GDI)“ der die Darstellung von beliebig vielen Punkten explizit ermöglicht und auch dies nur auf einer Ebene, sodass diese Punkte in zwei Dimensionen positionierbar sind. Implizit ermöglicht aber jeder Knoten, der im Renderer gezeichnet werden kann, auch eine Darstellung als Punktewolke der Eckpunkte. Der Knoten der einen gefüllten Würfel zeichnet, kann also auf eine Darstellung umgeschaltet werden, die nur die 6 Eckpunkte des Würfels zeigt.

2.2.2. Linien

Linien dienen dazu, mindestens zwei, aber auch mehr Punkte im Raum miteinander zu verbinden. Wie sich schon anhand dieser Definition erkennen lässt, greift eine Linie zwingend auf Punkte zurück, da diese zur Beschreibung des Linienverlaufs unabdinglich sind. Eine gerade Linie, in der euklidischen Geometrie entspräche dies einer Geraden mit unendlicher Länge oder einer Strecke mit endlicher Länge (vgl. Wolfram Research 2005b), wird als eindimensional bezeichnet, da sie sich nur in eine Dimension ausdehnen kann, aber keine Dicke besitzt.

vvvv bietet jedoch Möglichkeiten, sowohl gebogene Linien, zum Beispiel mit „BézierLine (GDI)“ zu zeichnen, als auch das Aussehen (Dicke und Stil) dieser Linien zu definieren. Des Weiteren können alle Objekte auch als Gitternetz dargestellt werden.

2.2.3. Flächen

Flächen, oder auch Polygone, sind eine Sammlung von mindestens 3 Punkten, zwischen denen eine gefüllte Fläche aufgespannt wird. Das einfachste Polygon ist das Dreieck, welches aus genau 3 Punkten besteht. Das Hinzufügen eines weiteren Punktes ergibt ein Rechteck, mit weiteren Punkten entsprechende Flächen mit mehr Eckpunkten.

Ein Gedanke, der zwar offensichtlich erscheint, aber während der Arbeit mit solchen Flächen immer wieder Kopfzerbrechen bereitet ist die Tatsache, dass die Reihenfolge, in welcher die Punkte verbunden werden, von entscheidender Bedeutung für das Aussehen der Fläche ist. vvvv bietet diverse Flächen an (Quadrate, abgerundete Quadrate, Dreiecke etc.) die als Ausgangsobjekte für generative Designs verwendet werden können.

2.2.4. Körper

Körper setzten sich aus unterschiedlichen Flächen zusammen, die so angeordnet werden, dass ein dreidimensionales Modell eines Objektes entsteht. In vvvv können solche Körper entweder generiert (Sphere (DX9), Cylinder (DX), usw.) oder importiert werden. So kann mit jedem 3D Modelling Programm ein Modell erstellt werden, das dann in vvvv in direkt manipulierbar ist.

vvvv stellt jedoch auch Knoten bereit, die die Erstellung von Körpern aus Punktewolken ermöglichen. So können komplexe 3D-Objekte direkt generiert und punktgenau manipluliert werden. Zudem existiert ein Knoten, der 3D-Text erzeugen kann. Dies funktioniert mit allen installierten Fonts und Symbol-Fonts.

2.3. Die Vervielfältigung von Objekten um komplexere Gebilde zu erhalten

Um die Grundformen nun in komplexere Objekte verwandlen zu können, stellt vvvv diverse Knoten ad hoc zur Verfügung. Ein zentraler Begriff ist hierbei das Konzept der Spreads:

> *"vvvv can simultaneously handle a large count of objects, either graphical or data, with very little effort by the user. This means that it is just as easy to do an operation on a single value or on a hundred, in the same sense it is as easy to draw one object as it is to draw a whole swarm of objects. This technique is called spreading. It is somehow similar to the concept of vectors, arrays or lists in other programming languages, but it lacks most of the overhead usually associated with these constructs."*
>
> *(meso 2004b)*

Technisch gesehen ist ein Spread nichts weiter als eine Liste von Werten, die in vvvv von vielen Knoten generiert und von praktisch jedem Knoten verarbeitet werden kann. So kann die Rotation eines Objektes um eine Achse mit einem Wert von 0 bis 1 (mit allen dazwischen liegenden Kommawerten) beschrieben werden. Ist dieser Wert 0, beträgt die Drehung 0°, bei 1 360°. Wenn jedoch statt nur einem Wert ein Spread mit zwei Werten eingegeben wird, entstehen zwei Objekte. Jedes mit seiner eigenen Drehung.

Abbildung 11 zeigt ein Beispiel für einen Knoten (Linear Spread), der eine definierte Anzahl von Werten auf einer Achse liefert, die alle den selben Abstand voneinander haben. Diese Werte werden dann an einen Knoten übergeben, der Punkte zeichnet. Abbildung 12 zeigt einige Knoten, die vvvv zur Erzeugung von Spreads bereithält und welche Formen diese erzeugen.

Abb. 11

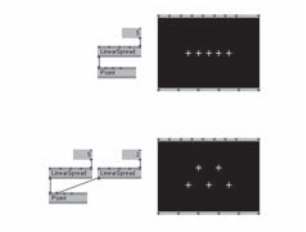


Abb. 13

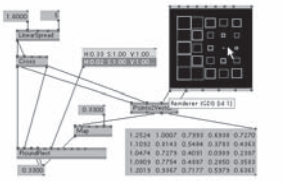
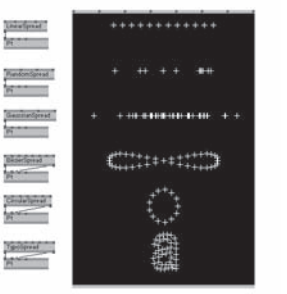


Abb. 12



Dies sind nur sehr einfache Beispiele und Komplexität wird vor allem durch Verknüpfung dieser Knoten untereinander und mit anderen Knoten erreicht. So zeigt Abbildung 13, wie die Elemente eines kachelförmigen Spreads (Cross) mit der Maus beeinflusst werden können.

Hier wurden nur sehr wenige Spreads verwendet um die Übersichtlichkeit nicht zu beeinträchtigen, auch wenn vvvv seine Stärken erst bei sehr großen Spreads (1000+ Elemente) voll ausspielt, da durch die sehr effiziente Verarbeitung von eindimensionalen Arrays (d.h. Listen), die vorhandene Hardware optimal ausgenutzt wird. ¬ 60

Durch die individuelle Verknüpfbarkeit jedes Pins mit einem Spread entsprechenden Datentyps (d.h. Farben zu Farben, Textstrings zu Textstrings etc.), lassen sich auch mehrere verschiedene Typen von Spreads mit unterschiedlicher Größe an einen Knoten hängen.

2.4. Mathematische Algorithmen zur Formengenerierung

Da nun alles nötige Wissen vermittelt wurde um ein grundsätzliches Verständnis von vvvv gewährleisten zu können, soll nun eine erste Anwendung gezeigt werden, die die bisher erklärten Punkte visualisieren soll.

Auch wenn es an den Designern liegt, zu entscheiden, welche Werkzeuge sei für welche Aufgaben einsetzt, sind gewisse Arbeiten dazu prädestiniert, von einem Computerprogramm erledigt zu werden. Die Konstruktion einer Spirale ist ein gutes Beispiel. Jeder der schon einmal versucht hat, diese eigentlich äußerst simpel anmutende Form mit der Hand zu zeichnen, weiß, welcher Aufwand dazu erforderlich ist. Das erzielte Ergebnis kann dabei nicht annähernd so perfekt sein, wie die Spirale, die von einem Computer erzeugt wurde. Alles was zu tun ist, ist ein mathematisches Formelsystem in den Computer einzugeben und diese von ihm berechnen zu lassen.

Das Ergebnis dieser, mit vvvv ausgeführten, Berechnungen zeigt Abbildung 14.

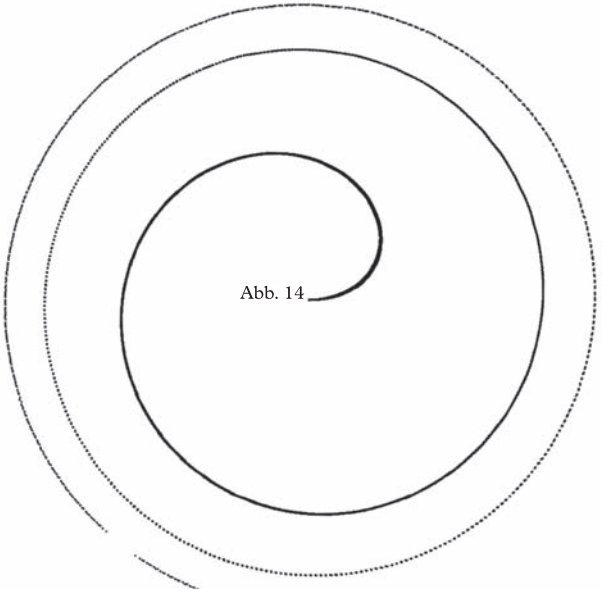


Abb. 14

Anzumerken ist, dass dem Zeichner die Formel in jedem Fall bekannt sein muss um die gewünschte Spiralform – im vorliegenden Fall, die schon von Archimedes für seine Studien verwendete Variante – zu erhalten. Sich von Hand die benötigten Punkte auszurechnen und die Figur dann zu konstruieren, erfordert einige Geduld und ist nur sehr engagierten Lesern zu empfehlen. Deren Umsetzung in vvvv ist jedoch sehr einfach zu lösen und sollte mit Hilfe von Abbildung 15 leicht reproduzierbar sein. Zudem kann dieser Patch zusammen mit allen anderen zur Erstellung dieses Werks verwendeten Patches von diploma.ampop.net heruntergeladen werden.

Abb. 15

Als zweites Beispiel in diesem Kapitel wird eine mathematische Formel präsentiert, die, obwohl für gänzlich andere Anwendungsgebiete bestimmt, sich sehr gut zur Erzeugung von generativen Bildern eignet. Mit einem Algorithmus zur Berechnung von seltsamen Attraktoren (ein Attraktor mit einer nicht ganzzahligen Dimension) kann ein Spread (d.h. Liste) generiert werden, die dann in ein zwei- oder dreidimensionales Koordinatensystem übersetzt werden kann um Objekte mit chaotischen Eigenschaften zu erzeugen. Dieses Kapitel stützt sich zum großen Teil auf Informationen aus dem, leider vergriffenen, Buch „Strange Attractors: Creating Patterns in Chaos“ (Sprott 2000) in welchem die hinter Fraktalen und chaotischen Funktionen steckenden Konzepte sehr ausführlich und verständlich besprochen werden. Es ist sogar der Quelltext (Basic und C+) von Programmen zur Generierung von Fraktalen abgedruckt und dokumentiert. Diese wurden vom Autor als Inspirationsquelle verwendet und in vvvv Patches übersetzt.

Die Formel $x_{n+1} = a + b*x_n + c*x_n^2$ enthält die 3 Kontrollparameter a, b und c. Sie stellt die allgemeinste eindimensionale quadratische Funktion dar. Eindimensional, wegen der einen freien Variable x, quadratisch, weil die höchste Potenz, die x in dieser Formel besitzt 2 ist.

Gefüttert mit unterschiedlichen Werten für die 3 Kontrollparameter ist es möglich, dass die Ergebnisse nach einigen Iterationen (Schleifendurchläufen) gegen unendlich streben oder sich auf einem gewissen Wert einpendeln und dort für immer verbleiben. Die ersteren werden als „uneingeschränkt" bezeichnet – die letzteren nennt man „Punkt-Attraktoren" – doch beide sind optisch relativ uninteressant. Ebenso verhält es sich mit zyklischen Attraktoren bei denen der Ausgabewert nach ein paar Iterationen nur noch zwischen zwei Werten hin und her springt. Die optisch interessanten Attraktoren hingegen sind die Instabilen, aber doch Eingegrenzten.

Die Variable x springt unvorhersagbar von einem Wert zum anderen, bleibt aber immer innerhalb gewisser ¬ 63
Grenzen. Es gilt nun also Tripletts von Kontrollparametern a, b und c zu finden, damit die Formel $x_{n+1} = a + b*x_n + c*x_n^2$ das gewünschte chaotische Verhalten zeigt. Da chaotische Prozesse extrem empfindlich auf die Ausgangsbedingungen reagieren und selbst kleinste Abweichungen das Aussehen dieser Funktion extrem verändern, ist es sehr schwierig solche Tripletts zu finden. Trial-and-Error ist laut Sprott nicht zielführend, jedoch gibt es den sogenannten „Lyapunov Exponenten".
Dieser kann Aussagen über das Verhalten einer Funktion treffen und so lassen sich interessante Werte zum Einsetzen in die Formel finden. (vgl. Sprott, 2000, 14ff)

Anhand dieser, von Sprott gezeigten Methoden, wurden vvvv-Patches angefertigt, die eine Berechnung von einfachen, eindimensionalen Funktionen ermöglichen, ebenso wie die Übersetzung derer Ergebnisse in zweidimensionale Kurven (Abb. 17-19), aus denen schließlich dreidimensionale Objekte (Abb. 23-25) generiert werden können.

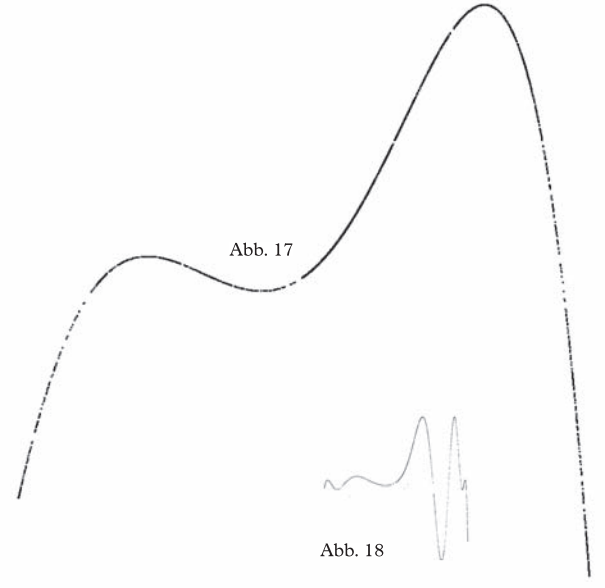


Abb. 17

Abb. 18

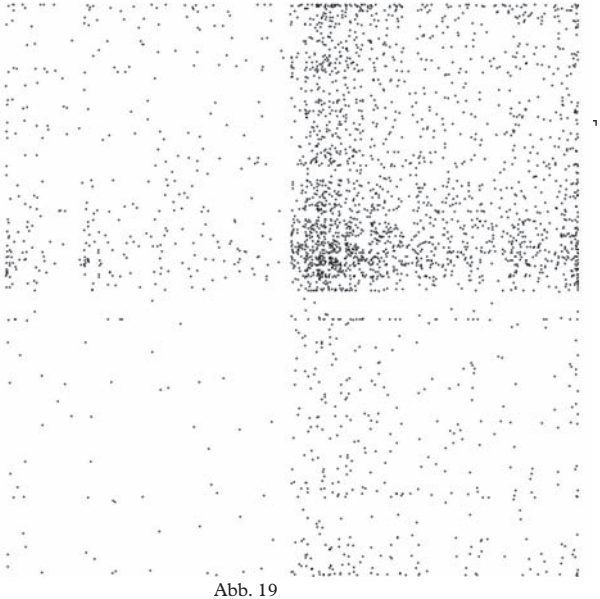


Abb. 19

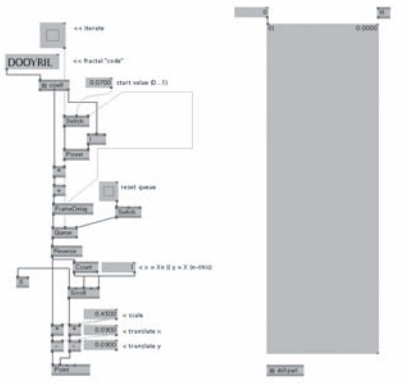


Abb. 20

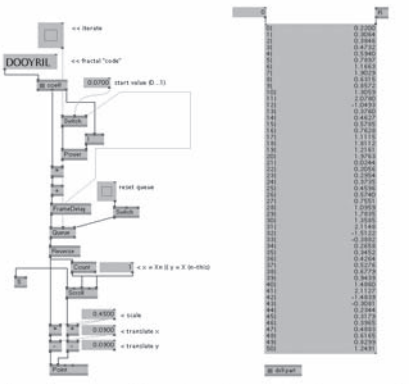


Abb. 21

Der in den Abbildungen 20 und 21 gezeigte Patch ist für die Berechnung der Grundkurve zuständig. Diese Kurve dient dann als Grundelement, das in einem dreidimensionalen Raum dupliziert und unterschiedlich transformiert werden kann. Dies geschieht mit dem in Abbildung 22 gezeigten Patch.

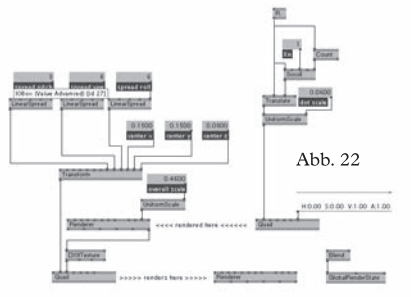


Abb. 22

Durch Veränderung der Kontrollparameter, insbesondere bei einer großen Anzahl an Objekten lassen sich die, in den Abbildungen 23 & 24 gezeigten, Rotationskörper erzeugen. Die Rotationen beschränken sich jedoch nicht auf eine Achse, sondern schließen alle drei Dimensionen ein. Auch die Skalierung und Positionierung der Einzelteile kann variiert werden. Diese so erzeugten Körper können nun wiederum als Basisobjekt für eine noch komplexere Transformierung in ein neues Objekt (Abb. 25) verwendet werden.

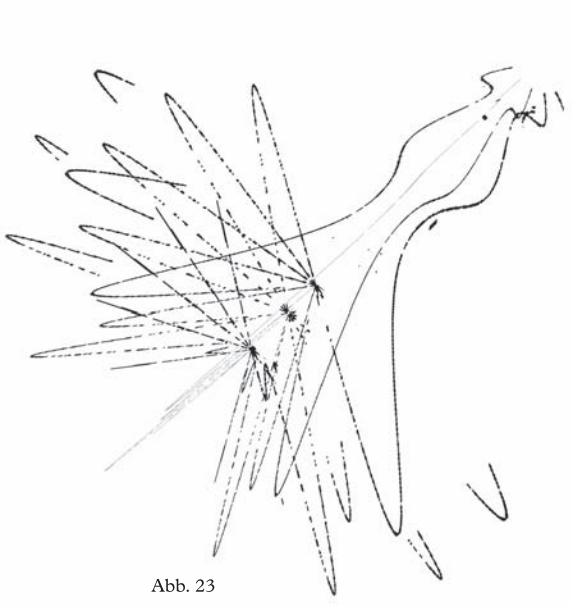


Abb. 23

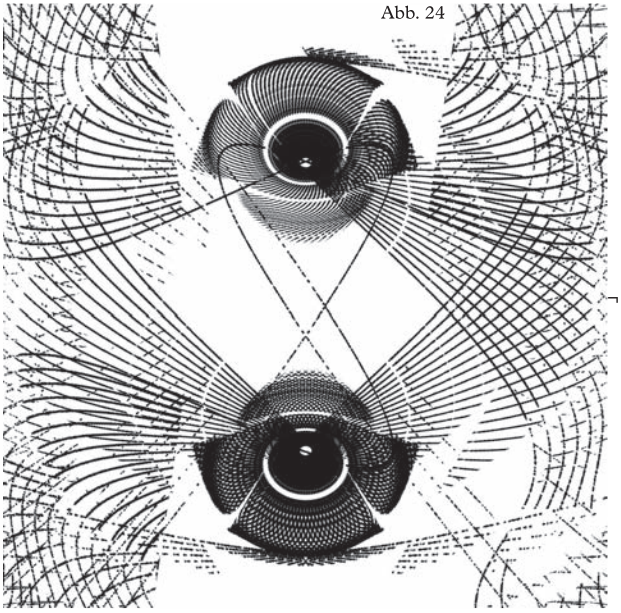
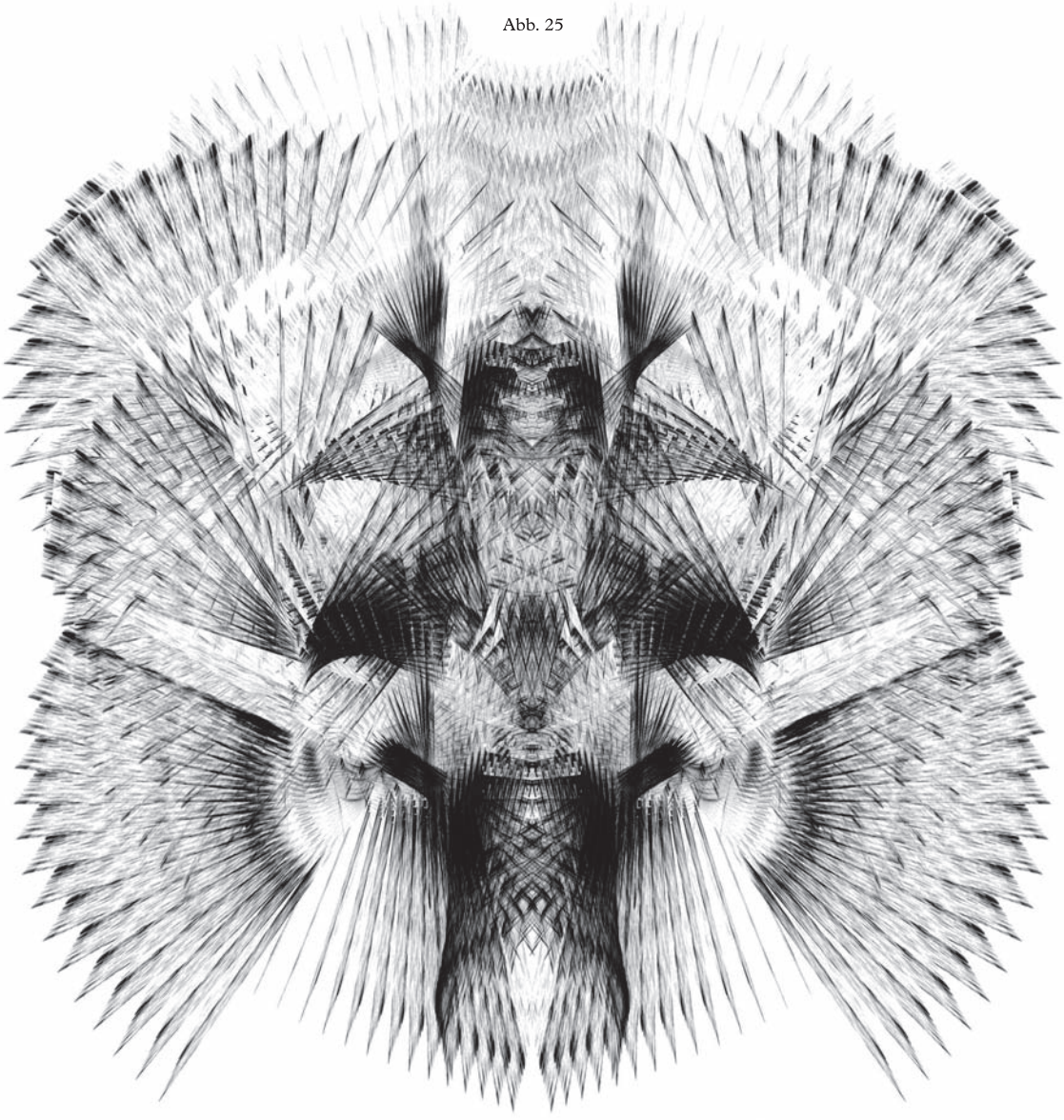


Abb. 24

Dies ist eine Methode die in ausgereifterer Form zur Erzeugung einiger der im Bildband gezeigten Werke verwendet wurde und stellt ein mächtiges Werkzeug dar, um die Beschränkung auf elementare Objekte zu umgehen. Ein Objekt wird mittels Algorithmen erzeugt und dann als Basisobjekt definiert, auf welches weitere Transformationen angewendet werden können. Sollte es hier noch an Komplexität mangeln, kann dieser Schritt beliebig oft wiederholt werden.

Abb. 25



2.5. Single Source Publishing mit vvvv

Während im vorherigen Kapitel gezeigt wurde, wie generative Grafiken erzeugt werden können, soll nun gezeigt werden, wie die so erhaltenen Bilder in anderen Applikationen weiterbearbeitet werden können und wie dies mit größtmöglicher Effizienz geschieht.

Eine zentrale Rolle beim Austausch von Daten zwischen verschiedenen Applikationen und Betriebssystemen spielt der immer mehr in Mode kommende XML Code. XML ist die Abkürzung für „Extensible Markup Language“ und ist eine so genannte Beschreibungs- oder auch Auszeichnungssprache (ähnlich HTML) zum zeichensystemunabhängigen Austausch von Informationen zwischen verschieden Applikationen und sogar Betriebssystemen. Als Referenz und einfacher Einstieg in die Thematik empfiehlt sich hier „XML. Der einfache Einstieg in den führenden Dokumenten- und Web-Standard“ von Günther Born. (Born 2005, 442)

XML ist eine gar nicht so neue Technologie, deren Vorläufer (GML, „Generalized Markup Language“) bereits in den siebziger Jahren von Dr. C. F. Goldfarb bei IBM entwickelt wurde und zur Verwaltung großer Dokumentmengen für Militär- und Raumfahrtprojekte entwickelt wurde. Daraus entwickelten sich dann SGML („Standardized Generalized Markup Language“), HTML und XML, das sich allerdings erst in den letzten Jahren zu einem verwendbaren Standard gemausert hat.

Auch vvvv arbeitet intern mit XML und alle Programme („Patches“) werden ausschließlich als XML Dateien gespeichert. Hier ein einfaches Beispiel für einen Patch mit zwei Knoten und einer Verknüpfung zwischen diesen:

```
<!DOCTYPE VVVV SYSTEM “http://vvvv.meso.net/versions/vvvv33beta8.1.dtd” >
<PATCH nodename=” addittion.v4p” componentmode=”InAWindow”>
 <BOUNDS type=”Window” left=”5775” top=”5790” width=”9000” height=”6000”></BOUNDS>

 <NODE nodename=”Add (Value)” id=”2”>
        <BOUNDS type=”Node” left=”2670” top=”2205” width=”0” height=”0”></BOUNDS>
        <PIN pinname=”Input 1” visible=”1”></PIN>
        <PIN pinname=”Input 2” visible=”1”></PIN>
        <PIN pinname=”Output” visible=”1”></PIN>
 </NODE>

 <NODE nodename=”IOBox (Value Advanced)” componentmode=”InABox” id=”3”>
        <BOUNDS type=”Node” left=”1920” top=”1635” width=”0” height=”0”></BOUNDS>
        <BOUNDS type=”Window” left=”7560” top=”7080” width=”215” height=”160”></BOUNDS>
        <BOUNDS type=”Box” left=”1920” top=”1635” width=”795” height=”240”></BOUNDS>
        <PIN pinname=”Y Output Value” visible=”1”></PIN>
        <PIN pinname=”Y Input Value” slicecount=”1” values=”2.00000”></PIN>
</NODE>

<LINK srcnodeid=”3” srcpinname=”Y Output Value” dstnodeid=”2” dstpinname=”Input 1”></LINK>

</PATCH>
```


Wie aus dem XML-Code klar ersichtlich wird, setzt sich das XML File aus verschiedenen Elementen zusammen. Zuerst werden die Eigenschaften des gesamten Patches deklariert, dann die Eigenschaften einzelner Knoten und schließlich die Verknüpfungen zwischen den Knoten.

Ein wichtiges Konzept von XML ist, dass es dem Anwender erlaubt, genau jene Tags zu deklarieren die für eine gewisse Anwendung gebraucht werden und nach deren Deklaration (mittels dtd Datei oder XML-Schemata) nur noch die deklarierten Tags unter genau festgelegten Umständen zulässt.

XML stellt nur Regeln für den Aufbau von Tags zur Verfügung, nicht aber die Tags selbst. Im Gegensatz zu HTML, wo alle Tags genau vordefiniert sind und dem User nichts anderes übrig bleibt, als die vorhanden Tags zu benutzen. Neue Tags zu erstellen ist nicht möglich.

HTML kann also spätestens seit der Einführung von XHTML als XML Dokument gesehen werden, das vom w3 Komitee (www.w3.org) mit genau definierten Tags spezifiziert wurde.

Dementsprechend besitzt die für das folgende Beispiel gewählte Struktur der XML Daten für den Datenaustausch zwischen einer Online-Bilddatenbank und einer InDesign-Dokumentvorlage für ein Buch gänzlich andere Tags:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE book SYSTEM "vvvvmanual.dtd">
<book>

<chapter id="10">
<h1>Chapter Headlines</h1>
<text>
<h2>Headline 2</h2>
here comes lots of text ...</text>

<!-- defining the images belonging to a chapter -->
<images>
<image id="0" size="100" href=file:///images/bild_1.tif comment="Ein Kommentar"></image>
<image id="1" size="100" href=file:///images/bild_2.tif comment="zu Bild 2"></image>
</images>

<!-- defining the footnotes belonging to a chapter -->
<footnotes>
<note id="0">Note #1</note>
<note id="1">Note #2</note>
</footnotes>
</chapter>
</book>
```


Hier gibt sich die Grundstruktur des Buches sehr gut zu erkennen. Jedes Kapitel enthält eine Kapitelüberschrift (<h1>), den dazugehörigen Fließtext mit allen benötigten Formatierungstags (<h2>, <h3>, <b>, <i>, <code>, <link> und <node>), Bildern und Fußnoten.

XML Files sehen also immer ähnlich aus, die verwendeten Tags und die damit geschaffenen Strukturen sind aber nur für eine spezielle Anwendung ausgelegt.

Da vvvv viele Funktionen zur Manipulierung von Texten bereithält, lässt es sich hervorragend für die Erstellung von XML Dokumenten aus vorhandenen Datenbeständen verwenden. Der dieser Diplomarbeit beiliegende Bildband wurde vollständig automatisch, anhand von ausgewählten Bildern und dazugehörigen Bildbeschreibungen erstellt und für den Export als pdf und für die Publikation im Internet aufbereitet. In der Werkdokumentation im Anhang wird dieser Prozess ausführlich erklärt und auf diploma.ampop.org können Beispieldateien und Stilvorlagen heruntergeladen werden.

{3} Conclusi(vvv)o

Im Laufe dieses Werks konnte gezeigt werden, dass es eine Methode gibt um dem Computer beizubringen, Designstücke nach definierten Regeln zu erzeugen und dass diese Methode dem Designer neue Werkzeuge liefert um seine kreative Freiheit ausleben zu können. Eine kleine Auswahl dieser Werkzeuge wurde im ersten Kapitel vorgestellt, doch im Zentrum der bisherigen Ausführungen stand die grafische Programmierumgebung vvvv. Diese wurde auch für die Erstellung und dem Management der im Werkteil präsentierten Arbeiten eingesetzt. Deshalb ist der beiliegende Bildband der eigentliche Beleg für die Gültigkeit der in dieser Diplomarbeit aufgestellten Definition von generativem Design und dessen Einfluss auf den Designprozess.

Indem gezeigt werden konnte, dass sich die Erstellung eines Bildbandes oder eines anderen Druckstückes automatisieren lässt, wird ein Werkzeug wie vvvv sehr wertvoll für den Designer. Wenn das Management von Datenbeständen (siehe Anhang I.I) und sogar die Ausführung von Designs dem Computer übergeben wird (Kapitel 2.3 & 2.4), kann sich der Designer ganz auf die dem Werk zugrunde liegende Idee konzentrieren. Dass sich der Designer bei seinen Überlegungen, wie ein gedachtes Design für einen Computer verständlich formuliert werden kann, auf einer anderen Ebene befindet als wenn er seine Ideen manuell umsetzt, wurde in Kapitel 1.2 erörtert.

Der große Vorteil von generativem Design ist, dass sich auf dieser Ebene viele Einschränkungen des analogen, händischen Designs umgehen lassen. Dies ermöglicht Resultate, die ohne den Computer nicht erreichbar wären (Kapitel 2.4). Der große Nachteil ist die Unverzichtbarkeit eines mathematischen Grundverständnisses und der Umgang mit einer Programmiersprache. Die Rolle der Mathematik kommentiert John Maeda sehr treffend in seinem Buch „Design by Numbers“:

> *„The thought of mathematics might seem horrifying at first, but after you see how much more can be accomplished with a few simple calculations, you will surely be addicted for life. […] Manipulating numbers is much like sculpting in clay or mixing paints. The only difference is in the computer's speed and precision."*
>
> *(Maeda 2001, 69)*

Indem er das Potenzial von einigen wenigen mathematischen Berechnungen hervorhebt und gleichzeitig die Mathematik mit einer Tätigkeit wie dem Mischen von Farben oder dem Schaffen einer Tonskulptur vergleicht, versichert Maeda, dass die Ergebnisse die Mühen rechtfertigen.

Der Prozess, der zu der Fertigstellung eines für die Erzeugung von Designstücken verwendbaren Programmes führt, ist dem Tonmodellieren tatsächlich sehr ähnlich: Aus vielen kleinen Stücken wird ein Programm geformt, das eine vom Designer definierte Grafik erzeugt. Mögliche Herangehensweisen hierzu werden in Anhang I.IV ausgeführt. Auch in Maedas Arbeit ist der Prozess sehr wichtig: ein Programm wird grob aufgebaut und die Ergebnisse ständig beurteilt um das Programm so lang zu verfeinern zu können, bis es das gewünschte Ergebnis liefert.

Da sich das Programm auf einer anderen Schaffensebene befindet, als die daraus erzeugte Grafik, muss generatives Design auf mehr als einer Ebene ästhetisch sein. Der beschrittene Weg ist ebenso Bewertungskriterium, wie das Ergebnis. Dies sollte auch im klassischen Design, wo ein leeres Dokument erstellt und händisch mit Inhalten gefüllt wird, so geschehen. Aber hier gibt es keine dem Design zugrunde liegende Beschreibung des Bildes, die eine Maschine verstehen könnte. Beim generativen Design existiert hingegen ein Programm, das alle Informationen implizit enthält um ein Design zu erzeugen. Paola Antonelli hebt dies im Vorwort von "Design by Numbers" ganz klar hervor:

> *„The most important part of Maeda's production, and the one he is most proud of, is not the final object, but rather the process. In his work, the process is the core that informs the real outcome"*
> *(Maeda 2001, 11)*

„In seiner Arbeit ist der Prozess der Kern, der das endgültige Ergebnis bestimmt." Ein Programm, das ein Kunstwerk erzeugt, kann aufgrund des in ihm enthaltenen Potenzials an sich ein Kunstwerk sein. Aus diesem Grund wurde großer Wert auf die Qualität der Patches gelegt, die in dieser Diplomarbeit verwendet wurden. Die meisten befinden sich auf der beiliegenden DVD oder lassen sich von diploma.ampop.net herunterladen. Diese Patches belegen die Funktionalität der in dieser Arbeit vorgestellten Methoden und sollen allen Interessierten einen leichteren Einstieg in die Materie ermöglichen. Dies geschieht in der Hoffnung, dass eine visuelle Programmierumgebung für Designer leichter erlernbar ist als eine textbasierte Programmiersprache, was zu einer weiteren Verbreitung des generativen Ansatzes führen könnte.

{a} Anhänge

I.) Die Werkdokumentationen

Im Zuge des Werkteils wird vor allem auf die Funktionen von vvvv zurückgegriffen, die es erlauben, Metadaten für eine Bilddatenbank zu generieren. Diese sollen dann in XML Daten übersetzt werden, die alle benötigten Informationen für eine Publikation im Web und als druckfertiges pdf enthalten.

Der Prozess, der für die Erstellung des Werkteiles eingesetzt wurde, lässt sich in 4 Schritte einteilen:

* Erstellung von generativen Grafiken
* Selektion der zu publizierenden Bilder, deren Kombination mit Informationen (Bildtitel, Beschreibungstext und andere Meta-Daten) und automatische Erstellung eines druckfertigen Bildbandes
* Verwertung der Daten für eine Online-Bilddatenbank
* Verwertung der generativen Grafiken für Live gespielte Animationen.

Auf Methoden zu Erstellung der Grafiken wird in Kapitel 2.3 und 2.4 eingegangen, der Prozess der Erstellung wird in Kapitel 2.6 beschrieben.

Diese Werkdokumentation konzentriert sich deshalb auf die automatische Erzeugung eines Bildbandes (Anhang I.I) und einer Online-Bilddatenbank (Anhang I.II) Auf die Animierbarkeit von Patches für Live Projektionen als Visuals geht Anhang I.III genauer ein. Und die Strategien, die zur Entwicklung der Patches eingesetzt wurden, werden in Anhang I.IV beschrieben

I.I Bildband Workflow

Für das Management der Bilddaten und deren Kombination mit Meta-Daten wurde ein vvvv Patch angefertigt der ein XML File (siehe Kapitel 2.5) produziert, das alle gewünschten Informationen enthält. Dieses kann mittels des vvvv Patches (Abb. 26) jederzeit verändert werden, um Bildtexte oder die Reihenfolge der Bilder zu verändern. Des Weiteren können Meta-Daten auch direkt aus bestehenden Datenbanken oder anderen Quellen importiert werden.

Im Falle des beiliegenden Bildbandes wurden nur die Bildtitel vom Autor vergeben, die Texte sind eine Kombination von Homers Odyssee (www.bookrags.com) mit tagesaktuellen Schlagzeilen aus dem BBC rss-feed (www.bbc.co.uk).

Abb. 26

Das bearbeitete XML File kann dann sofort in ein Adobe InDesign Template (Abb. 27) importiert werden. In diesem werden dann alle zuvor definierten Felder mit dem neuen Inhalt befüllt und das Dokument ist druckfertig (Abb. 28).

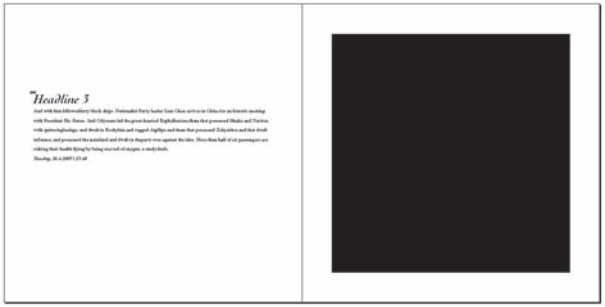


Abb. 27

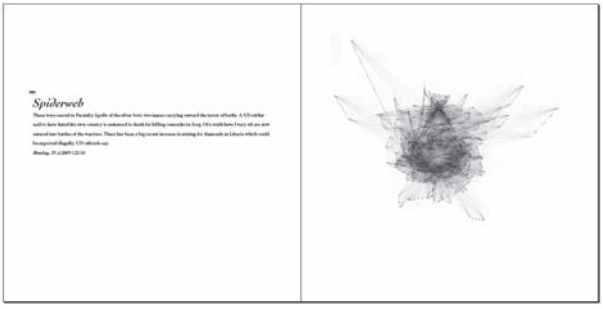


Abb. 28

Das Layout dabei auf Zweierpotenzen (2^0, 2^1, 2^2, 2^3, 2^4, etc) und alle Raster sowie die Objektformen sind quadratisch angelegt. Dies soll die Verbindung mit vvvv widerspiegeln, wo das Quadrat überall zu finden ist, was das vvvv-Logo (graues Quadrat) beweißt.

Als Schrift kam die Cochin zum Einsatz, die auch für den ***vvvv***-Schriftzug Verwendung findet. Dort in fett-kursiv, im Bildband auch im normalen Schnitt.

Wenn das Template korrekt aufgesetzt ist, dauert der Import der von vvvv aufbereiteten Daten, inklusive Formatierungsinformationen, nur wenige Sekunden. Selbstverständlich ist für das Aufsetzen des Templates einige Mehrarbeit erforderlich. So ist die Erstellung von Stilvorgaben und Platzhaltern erforderlich. Der eigentliche Prozess des Layoutens bleibt jedoch unverändert. Der Designer legt nach wie vor das Grund-Layout (Satzspiegel, Textformatierungen, Bild-und Textpositionen) manuell an. Dies muss er jedoch nur für eine Seite machen. Anschließend werden allen Elementen XML-Tags zugeordnet und die Seite beliebig oft vervielfältigt. Aus dieser Dokumentstruktur (Abb. 29) kann InDesign ein XML File generieren, das als Vorlage verwendet wird um vvvv so zu programmieren, dass es Bild- und Meta-Daten für InDesign generieren kann. Hier ein Beispiel XML File für ein zweiseitiges Buch:

```
<?xml version="1.0" encoding="UTF-8"?>
<Root>
<page>
<image href="file:///Images/filename.tif"></image>
<textborder><h1>Headline 01</h1>
<text>Any Text</text>
<date>Tuesday, 26.4.2005 | 23:48</date></textborder>
</page>

<page>
<image href="file:///Images/nichts.tif"></image>
<textborder><h1>Headline 1</h1>
<text>Text</text>
<date>Tuesday, 26.4.2005 | 23:48</date></textborder>
</page>
</root>
```

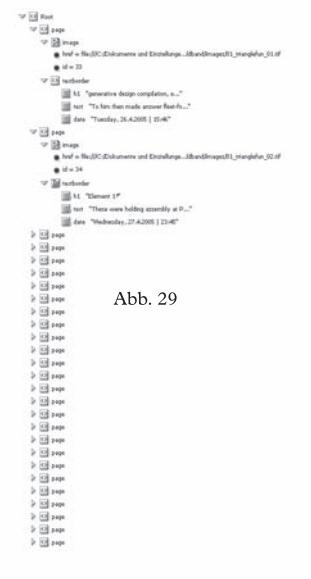


Abb. 29

Mit diesem System ist es möglich, eine große Anzahl von Bildern zu verwalten und je nach Bedarf, ein Dokument (z.B. einen Katalog) vollkommen personalisiert zu drucken. So könnte z.B. ein Portfolio für die Bewerbung in einer Grafikagentur, je nach Spezialisierung der Agentur, unterschiedlich zusammengesetzt sein. Und das ohne großen Mehraufwand.

I.II Web Workflow

Die für den Bildband erstellten Daten wurden von vvvv so umgewandelt, dass sie für den “Simple Viewer” (www.airtightinteractive.com), einem auf XML basierten Online-Bildbetrachter, verwendbar sind. Dazu wurde der in Abbildung 30 und 31 gezeigte Patch angefertigt:

Abb. 30

Abb. 31

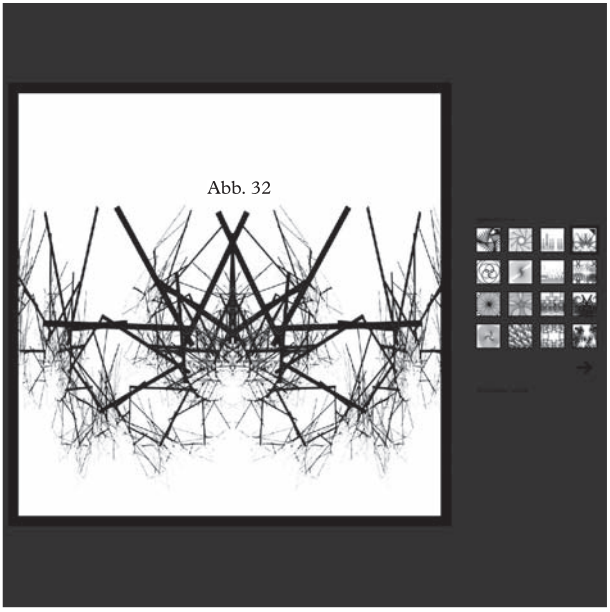


Abb. 32

Dieser übersetzt die InDesign XML-Tags und -Struktur in Code, der vom SimpleViever (Abb. 32) interpretiert werden kann. Auf diese Weise ist der Export in verschiedene Ausgabemedien gleichzeitig (Single Source Publishing, Kapitel 2.5) auf sehr schnelle und effektive Art und Weise möglich.

I.III VJing Workflow

Die in diesem Werk gezeigten Bilder stammen aus insgesamt 4 vvvv Patches die zusätzlich zur Ihrer Verwendung als Beispiele generativen Designs auch anlässlich einer Drum n' Bass Party in Wien als Live Performance gezeigt wurden.

Während des Einsatzes zur Erstellung unbewegter Designstücke standen die Patches still und jeder Parameter wurde manuell eingestellt um zu einem Resultat zu gelangen. Für den Einsatz als Live Projektionen musste jedoch der Faktor Zeit und somit Bewegung hinzugefügt werden. Statische Bilder ziehen erfahrungsgemäß weniger Aufmerksamkeit auf sich als bewegte Strukturen. Eine sehr kurze Einführung in das Thema Animation und vvvv findet sich in Kapitel 2.7.

Als Beispiel für den Workflow dient der für die Erzeugung der Bilder des Kapitels „Fun with Triangles" im Bildband (Seiten 0-15) eingesetzte Patch. Dazu musste der Formgenerierungspatch (Abb. 33) in eine Struktur eingebettet werden, die eine Kontrolle über das Verhalten dieses Patches ausüben kann. Dazu werden zuerst alle zu verändernden Parameter mit benannten Eingabefeldern („spreadcount", „color", „overall scale", usw.) verknüpft, damit diese dann von einem übergeordneten Patch mit Werten gefüttert werden können.

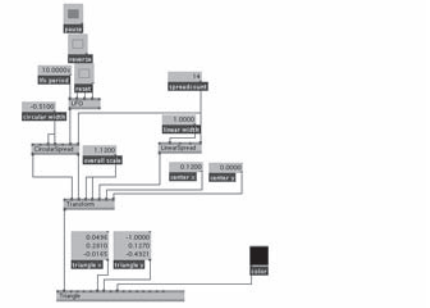


Abb. 33

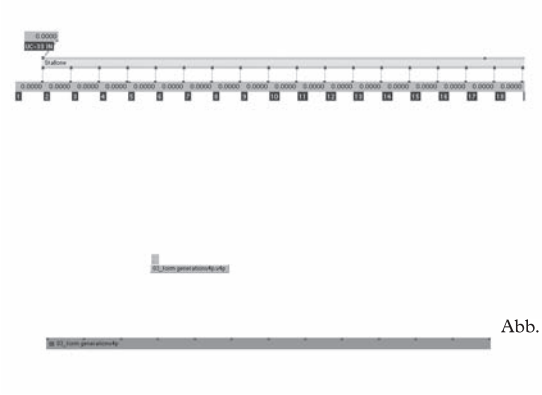


Abb. 34

Dieser übergeordnete Patch (Abb. 34) ist ein vorgefertigtes Template, das vom Autor für die Verwendung mit einem speziellen Eingabegerät in einem Live Umfeld ausgelegt wurde. Es gibt eine breite Palette von Peripheriegeräten für Computer, die eine Dateneingabe ermöglichen. In diesem Fall handelt es sich über eine sogenannte „Faderbox“, ein Eingabegerät mit 9 Schiebereglern, 24 Drehreglern und diversen Knöpfen. Damit können Zahlenwerte an vvvv geliefert werden: Jeder Regler gibt einen Wert zwischen 0 und 1 aus, je nach Position der Schieber bzw. Drehung der Knöpfe.

In diesen Patch wird nun der zuvor präparierte Formgenerierungspatch (Abb. 33) eingebettet, wobei dieser nur noch als einfacher Knoten angezeigt wird über dessen Pins Werte an die benannten Eingabefelder im Formgenerierungspatch geliefert werden.

Der Kontrollpatch (Abb. 34) wird wiederum in einen Patch (Abb. 35) eingebettet, der die oberste Kontrollinstanz repräsentiert und ebenfalls als Template vorliegt. Dort werden später alle Informationen über den Zustand des Formgenerierungspatches zusammenlaufen.

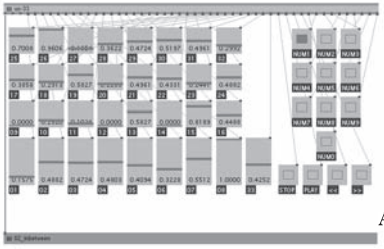


Abb. 35

Als nächstes werden alle Eingabefelder des Formgenerierungspatches eine Ebene nach oben in den Kontrollpatch kopiert und mit den entsprechenden Pins verbunden (Abb.36+37)

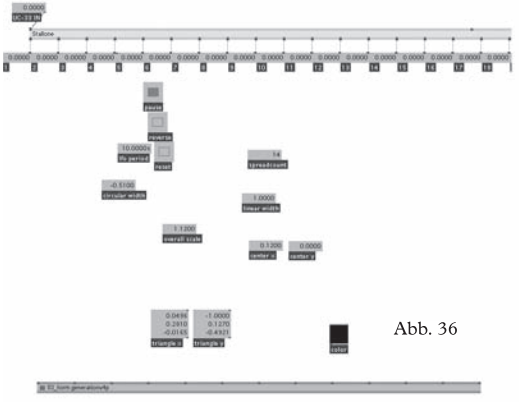


Abb. 36

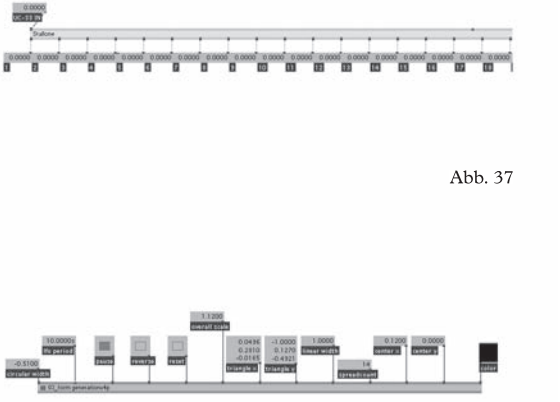


Abb. 37

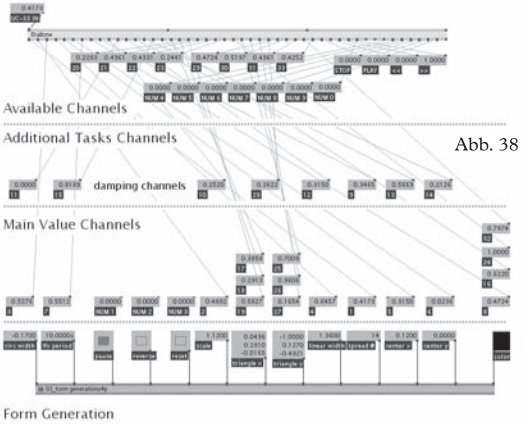


Abb. 38

Nun muss überlegt werden, welche Regler welche Parameter steuern sollen und ob neben der Faderbox noch auf andere Datenquellen, wie z.B. Audio-Analysedaten, zugegriffen werden soll. Abbildung 38 zeigt so eine Anordnung, wobei eine Unterscheidung zwischen primären und sekundären Kontrollparametern getroffen wird.

Erstere kontrollieren die Formgenerierung an sich, während die sekundären das Verhalten der Kontrollparameter beeinflussen. So kann ein Regler gedämpft werden um weiche oder sehr langsame Bewegungen zu ermöglichen. Ein sekundärer Parameter würde dann die Dauer dieser Verzögerung steuern.

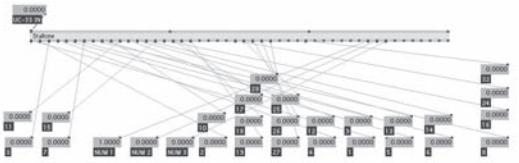
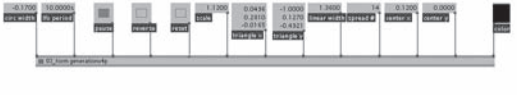


Abb. 39

Nach dem Löschen der nicht benötigten Eingangswerte („Available Channels“) werden die zu verknüpfenden Kanäle, wie in Abbildung 39 gezeigt, übersichtlich angeordnet.



Nun müssen die von der Faderbox gelieferten Werte in Positionen, Skalierungen, Bewegungsgeschwindigkeiten und Farbwerte umgewandelt werden.

Für den Einsatz in einem Live-Kontext muss eine gewisse Stabilität und Vorhersehbarkeit der berechneten Formen gewährleistet sein um eine hochqualitative Performance zu ermöglichen. Während bei Standbildern nur ganz spezielle Eingabewerte auf deren generative Ästhetik hin überprüft werden müssen, bedarf es bei ständiger Variation dieser Werte eines gut eingegrenzten Wertebereiches, in dem ALLE Eingabewerte schöne Formen erzeugen.

Dafür wurden vom Autor oft verwendete Kontrollstrukturen in Module umgewandelt, die eine schnelle Verknüpfung zwischen Eingangswerten und zu kontrollierenden Parametern ermöglichen. Die Abbildungen 40 bis 42 zeigen:

* ein Modul zur Umwandlung von 4 Eingangswerten in eine Farbe.
* einen Patch, der einen Wert von 0 bis 1 dämpft und in einen beliebigen Wertebereich (z.B. von -100 bis +100) umrechnet.
* ein Modul, das für die Steuerung von Rotationsanimationen verwendet wird.

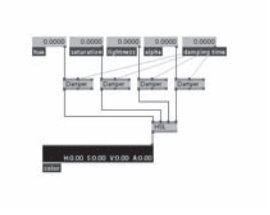


Abb. 40

Abb. 41

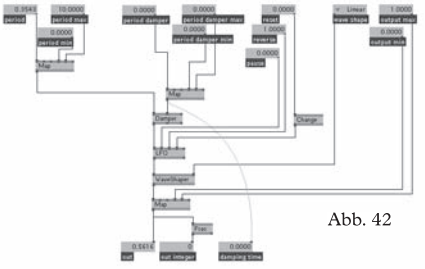


Abb. 42

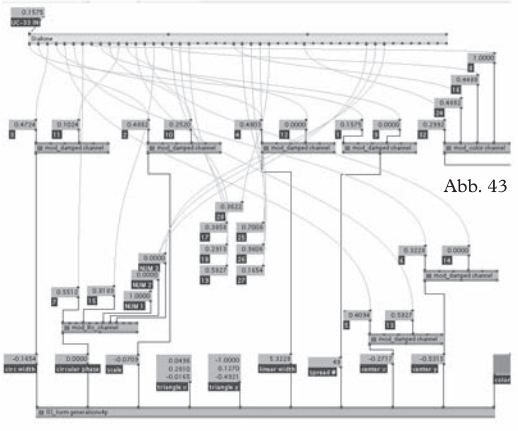


Abb. 43

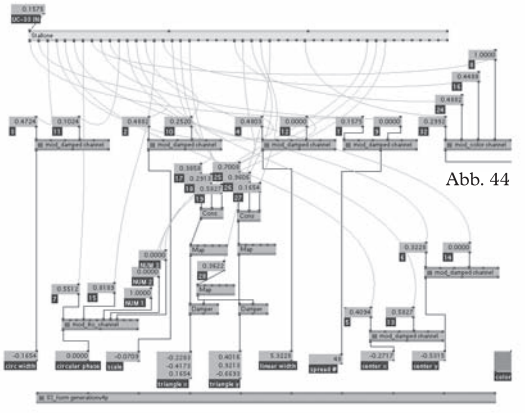


Abb. 44

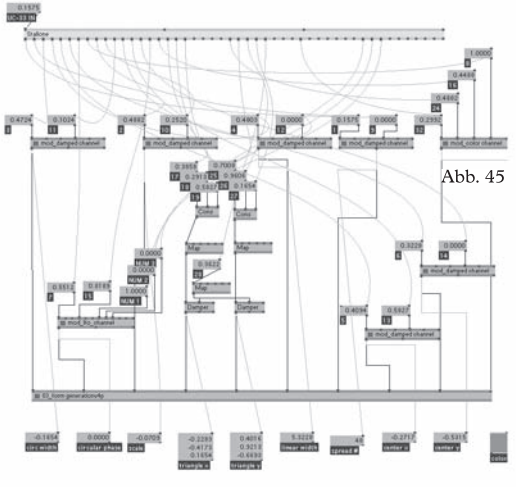


Abb. 45

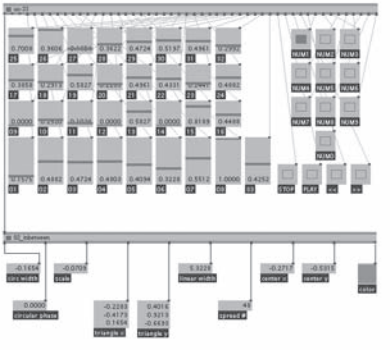


Abb. 46

Diese Module können nun im Kontrollpatch (Abb. 34) die Übersetzung der Faderbox-Werte in passende Werte für die Formgenerierung übernehmen (Abb. 43). Allerdings wird es sich nie vermeiden lassen, dass für gewisse Umrechnungen kein Modul bereit steht, so dass diese Verbindungen dazuprogrammiert werden müssen, was Abbildung 44 zeigt. Anschließend (Abb. 45) werden die berechneten Werte direkt mit den entsprechenden Pins verbunden, so dass die benannten Eingabefelder Daten an die oberste Instanz (Abb. 35) übergeben können. Durch Kopieren und Verknüpfen mit den so geschaffenen Output-Pins kann nun im obersten Patch (Abb. 46) die Position jedes Faderbox-Reglers und die Veränderung jedes Wertes abgelesen werden.

Nun kann das Projekt getestet werden und meist finden sich noch viele Möglichkeiten, die Kontrolle des Users über den Patch zu optimieren und die Bedienbarkeit zu erleichtern. Dies führt zu den in Abbildung 47 und 48 gezeigten Kontrolloberflächen.

Alle nicht benötigten Knoten und Verknüpfungen lassen sich sperren und | oder verstecken, wodurch größtmögliche Übersichtlichkeit erreicht werden kann.

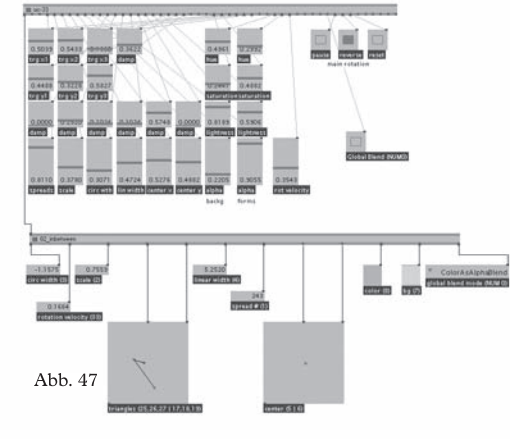


Abb. 47

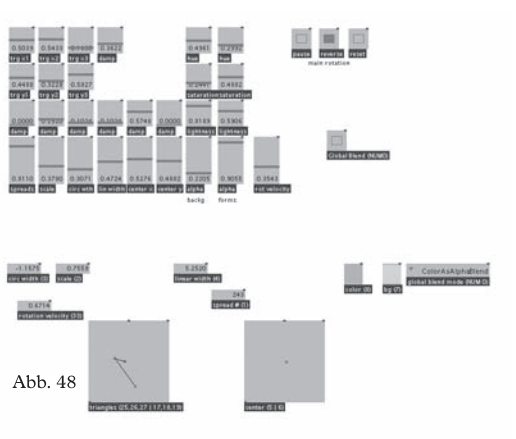


Abb. 48

Der in dieser Dokumentation besprochene Patch ist eine stark vereinfachte Form des bei der Live Performance in der Arena Wien gespielten Patches. Zur Illustration der Vorgehensweise bei der Umwandlung eines statischen Projektes in ein dynamisches, sollte dieses Beispiel jedoch genügen.

Die Abbildungen 49 bis 54 zeigen Bilder dieser Live Visuals, die in Kombination mit den wunderbar passenden Texturen und Strukturen von Tanja Tomic entstanden. Zudem wird der hier vorgestellte Workflow anlässlich eines Vortrags bei der „Schmiede the Conference“ (www.schmiede.ca), einer internationalen Medienkulturkonferenz auf der Pernerinsel in Hallein, vorgestellt werden. Bei der Abschlussveranstaltung „FocusOn“ werden die selben Visuals in einer alten Salzfabrik projiziert.

Abb. 49

Abb. 50

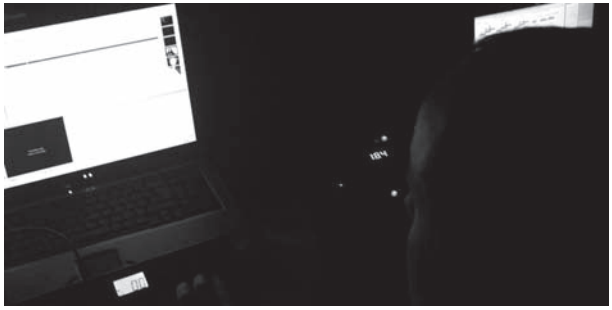


Abb. 51

Abb. 52

Abb. 53

Abb. 54

I.IV Strategieen für die Erstellung von generativem Design

I.IV.I. Intuitiv

Bei einem intuitiven Umgang mit vvvv verlässt sich der Designer mehr auf den Zufall und versucht, durch ein intuitives Auswählen und Verknüpfen von Knoten interessante Ergebnisse zu erzielen. Er weiß dabei nicht immer, was er macht und wie genau seine Eingriffe den Datenfluss zwischen den Knoten beeinflussen. Wenn etwas funktioniert, wird es beibehalten und wenn eine Intervention kein erwünschtes Ergebnis bringt, wird diese wieder rückgängig gemacht. Meist entstehen so äußerst organisch anmutende Patches, mit Verknüpfungen in alle Richtungen und keiner erkennbaren Strukturierung.

Der Datenfluss ist bei einem solchen Patch kaum noch nachzuvollziehen, was daher rührt, dass während der Erstellung nicht über eine logische Anordnung nachgedacht wurde, sondern alle Funktionen „gewachsen“ sind. Was gefiel blieb, was nicht gefiel, wurde gelöscht. Es kann auch sein, dass Knoten im Programm verbaut sind, die absolut keinen Sinn ergeben oder keinerlei Einfluss auf den Output vom Patch haben. Auch von einer „Evolution“ kann hier gesprochen werden, die, nach dem heutigen Wissensstand, auch nicht zielgerichtet, aber sehr wohl regulierend ist. Interessant ist hier, dass sich nach eingehender Analyse und Neuanordnung von solchen darwinistischen Patches sehr oft ein strukturierter Datenfluss herauskristallisiert, was auf eine gewisse Selbstorganisation schließen lässt.

Diese Methode bietet sich vor allem für Anfänger an, da nur ein minimales Grundverständnis notwendig ist um einen Einblick in viele Funktionen und vor allem die Logik von vvvv zu erhalten, die dann helfen, die nächsten Schritte zu machen.

I.IV.II. Zielgerichtet

Bei der zielgerichteten Methode wird ein Patch absolut logisch aufgebaut. Der Designer hat ein klares Ziel vor Augen und den Weg, den er dazu beschreiten muss. Dies erfordert natürlich mehr Wissen über vvvv als andere Ansätze, doch für Projekte mit genau festgelegtem Ergebnis ist dies der einzig zielführende. Das Projekt wird in logische Blöcke zerlegt und diese dann zu einem Ganzen zusammengefügt.

Dies gewährleistet eine relativ einfache Fehlersuche, da die Funktion von einzelnen Blöcken (auch Subpatches genannt) leicht getestet werden kann und der Datenfluss zwischen den Subpatches gut nachvollziehbar ist. Falls es zu Performanceproblemen kommen sollte, lassen sich hier leicht Möglichkeiten finden um die Leistung des Patches zu optimieren.

Vor allem in Produktionsumgebungen, in denen mehrere Leute an einem Projekt arbeiten, sollte eine solche Methode angewendet, werden um Nachvollziehbarkeit zu gewährleisten.

I.IV.III. Gesteuert

Bei der gesteuerten Arbeit mit vvvv handelt es sich um eine Kombination der zwei vorangegangenen Methoden. Nicht ganz so chaotisch wie die intuitiv gewachsenen Patches aber auch nicht so streng gegliedert und determiniert wie die konstruierten Patches der zielgerichteten Arbeit.

Der Designer muss dabei nur ein ungefähres Ziel vor Augen haben und eine grobe Vorstellung, wie der Patch aussehen könnte, um vvvv die gewünschten Bilder zu entlocken. Meist kommt dabei nicht ganz das ursprünglich geplante heraus, kann aber trotzdem den gestellten Anforderungen genügen.

Diese 3 Methoden sind allerdings nur als grobe Unterscheidungen zu sehen und es ist jedem selbst überlassen, den passenden Weg zu finden.

II.) Literaturverzeichnis

ADOBE, Creative Team (2003): Classroom in a Book: Adobe Illustrator CS. Berkeley, CA: Adobe Press.

ALIAS Systems Corp. (2005): Modeling.

URL: http://www.alias.com/eng/products-services/maya/technical_features/modeling.shtml (08.05.2005)

ARTCYCLOPEDIA (2003): Artists by Movement: Optical Art.

URL: http://www.artcyclopedia.com/history/optical.html (13.04.2005)

BAEZA-YATES, Ricardo (2005): Visual Programming.

URL: http://www.dcc.uchile.cl/~rbaeza/cursos/vp/todo.html (10.05.2005)

BEGEL, Andrew (1996): LogoBlocks: A Graphical Programming Language for Interacting with the World. Cambridge, MA: Epistemology and Learning Group, MIT Media Laboratory.

BORN, Günter (2005): XML. Der einfache Einstieg. München: Markt+Technik Verlag.

BURNETT, M. | GOLDBERG, A. | LEWIS, T. (1994):

Visual Object-Oriented Programming: Concepts and Environments. Prentice-Hall.

CAGLE, Kurt (2002): SVG Programming: The Graphical Web. Berkeley, CA: Apress.

CHURCH, Alonso (1936): Review of Turing: Journal of Symbolic Logic, 2, 42-43.

CNET (2005): Adobe übernimmt Macromedia.

URL http://www.zdnet.de/news/business/0,39023142,39132388,00.htm (08.05.2005)

CYCLING ´74 (2004): Max/MSP. URL: http://www.cycling74.com/products/maxmsp.html (10.04.2005)

ECO, Umberto (1962): La definizione dell'arte.

In: Introduzione al catalogo della mostra Arte programmata, negozio Olivetti. Mailand: Garzanti.

EUKLID | THAER, Clemens (2003): Die Elemente, Buch 1-13. : Harri Deutsch.

FERNANDO, R. | KILGARD, Mark (2003): The CG Tutorial. The Definitvie Guide to Programmable Real-Time Graphics. Boston: Addison-Wesley.

FITZGERALD, Michael (2004): XML Hacks. 100 Industrial-Strenght Tips &Tools. Sebastopol, CA: O´Reilly Media, Inc..

FORSSMAN, Friedrich | DE JONG, Ralf (2002): Detailtypografie. Nachschlagewerk für alle Fragen zu Schrift und Satz. Mainz: Verlag Hermann Schmidt.

FRANKLIN, Derek | MAKAR, Jobe (2002): Macromedia Flash MX ActionScripting: Advanced Training from the Source. San Francisco, CA: Macromedia Press.

FRY, Ben / REAS, Casey (2004): Processing 1.0 _ALPHA_. URL: http://processing.org/ (03.04.2005)

GALILEI, Galileo (1994): Dialog über die beiden hauptsächlichen Weltsysteme. In: MAURY, Jean Pierre: Und sie bewegt sich doch. Ravensburg: Verlag Ravensburger, S. 130ff.

GOLIN, E. J. (1990): The Specification of Visual Language Syntax. In: jvlc. 01/1990. S141-157.

GOSLING, J. | JOY, B. | STEELE, G. | BRACHA, G. (2000): The Java Language Specification 2nd Edition. Boston: Addison-Wesley.

HEFLAND, Jessica (2001): Screen. Essays on Graphic Design, New Media, and Visual Culture. Princeton: Princeton Architectural Press.

HOFSTADTER, Douglas R. (1991): Gödel Escher Bach. Ein endloses geflochtenes Band. München: dtv.

HOPCROFT, John E. | ULLMAN, Jeffrey D. (2002): Einführung in die Automatentheorie, Formale Sprachen und Komplexitätstheorie. München: Pearson Studium.

IBM (2004): IBM Archives: IBM international recognition. URL: http://www-03.ibm.com/ibm/history/exhibits/logo/logo_8.html (08.05.2005)

KNUDSEN, Jonathan (1999): Java 2D Graphics. Sebastopol, CA: O´Reilly Media, Inc.

LOTT, Joey (2003): ActionScript Cookbook. Sebastopol, CA: O´Reilly Media, Inc.

MACROMEDIA, Inc. (2005a): Macromedia Flash MX 2004 At a Glance. URL: http://macromedia.com/software/flash/productinfo/overview/ (17.04.2005)

Macromedia, Inc. (2005b): Macromedia Flash Player Statistics.
URL: http://www.macromedia.com/software/player_census/flashplayer/ (08.05.2005)

MAEDA, John (2001): Design by Numbers. Cambridge, MA: MIT Press.

MAEDA, John (2004): Creative Code. New York: Thames and Hudson Ltd.

MCCLELLAND, Deke (2004): Photoshop Cs Bible. New York: Wiley.

MESO (2004a): vvvv : Propaganda.
URL: http://vvvv.meso.net/tiki-index.php?page=Propaganda (05.05.2005)

MESO (2004b): vvvv : Spreads. URL: http://vvvv.meso.net/tiki-index.php?page=Spreads (05.05.2005)

MOHOLY-NAGY, László (1933): The New Typography. Köln: Bauhausverlag.

MYERS, B. A. (1990): Taxonomies of Visual Programming and Program Visualization.
In: jvlc. 01/1990. S.97-123.

SPROTT, Julien C. (1993): Strange Attractors: Creating Patterns in Chaos. New York: M&T Press.

STEIN, Dorothy (1984): Ada, A Life and a Legacy. Cambridge, MA: MIT Press.

SVG Working Group (2003): Scalable Vector Graphics (SVG) 1.1 Specification.
URL: http://www.w3.org/TR/SVG/ (19.04.2005)

TURING, Alan (1936): On Computable Numbers, With an Application to the Entscheidungsproblem.
New York: Proceedings of the London Mathematical Society, Series 2, Volume 42.

WIENER, Oswald (1969): Die Verbesserung von Mitteleuropa: Reinbek.

WITTGENSTEIN, Ludwig (1921): Logisch-Philosphische Abhandlung. Wien.

WOLFRAM Research, Inc (2005a): Point. URL: http://mathworld.wolfram.com/Point.html (08.05.2005)

WOLFRAM Research, Inc (2005b): Line. URL: http://mathworld.wolfram.com/Line.html (08.05.2005)

ZMÖLNIG, Johannes (2002): GEM. URL: http://gem.iem.at/ (10.04.2005)

III.) Abbildungsverzeichnis

Alle Abbildungen des Autors, außer den 4 mit n_gen erzeugten Grafiken, wurden mit vvvv gefertigt oder sind Screenshots von vvvv Patches.

27	2005	Grafik des Autors	Leeres InDesign Template
28	2005	Grafik des Autors	InDesign Template mit importierter XML Struktur
29	2005	Grafik des Autors	InDesign XML Struktur (Auszug)
30	2005	Grafik des Autors	XML Konverter: InDesign zu SimpleViewer #1
31	2005	Grafik des Autors	XML Konverter: InDesign zu SimpleViewer #2
32	2005	Grafik des Autors	SimpleViewer
33	2005	Grafik des Autors	Formgenerierungspatch
34	2005	Grafik des Autors	Kontrollpatch
35	2005	Grafik des Autors	Root Patch
36	2005	Grafik des Autors	Kontrollpatch #2
37	2005	Grafik des Autors	Kontrollpatch #3
38	2005	Grafik des Autors	Kontrollpatch #4
39	2005	Grafik des Autors	Kontrollpatch #5
40	2005	Grafik des Autors	Modul: Farbgenerierung
41	2005	Grafik des Autors	Modul: Damper
42	2005	Grafik des Autors	Modul: LFO
43	2005	Grafik des Autors	Kontrollpatch mit Modulen #1
44	2005	Grafik des Autors	Kontrollpatch mit Modulen #2
45	2005	Grafik des Autors	Kontrollpatch mit Modulen #3
46	2005	Grafik des Autors	Root Patch #1
47	2005	Grafik des Autors	Root Patch #2
48	2005	Grafik des Autors	Root Patch #3
49	2005	Tanja Tomic	Visuals, Arena Wien, 23.04.2005
50	2005	Tanja Tomic	Visuals, Arena Wien, 23.04.2005
51	2005	Tanja Tomic	Visuals, Arena Wien, 23.04.2005
52	2005	Tanja Tomic	Visuals, Arena Wien, 23.04.2005
53	2005	Tanja Tomic	Visuals, Arena Wien, 23.04.2005
54	2005	Tanja Tomic	Visuals, Arena Wien, 23.04.2005

IV.) DVD Inhaltsverzeichnis

cut here for dvd cover

DVD ROOT

disclaimer.gif

generatives design mit grafischen programmierumgebungen.pdf

generativvvve.pdf

firefox bookmarks.html

readme.txt

2^7 FILES

128 generativvvve images

- matrioschka baseform_01.tif
- [...]
- matrioschka baseform_27.tif
- matrioschka text_01.tif
- [...]
- matrioschka text_29.tif
- simple polygon_01.tif
- [...]
- simple polygon_17.tif
- strange attractor_01.tif
- [...]
- strange attractor_07.tif
- strukt logo_01.tif
- [...]
- strukt logo_21.tif
- typo experiments _01.tif
- [...]
- typo experiments _08.tif
- typo triangles_01.tif
- [...]
- typo triangles_19.tif

arena vvvvjing pictures

- strukt arena_01.jpg
- [...]
- strukt arena_52.jpg

vvvv_33beta8.1

- Args.txt
- difff.xml
- file.txt
- license.txt
- readme.txt
- root.v4p
- uninstall.exe
- ***vvvv***.exe

V4 FULL VERSION

- bin
- effects
- freeframe
- girlpower
- help
- modules

cut here for dvd cover

VVVV

- 00_image generation
  - high res render template.v4p
  - typo triangles.v4p
- 01_print workflow
  - 01_template
    - generativvvve template.indd
    - PrintData leer template.xml
  - 02_xml out of images
    - 02_random textgen.v4p
    - mod_illias.v4p
    - mod_news.v4p
    - mod_timestamp.v4p
    - mod_xml combine.v4p
  - 03_imported xml
    - generativvvve.indd
    - PrintData_first import.xml
  - 04_data manipulation
    - 04_data manipulation.v4p
    - mod_illias.v4p
    - mod_image browser.v4p
    - mod_news.v4p
    - mod_timestamp.v4p
    - mod_xml combine.v4p
    - mod_xml split.v4p
  - images
    - 01_trianglefun_01.tif
    - [...62 IMAGES...]
    - 06_rest_04.tif
    - nichts.tif
    - thumbs
      - 01_trianglefun_01.jpg
      - [...62 IMAGES...]
      - 06_rest_04.jpg
- 02_web workflow
  - 02_web export.v4p
  - mod_color hex code.v4p
  - mod_web xml combine.v4p
  - mod_xml split.v4p
  - simple viewer
    - flash_detect.js
    - imageData.xml
    - index.html
    - upgrade.gif
    - upgrade.html
    - viewer.swf
    - images
      - 01_trianglefun_01.jpg
      - [...62 IMAGES...]
      - 06_rest_04.jpg
    - thumbs
      - 01_trianglefun_01.jpg
      - [...62 IMAGES...]
      - 06_rest_04.jpg
- 03_vj-ing workflow
  - 01_topcontrol.v4p
  - 02_inbetween.v4p
  - 03_form generationv4p.v4p
  - mod_color channel.v4p
  - mod_damped channel.v4p
  - mod_lfo_channel.v4p
  - templates
    - 01_topcontrol.v4p
    - 02_inbetween.v4p
    - 03_form generationv4p.v4p
- 04_gimmicks
  - directory to list.v4p
  - vvvv layout machine.v4p
  - typotriangles.v4p
  - strange attractors
  - toc and list generator
    - commas to word convert.v4p
    - Diploma Literature.xls
    - unformatted list.txt

V.) Das Werk (generativvve)

Graphic design is the most prevalent form of all the arts. It fulfills personal needs as well as the needs of society and embraces both economic and ergonomic concepts; it is connected to numerous other disciplines including art, architecture and literature. Graphic design is produced in such volume that it covers most of our urban spaces and nearly every object we encounter in our everyday life. And still the designer's profession and his way of life has not changed much since the early 1930s. Until the Second World War, graphic design was for the most part a commercial art practiced by printers and typesetters and rarely seen as an opportunity to implement ideas in a creative way. This was very unlike the experimental approach of the Bauhaus members, with their influences from cubism and constructivism. Their movement was driven by the demands of commerce, but fueled by the idea of eliminating the economic hardship caused by the Depression. Designers like Lester Beall and Paul Rand then combined this more art-like approach to graphic design with the purely commercial graphic design that the American economy demanded. This helped to inaugurate a new visual language that would revolutionize the role of design worldwide and until today. The relationships between art and design, between design and aesthetics, and between aesthetics and experience are the central topics of this emancipated design, as are the role of intuition and ideas, the balance between form and function and, maybe most importantly, the universal language of geometry and typography.

To show how these ideas can be directly applied to the process of design creation, this book contains a selection of 64 computer-generated images which were chosen from 128 originally created. Because the images were not created by the designer himself, but by a computer program written by the designer, the central question is whether a computer is capable of creating appealing design on its own, or whether the principles of aesthetics can only be conceived by humans or other intelligent forms of life.

For each of the six chapters in this book, a little program was produced. Each program is capable of producing a virtually unlimited number of images, and each image is unique, yet similar to the others. This is caused by a random or planned variation of the program's input parameters. The designer defines behavioral rules, including their limits, and thus can, by carefully constructing this rulebook, achieve the desired result.

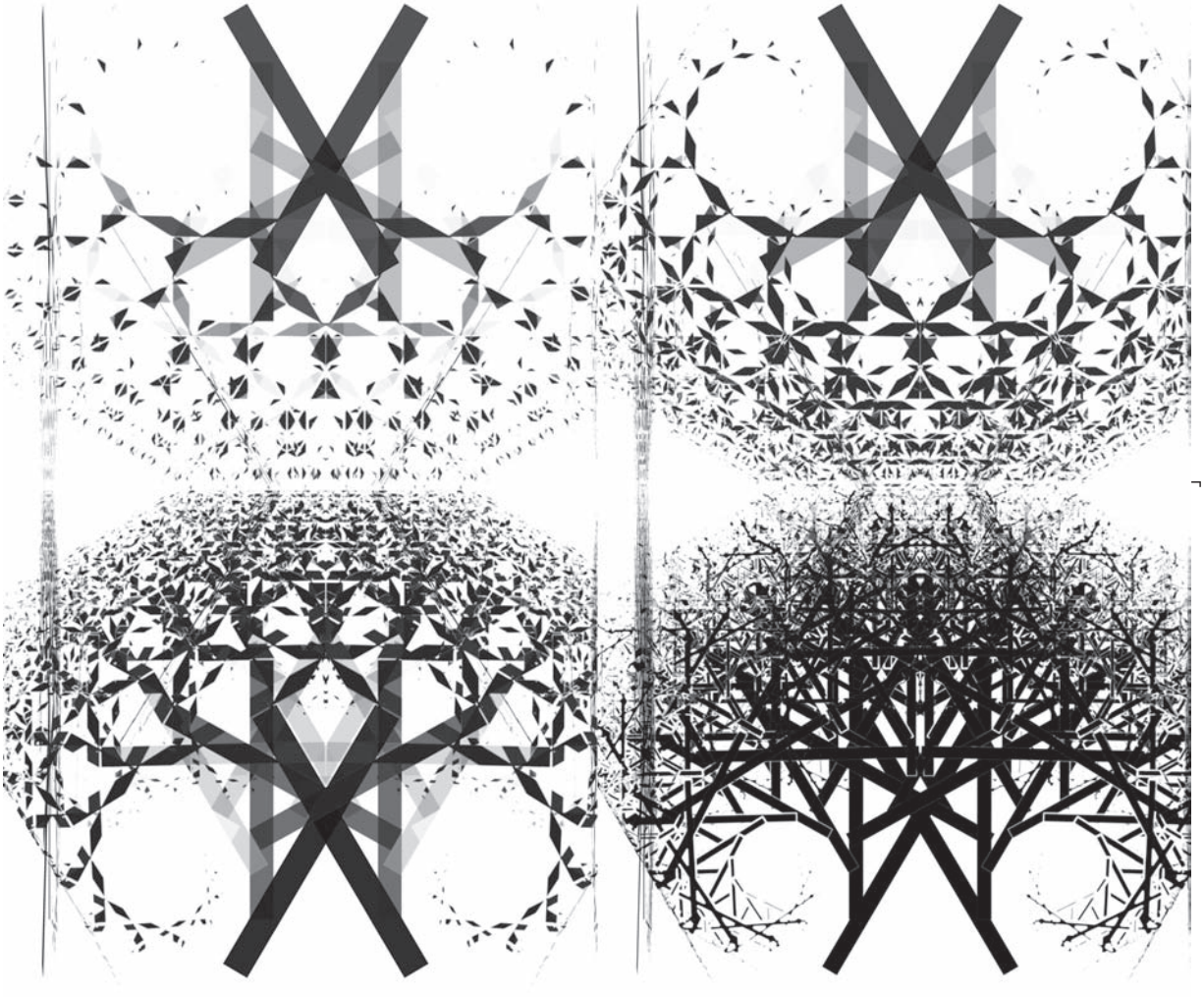
The basic idea was to incorporate the ideas emphasized in the work of Paul Rand ("My interest has always been in restating the validity of those ideas, which, by and large, have guided artists since the time of Polyclitus.") in the process of Generative Design by implementing the principles of aesthetics, form and function in a higher layer of the design process, outside of the designer's brain.

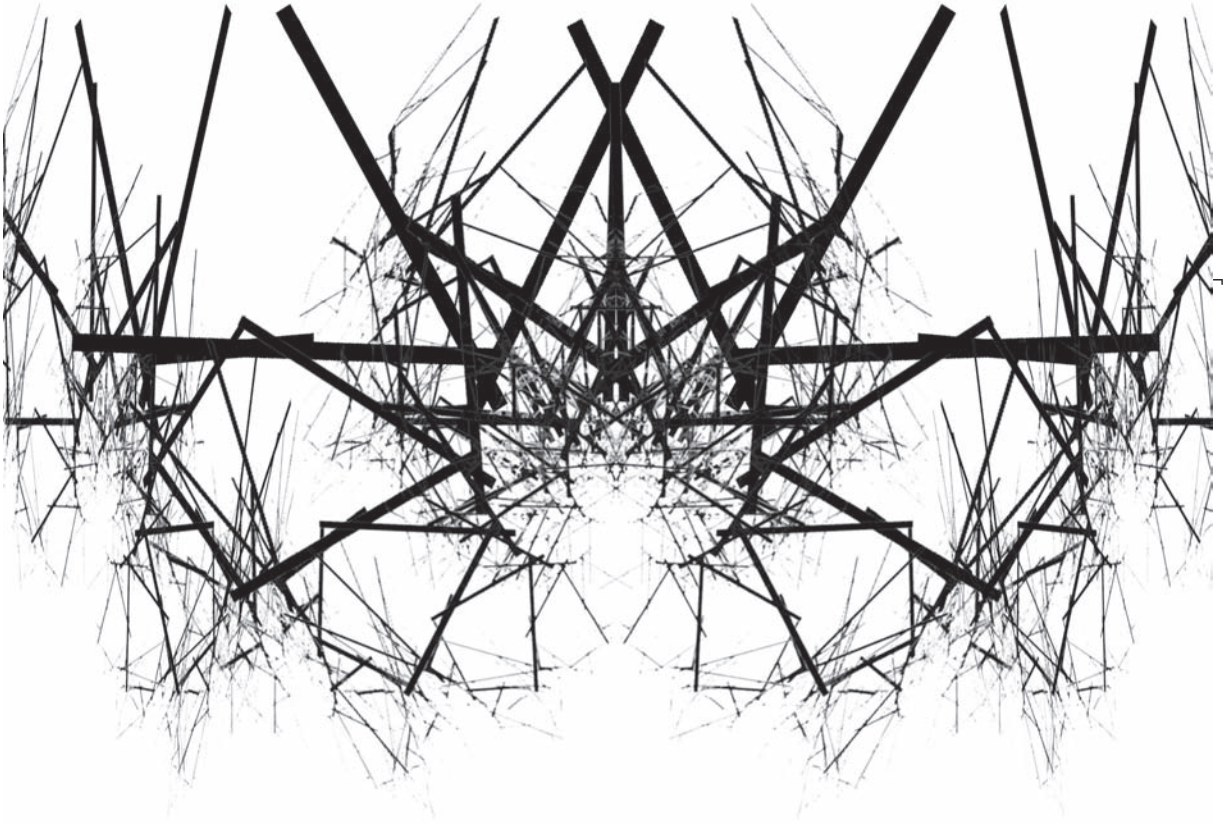
Therefore, Rand's modernist ideology is somehow present in all the images printed in this book, and even if the human mind steps aside and instructs the machine to make graphic designs on its own, the universal demands of aesthetics remain unchanged.

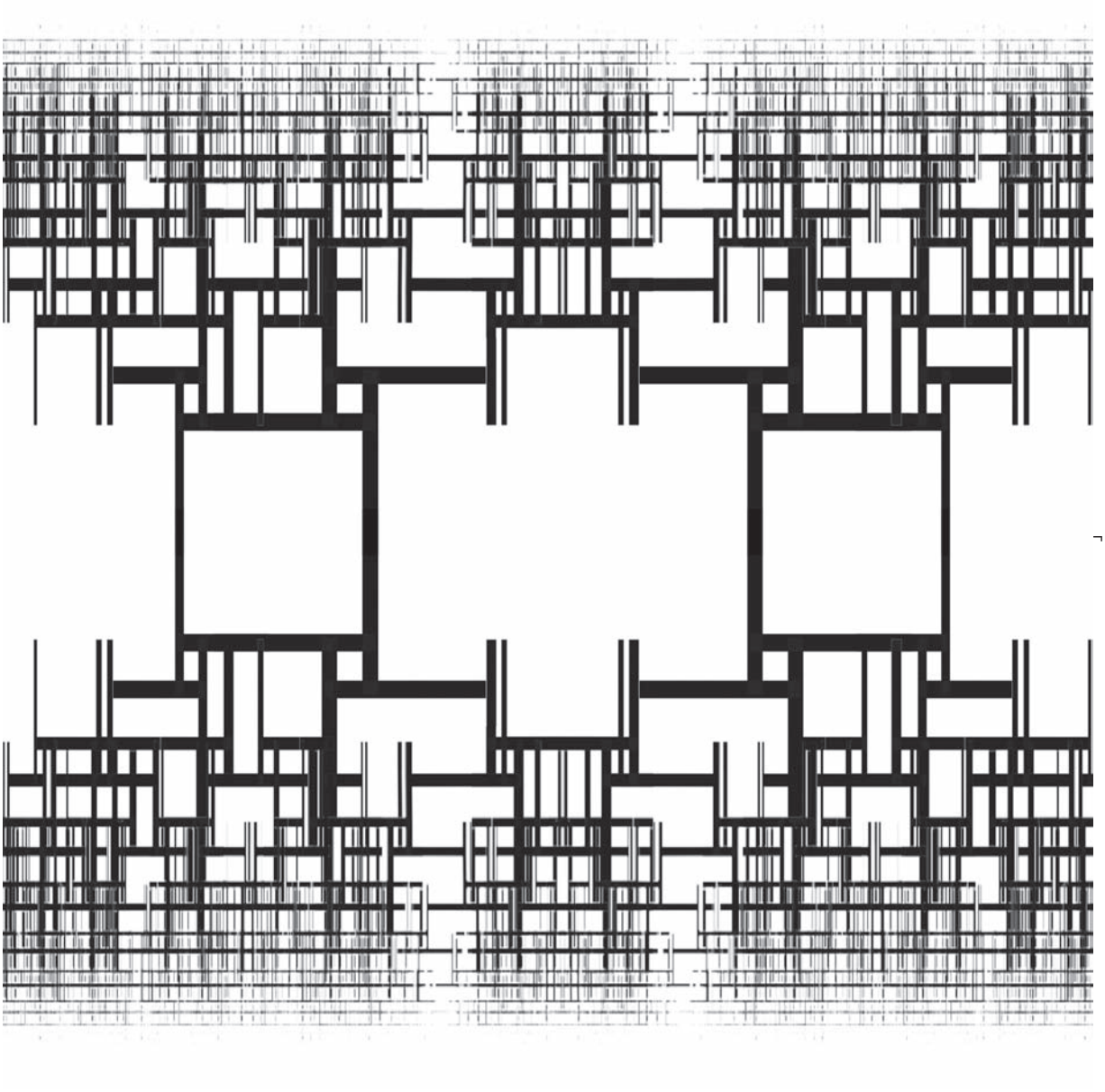
Define the rules and give orders, then leave it to the machine to carry them out before you select the best design(s) for your purpose. The images in this book are created solely with the graphical programming environment vvvv (vvvv.meso.net) and none of them has been edited in any way, except for cropping or resizing. At this point there is no need for further explanations, as the images should speak for themselves and everyone has to decide for themselves if the goal has been achieved.

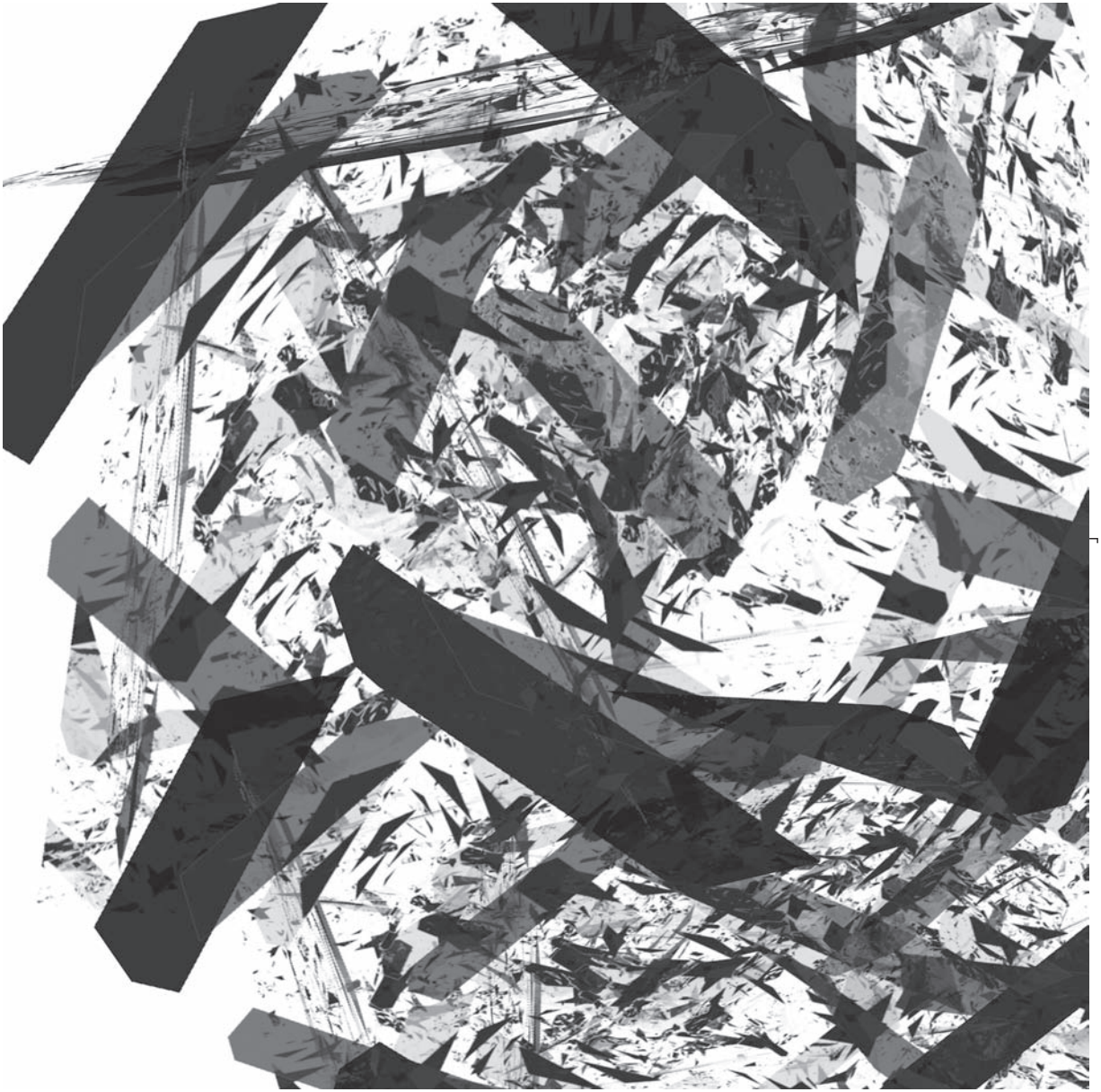
On a final note, regards to all the nice folks at meso (www.meso.net) and thanks for the great opportunities they and their software have to offer.

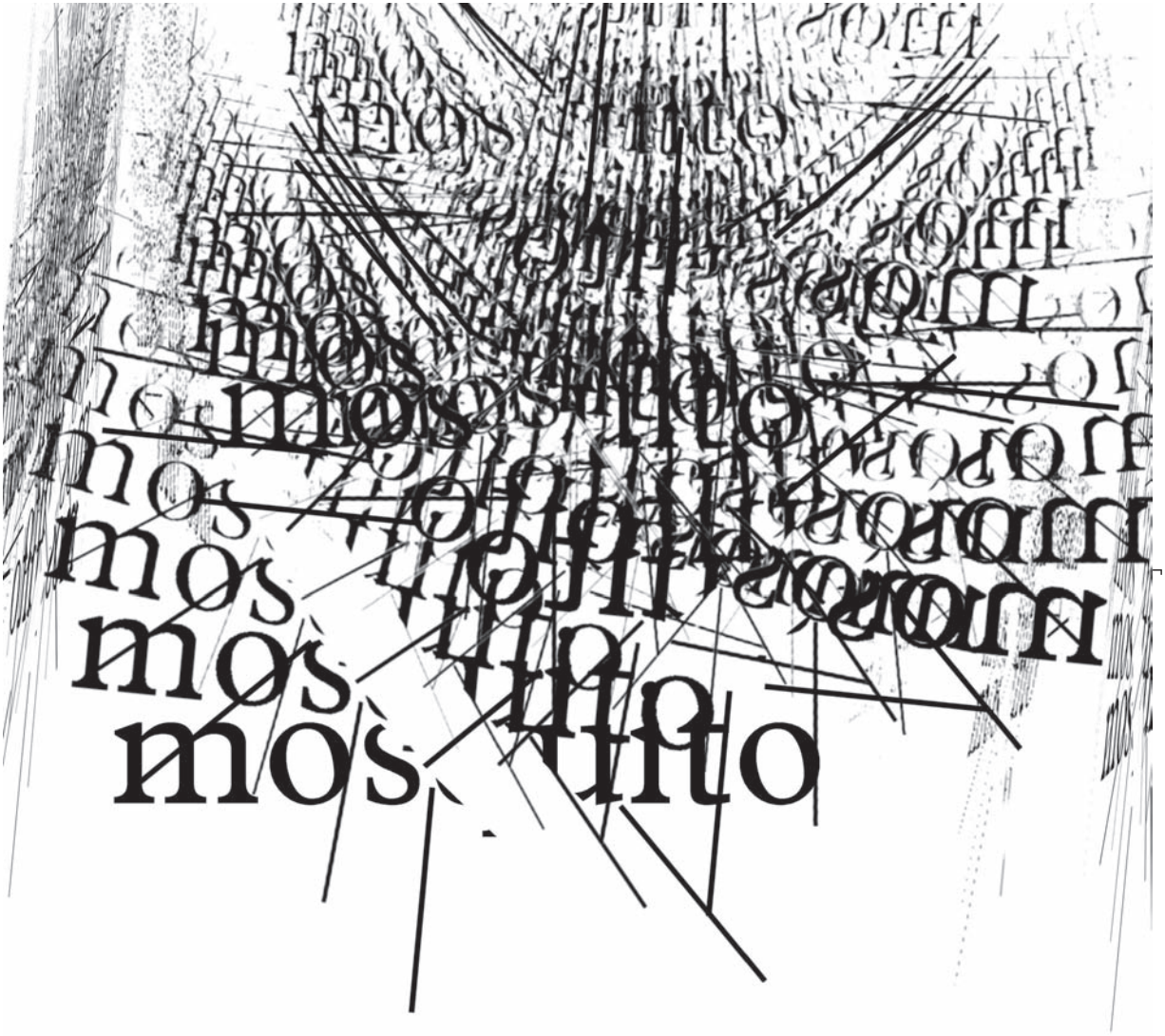
Vorwort des Werkteils

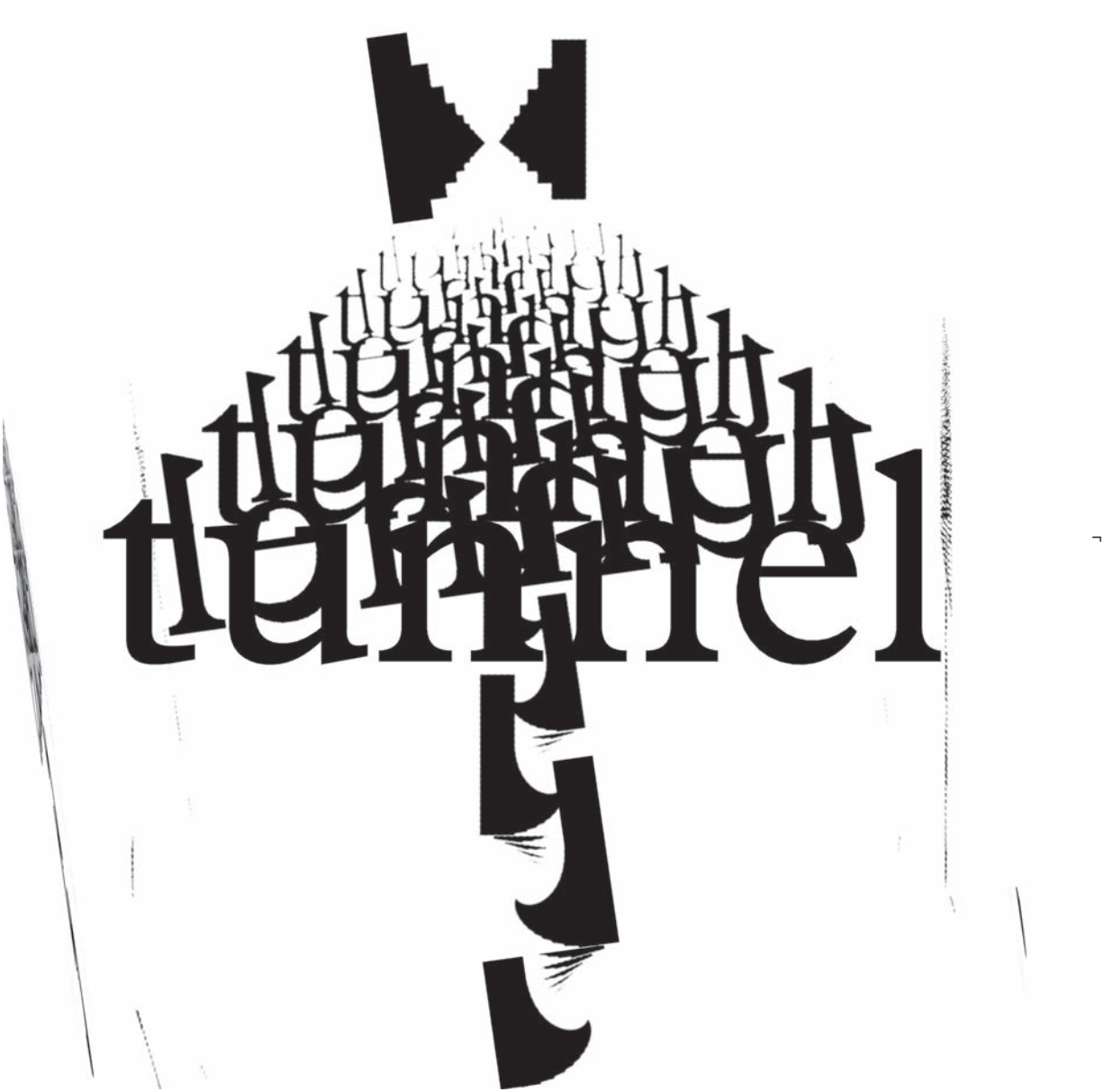


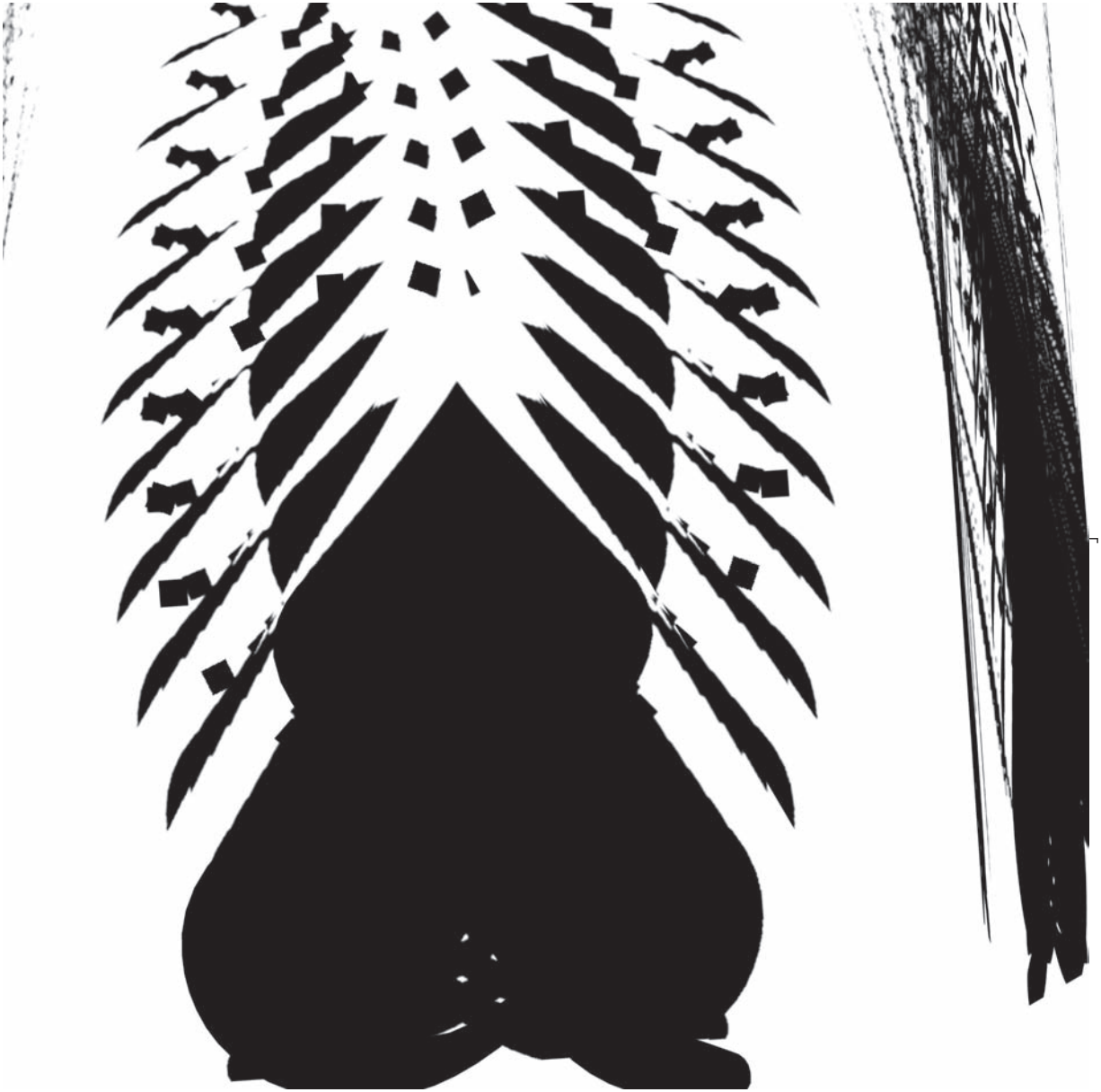


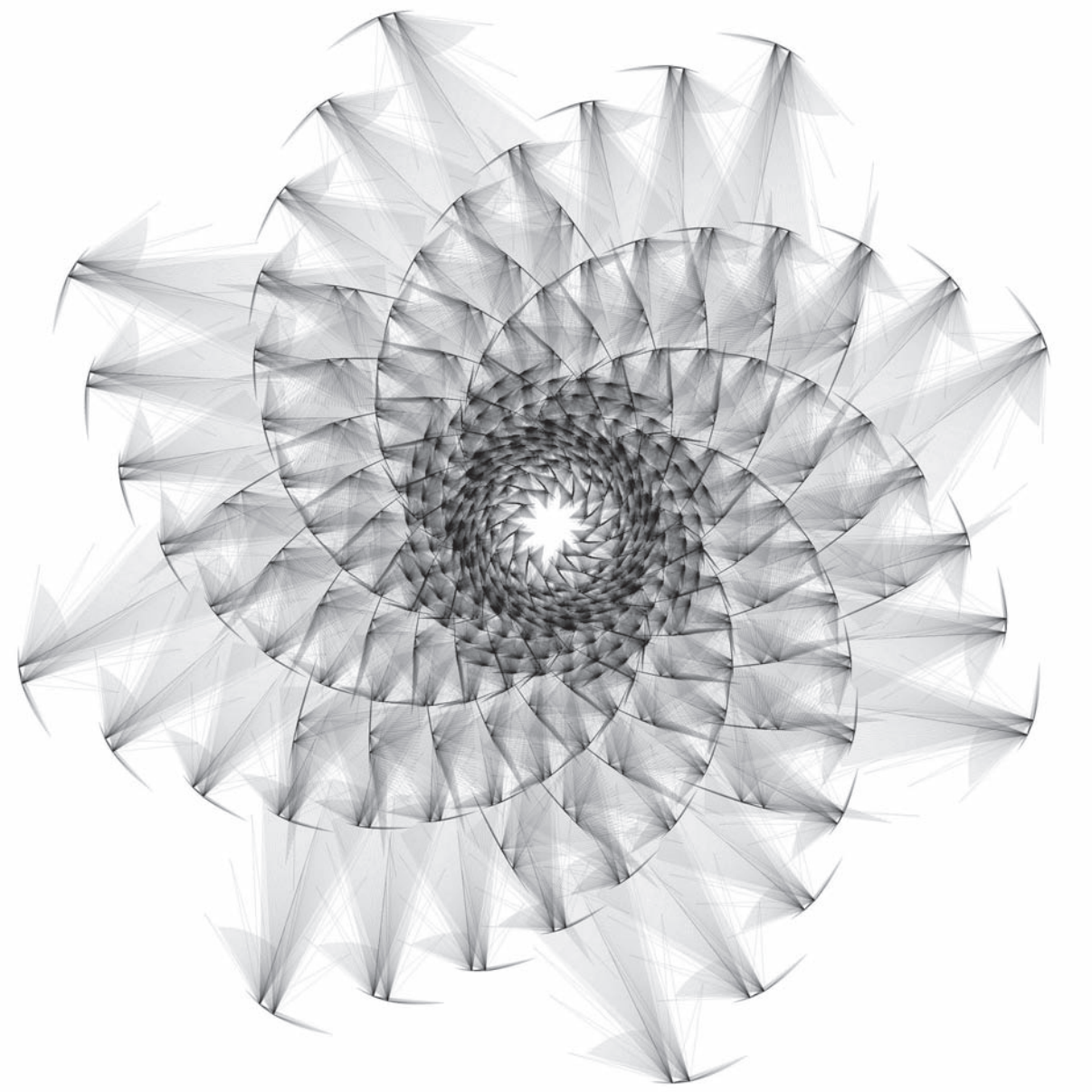


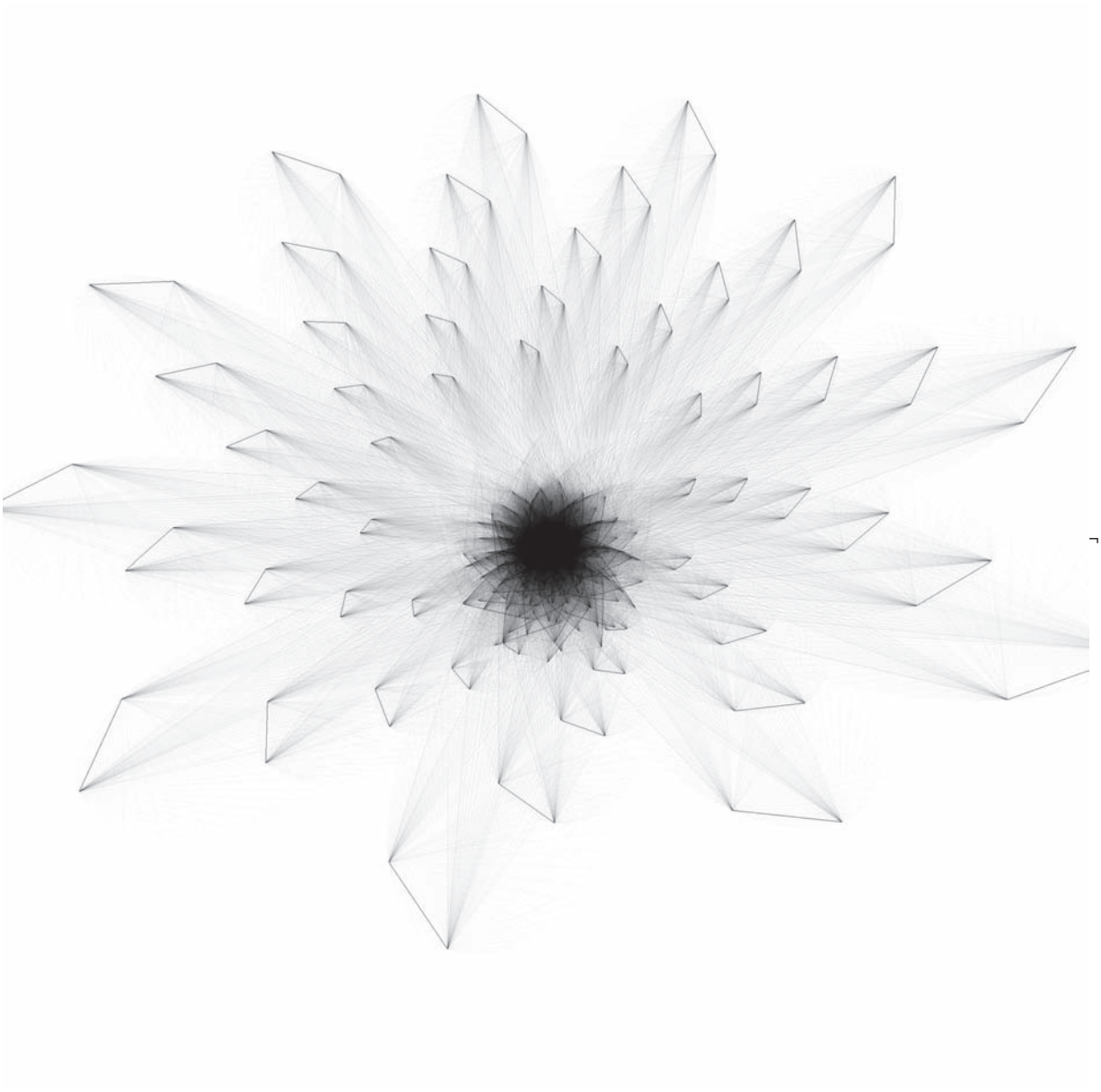


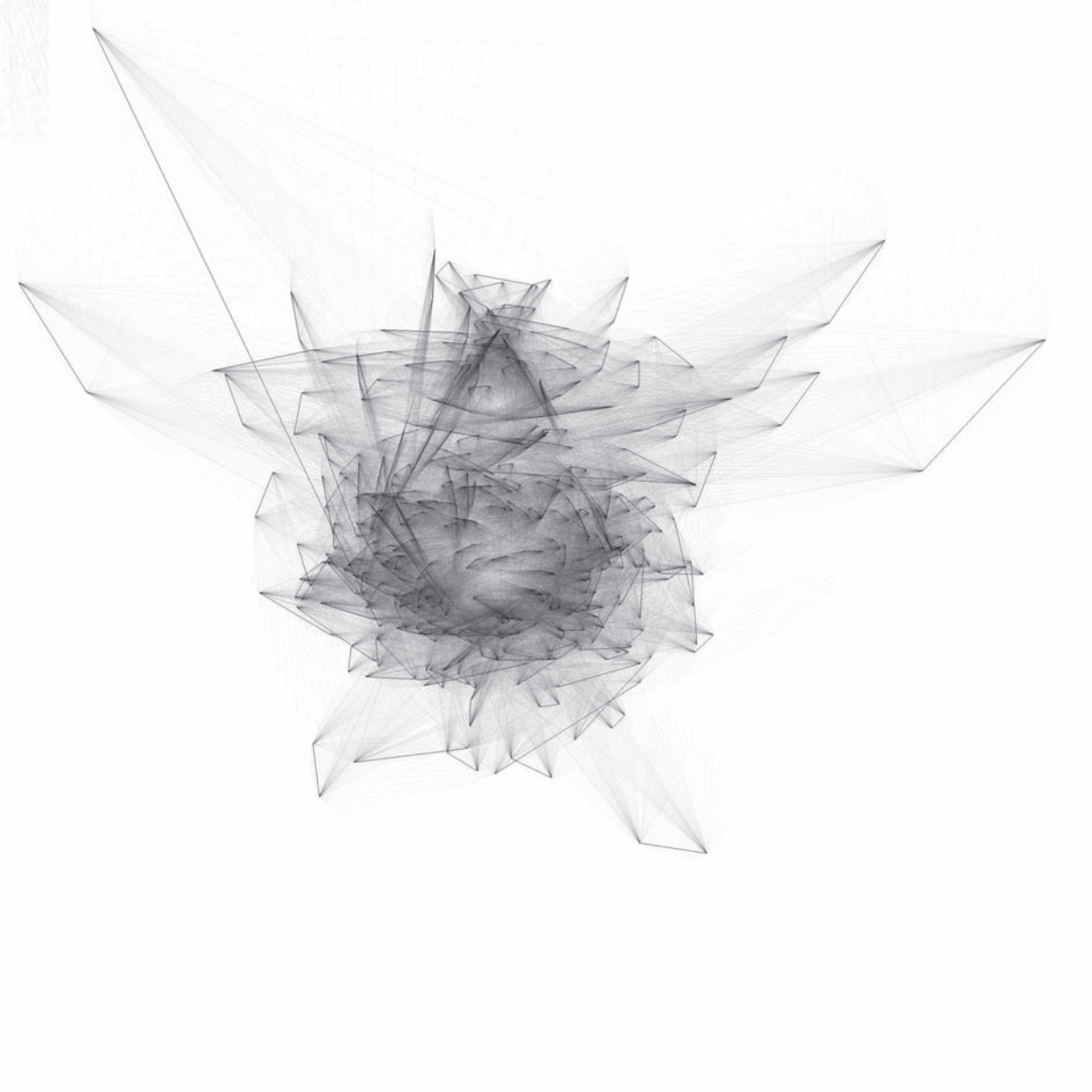

mos
mos
mos
mos

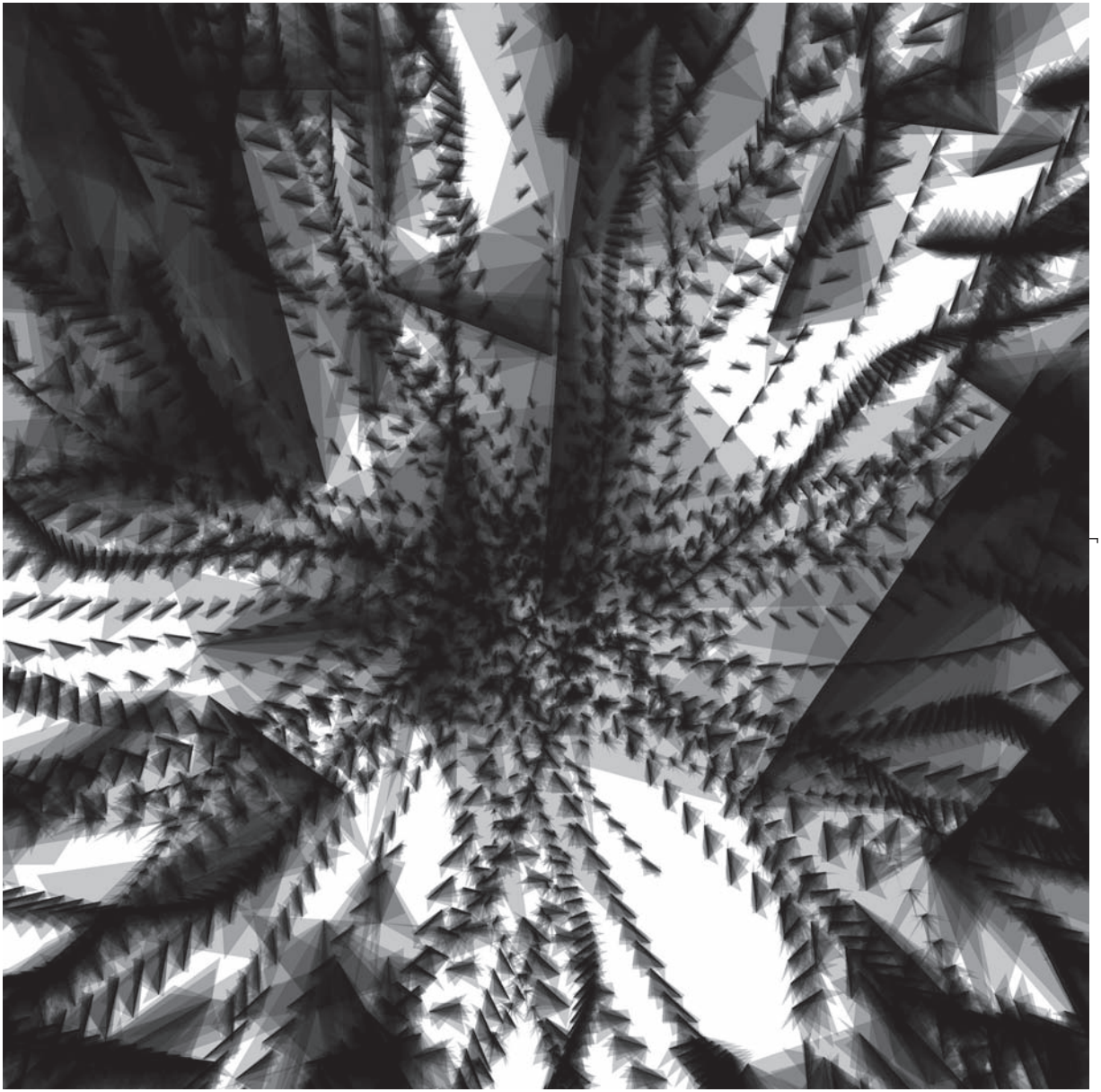

tunnel

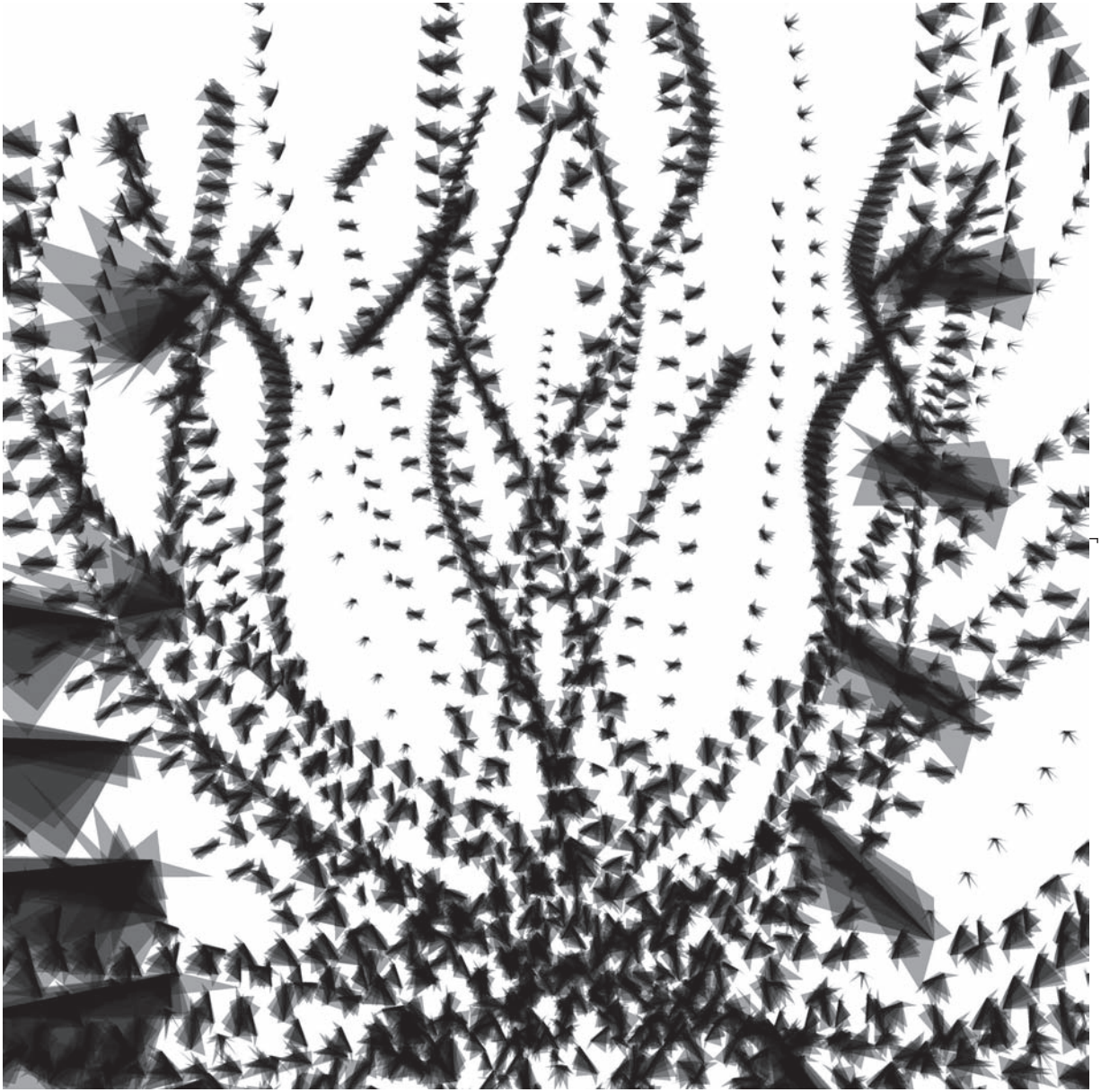


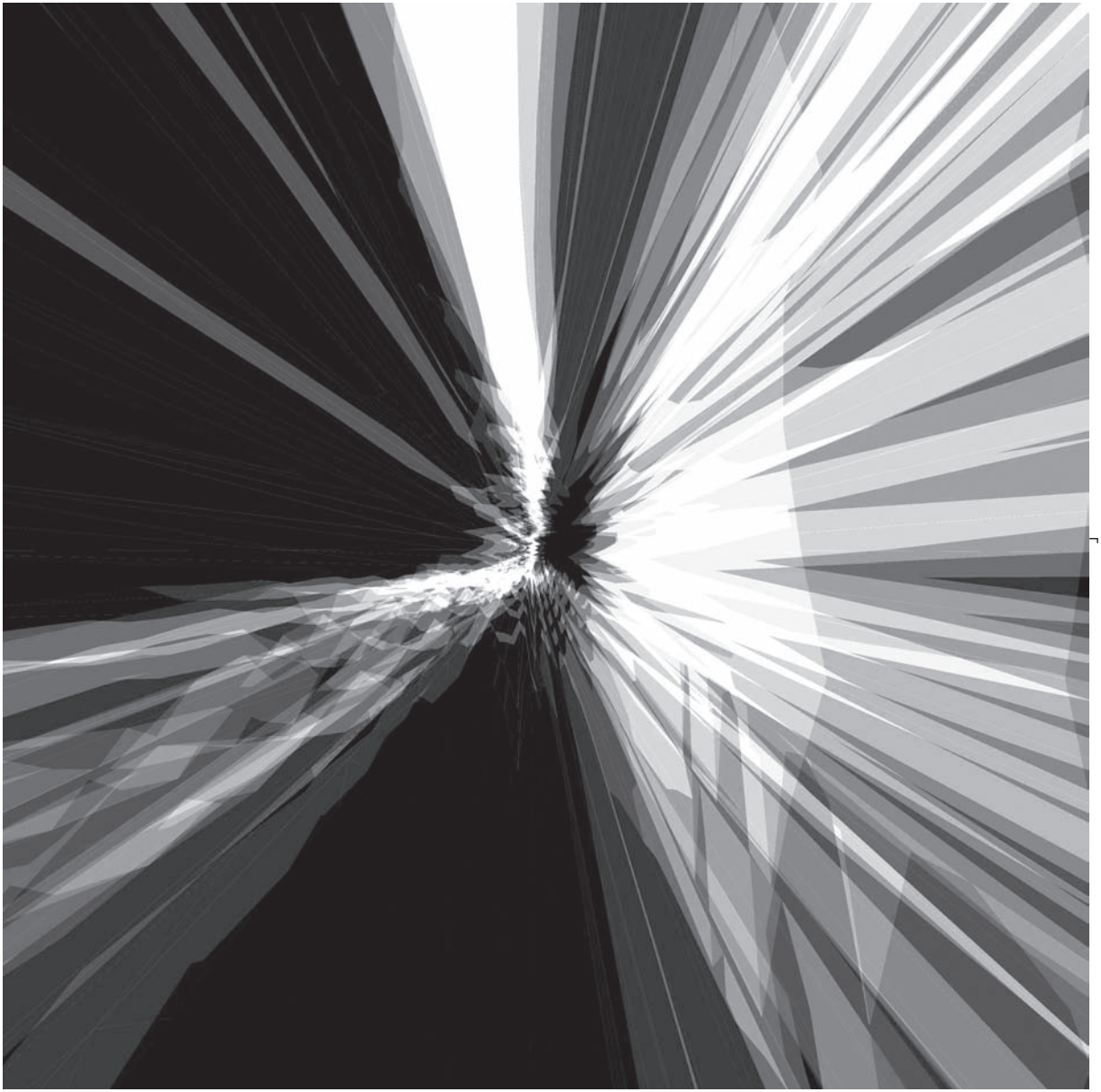




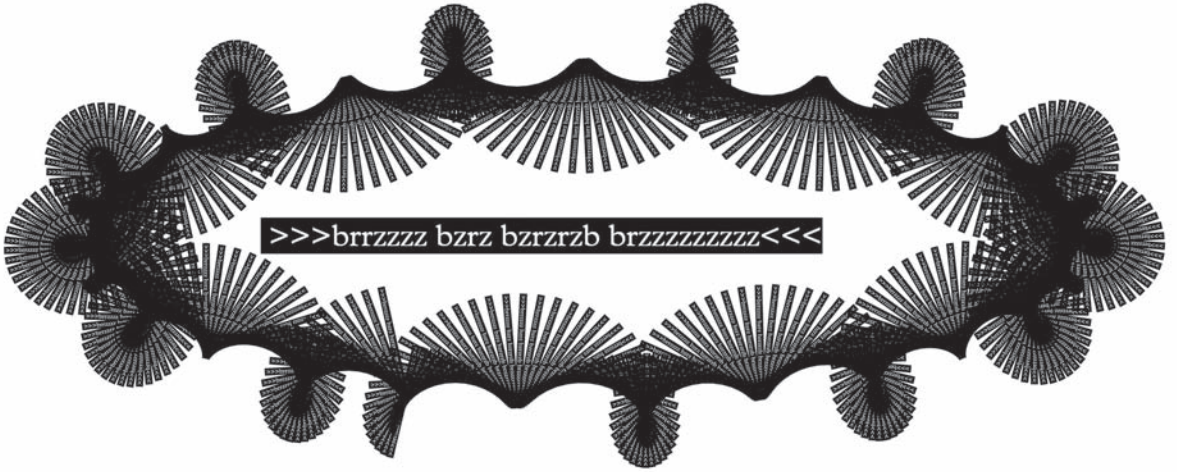


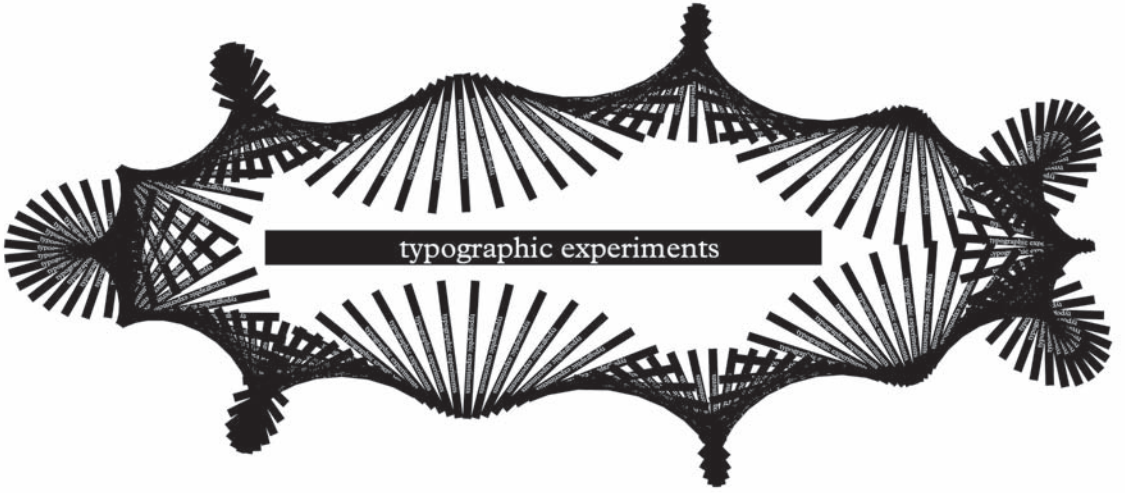







sex pistols typo tree


>>>brrzzzz bzrz bzrzrzb brzzzzzzzzz<<<


typographic experiments


STRUKTVISUALNETWORK
STRUKTVISUALNETWORK
STRUKTVISUALNETWORK
STRUKTVISUALNETWORK
STRUKTVISUALNETWORK