Hochauflösender Export aus 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.-Designer Christian Süß

Wien, 07.11.2005

Eidesstattliche Erklärung

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, 07.11.2005

Thomas Hitthaler, Matrikelnummer : 0110055017

Kurzfassung

vvvv ist eine grafische Programmierumgebung für die Generierung von interaktiven Videobildern. Für diese Software wird eine Methode vorgestellt, die es erlaubt, direkt aus vvvv die damit erzeugten Grafiken in beliebiger Größe zu exportieren.

Da vvvv ausschließlich Vektorobjekte zur Bildgenerierung verwendet, können diese beliebig vergrößert werden, ohne an Qualität zu verlieren. Dieser Umstand wird ausgenutzt, um die angestrebte Auflösungsunabhängigkeit im Export zu erreichen. Durch eine hard- und softwaretechnische Größenbeschränkung beim Rendering, muss ein zu exportierendes Bild jedoch zuerst in mehrere, gitterförmig angeordnete Abschnitte geteilt werden, damit diese anschließend sequenziell gerendert und auf die Festplatte geschrieben werden können. Wie dies geschehen kann, wird ausführlich erklärt und das dazugehörige vvvv Programm in allen Einzelheiten besprochen. Da diese Methode eine gewisse Anzahl von sequenziell nummerierten Einzelbildern liefert, müssen diese noch zu einem einzelnen, großen Bild zusammengefügt werden. Dieser Schritt ist mit vvvv wegen seiner systemisch bedingten Einschränkungen nicht zu realisieren. Deswegen wird ein separates Programm benötigt, das die Fähigkeit hat, beliebig viele Einzelbilder zu einem einzigen Bild zusammenzufügen. Da für diese Aufgabe kein dem Autor bekanntes und frei verfügbares Programm existiert, wird ein für diese Aufgabe in Delphi programmiertes Tool vorgestellt und erklärt. Die Funktionsfähigkeit der vorgestellten Konzepte und Lösungen wird anhand eines Bildbandes demonstriert, der generativ erzeugte und aus vvvv exportierte Grafiken enthält.

Das produzierte Exportmodul ist mittlerweile in der offiziellen vvvv-Distribution enthalten und steht auf der Entwicklerhomepage (vvvv.meso.net) zum kostenlosen Download bereit.

Abstract

vvvv is a graphical programming environment designed for the generation of interactive video installations. It is the purpose of this thesis to develop a method that allows vvvv to export the generated graphics without any restriction of the image size.

Because only resolution independent vector objects are used to generate images, there are no quality losses when enlarging. This fact is the key element needed to export very large images with vvvv. Due to hard- and software restrictions it is inevitable to split a file into multiple parts prior to exporting it. How this can be done and the resulting vvvv-patch are discussed in detail.

The used method produces a certain number of images that have to be combined to create one big single image. This is not possible with vvvv, due to the limited export size. Because of this, another program has to be created to stitch the images together. Despite some extensive research, no suitable tool could be found for this task. A program had to be developed from scratch, it will also be discussed in this thesis. It is able to combine any number of images into one big image.

The functionality of the discussed topics is shown by the creation of the accompanying illustrated book. It presents graphics produced and exported with vvvv. The created export module is already included in the official vvvv-distribution which is available on the developer's webpage (vvvv.meso.net)

Inhaltsverzeichnis

Hochauflösender Export aus vvvv

{0} Einleitung

Die Motivation zu dieser Arbeit war der Wunsch, Print Designerinnen und Designern ein neues Werkzeug zu erschließen, das Aufgaben erledigen kann für die „normale“ Grafikprogramme nicht geeignet sind. Dieses Werkzeug trägt den Namen „vvvv“ und ist eine Programmierumgebung, die für die Erzeugung von interaktiven Grafiken für Videoinstallationen konzipiert wurde. Die vielfältigen Funktionen, die es zur Generierung von komplexen Objekten besitzt, finden sich in keinem der klassischen Designtools (Photoshop, Illustrator, usw.)! Dies ermöglicht Resultate die sonst nur schwer realisierbar wären.

Mit vvvv können Grafiken erstellt werden, die nicht wie üblich aus einzelnen manuell bearbeiteten und positionierten Objekten bestehen: Objekte werden automatisch berechnet und positioniert. Wird die Ausgangssituation verändert, verändert sich auch das Ergebnis der Berechnungen und somit das Aussehen des erzeugten Bildes. Dies eröffnet die Möglichkeit, „one-of-a-kind“ Designobjekte (beispielsweise Flyer oder Visitenkarten) für den Digitaldruck zu produzieren. Anstatt 2.000 mal den selben Flyer zu drucken, werden 2.000 verschiedene Flyer je einmal gedruckt. Alle sehen sich ähnlich und trotzdem ist jedes Druckstück einzigartig. Geht einer dieser Flyer verloren, kann er nur durch erneutes Ausdrucken ersetzt werden. Existieren jedoch die digitalen Rohdaten nicht mehr, ist eine Reproduktion praktisch unmöglich. Somit erhält jedes einzelne Druckstück einen Wert zurück, der durch die technische Reproduzierbarkeit verloren ging: die Einzigartigkeit.

vvvv besitzt die Fähigkeit, Einzelelemente zu komplexen Mustern zu arrangieren und Variationen eines Grundmusters zu erzeugen. Dazu müssen lediglich Regeln vorgegeben werden die beschreiben, wie dies geschehen soll. Die Maschine kümmert sich dann um die Anwendung dieser Regeln und erzeugt daraus Grafiken. Diese Regeln müssen „programmiert“ werden und da vvvv eine gänzlich andere Bedienstrategie verfolgt als die klassischen Designtools ist der Einstieg nicht trivial.

Leider kann in dieser Arbeit nicht behandelt werden, wie sich Grafiken mit vvvv erstellen lassen. Eine vollständige Dokumentation zu diesem Thema bietet die Homepage der Entwickler: vvvv.meso.net.
Dort findet sich eine umfangreiche Beschreibung aller Funktionen und zusätzliche Tutorials zum schnellen Einstieg.

Die vorliegende Arbeit befasst sich mit der Konzeption und Erstellung eines Exportmodules, das die erzeugten Grafiken in beliebiger Größe auf einen Datenträger speichern kann, um diese auch für großformatige Printdesigns in hoher Auflösung verwenden zu können. Dies ist ohne dieses Modul unmöglich, da vvvv gewissen Einschränkungen für die direkt berechenbare Bildgröße unterliegt. Das beiliegende Werk wird demonstrieren, dass dieses Modul funktioniert und zeigen, wie die mit vvvv erzeugten Grafiken aussehen können.

Durch die Einbindung dieses Exportmodules in eine offizielle Distribution von vvvv könnte dessen Einsatzgebiet ausgeweitet werden, sodass es neben Videoinstallationen auch für Printdesigns eingesetzt werden kann.

0\1\ Lesetechnische Hinweise

Dieses Werk benutzt verschiedene Textformatierungen, um unterschiedliche Textarten voneinander zu trennen. So wird Quelltext in einer anderen Schriftart gesetzt als normaler Text.

Der Fließtext wird immer dann *kursiv* gesetzt, wenn der Text ein Zitat ist und immer unterstrichen wenn der Text auf ein programmtechnisches Objekt verweist. Dieses Objekt kann entweder ein Knoten in einer grafischen Programmierumgebung oder eine Prozedur in einer textbasierten Programmiersprache sein. Ein Beispiel: Writer (EX9.Texture Grid) ist ein Verweis auf den gleichnamigen vvvv-Knoten, btnRunClick ein Verweis auf die gleichnamige Prozedur des Quellcodes in Anhang A|2. Auch Pins von Knoten und Variablen werden unterstrichen.

```
Wird im Quelltext ein Textabschnitt unterstrichen, dann bedeutet das, dass eine Funktion
oder Prozedur an dieser Stelle aufgerufen wird. Die Anfangs- und Endzeile jeder Prozedur ist
fett geschrieben. Kursiv gesetzter Text ist immer ein Kommentar, der lediglich zur Erklärung
der Programmabläufe dient.
```

{1}

Auflösung von Bilddateien

Da sich diese Arbeit mit der Erstellung von hochauflösenden Bilddateien befasst, muss als erstes geklärt werden, was das Wort „Auflösung“ beschreibt. Jedes vom menschlichen Auge wahrgenommene Bild enthält Informationen, die dem Gehirn als Ausgangsbasis für die Wahrnehmung dieses Bildes dienen (vgl. Bruckmann 1981, 97). Der Inhalt dieses Bildes entspricht dem Gesichtsfeld des Menschen und den darin enthaltenen Objekten. Die Informationen die dabei vom Auge verarbeitet werden sind Licht und Farbe, welche von spezialisierten Zellen, den Photorezeptoren, wahrgenommen werden.

Da sich nur eine endliche Anzahl dieser Rezeptoren auf der Netzhaut des Auges befindet, ist die Menge an Informationen die wahrgenommen werden können, begrenzt. Ernst-Georg Beck definiert diese Beschränkung so:

> *„Das Auflösungsvermögen misst die minimale Trennung zweier Objekte, die man gerade noch getrennt erkennen kann; beim Auge 0,2 mm“ (Beck 2005, 1.1.3)*

So werden z.B. zwei dicht nebeneinander liegende Linien ab einer gewissen Entfernung als nur eine Linie wahrgenommen (vgl. Bruckmann 1981, 97). Die Distanz ab der dies geschieht ist von der Sehstärke des Betrachters abhängig und somit bei jedem Menschen unterschiedlich.

So wie die Auflösung bei der Wahrnehmung die Anzahl der in einem Bild enthaltenen Details definiert, bestimmt auch die Auflösung eines Reproduktionsmechanismus (z.B. Druck oder die Darstellung auf Bildschirmen) den Detailreichtum von Bildern. Je mehr Farbpunkte in einer bestimmten Fläche enthalten sind, desto mehr Informationen (Bilddetails) können vom Auge wahrgenommen werden.

1|1| Auflösung in der Drucktechnik

Die in der Drucktechnik gebräuchliche Einheit zur Definition der Auflösung eines Druckverfahrens ist Linien pro cm bzw. Inches in angloamerikanischen Ländern. Die Einheit Linien pro cm wurde durch die moderne Fotolithografie (erfunden im Jahre 1880 von Georg Meisenbach) eingeführt, welche die manuelle Holzstichtechnik für den Bilderdruck ablöste. Bei der Fotolithografie wird ein Bild mit Hilfe eines Rasters in mehr oder weniger große Punkte zerlegt. Diese werden auf einen fotografischen Film belichtet und von dort auf eine Druckform übertragen. Wie diese Druckform hergestellt wird, ist bei allen Druckmethoden (Hoch-, Tief- und Flachdruck) verschieden, entscheidend ist nur, dass der fotografische Film als Ausgangsbais für die Anfertigung der Druckform verwendet wird. Die eingefärbte Druckform kann schließlich auf ein Trägermedium (meist Papier, aber auch andere Materialien wie Stein, Holz, Plastik, Glas, usw.) gedrückt werden.

Das optische Auflösungsverfahren der eingesetzten lichtempfindlichen Schichten ist in der Fotolithografie ein begrenzender Faktor für die mögliche Dichte der Bildinformationen pro cm. Die eingesetzte Drucktechnik und vor allem die Druckformherstellung beeinflusst die Maximalauflösung von Druckverfahren genauso wie das verwendete Papier. So kann auf herkömmlichem Zeitungspapier nur mit relativ groben Rastern (25 - 28 Linien pro cm) gearbeitet werden, während auf feinem Druckpapier mit hochglänzender Oberfläche sehr feine Raster gedruckt werden können. Die gebräuchlichen Rasterweiten von Druckverfahren bewegen sich deswegen zwischen 25 und 120 Linien pro cm. Das entspricht 625 Bildpunkten pro cm^2 bei 25 Linien pro cm (Grobraster) und 14.400 Punkten pro cm^2 bei einem 120er Raster.

Bis zum Durchbruch der Computertechnik in Druckereien kamen die Daten für den Bilderdruck von Fotografien. Die Auflösung und Größe dieses fotografischen Ausgangsmaterials war also eine Grenze für die Detailmenge auf den Druckstücken. Genauso verhält es sich bei den nun verwendeten digitalen Daten. Diese müssen für einen hochqualitativen Druck mindestens ebenso viele Details enthalten wie sie das druckende Verfahren reproduzieren kann. Hier ist zu beachten, dass in der Computertechnik die englische Sprache vorherrscht und deswegen meistens mit Inches (Zoll) gerechnet wird. Ein Inch entspricht 2,45 cm und die Einheit wird mit dpi (Dots per Inch) abgekürzt.

Anzumerken ist noch, dass sich das gesamte Kapitel 1 | 1 auf verschiedene Abschnitte in Bruckmanns's Handbuch der Drucktechnik (Bruckmann 1981) auf den Seiten 97f, 109f, 189ff und 212f stützt.

1\2\ Auflösung von Bildschirmen

Um digitale Daten für den Druck anzufertigen und aufzubereiten, wird heute auf Bildschirmen gearbeitet und da drucktechnische Verfahren grundsätzlich anders arbeiten, gibt es hier andere Faktoren, welche die maximale Detailanzahl eines Bildes bestimmen. Jeder Bildschirm, egal ob Flüssigkristalldisplay (LCD-Bildschirm) oder Kathodenstrahlröhre kann eine gewisse Gesamtanzahl von Bildpunkten auf seiner Bildfläche darstellen. Bei LCD Bildschirmen bestimmt die Größe der einzelnen Leuchtdioden die Größe der Bildpunkte, bei Kathodenröhren die sog. Lochmaske. Nähere Informationen hierzu finden sich in „Druck & Medien Technik“ von Helmut Teschner (vgl. Teschner 2003, 8.50ff)

Aus der Pixelanzahl, die ein verwendeter Bildschirm darstellen, kann lässt sich die Auflösung berechnen, indem die Anzahl dieser Bildpunkte durch die Abmessungen des Bildschirms dividiert wird. Im Gegensatz zur linienorientierten Drucktechnik wird die Auflösung von Bildschirmen in Punkten pro Zentimeter bzw. Zoll angegeben.

Ein Beispiel: Ein Bildschirm, der 41 cm breit und 33 cm hoch ist und 1280 x 1024 Pixel darstellen kann, hat eine Auflösung von 75 dpi. 1280 Pixel / 41 cm x 2,45 = Auflösung in Pixeln pro Zoll.

Damit liegt die Auflösung von Bildschirmen weit unter der Auflösung von hochauflösenden Druckverfahren. Dies muss beachtet werden und wird im folgenden Kapitel näher erörtert.

1|3| Vom Bildschirm zum Druckstück

Soll ein computergeneriertes Bild über ein Druckverfahren reproduziert werden, muss das Bild eine gewisse Anzahl von Bildpunkten enthalten. Ein Beispiel: Ein Druckstück soll mit 300 dpi auf ein DIN-A4 Blatt (29,7 x 21 cm) gedruckt werden. Um die Auflösung des Druckverfahrens ausnutzen zu können, muss die Vorlage am Computer in einer Größe von mindestens 3508 x 2480 Pixeln angelegt werden.

Die Formel um diese Größe zu berechnen ist: Kantenlänge x Rasterauflösung = Pixelanzahl.

Die meisten Bildbearbeitungsprogramme berechnen die nötige Bildgröße in Pixeln automatisch, wenn die Auflösung und die gewünschte Ausgabegröße eingegeben werden, wie Abbildung 1 zeigt.

Da ein Bildschirm, wie in Kapitel 1 | 2 gezeigt, fast immer eine andere Auflösung besitzt als das zu verwendende Druckverfahren, ist ein Bild auf dem Bildschirm nie so groß wie auf dem Druckstück. Deswegen muss das Bild auf dem Bildschirm vergrößert werden damit alle Details dargestellt werden können. Das bedeutet auch, dass auf Details in der Darstellung verzichtet werden muss, wenn das gesamte Druckstück auf dem Bildschirm Platz finden soll. Wird diesem Umstand Rechnung getragen und ein Bild mit genügend vielen Pixeln angelegt, ist eine hochqualitative Reproduzierbarkeit gewährleistet.

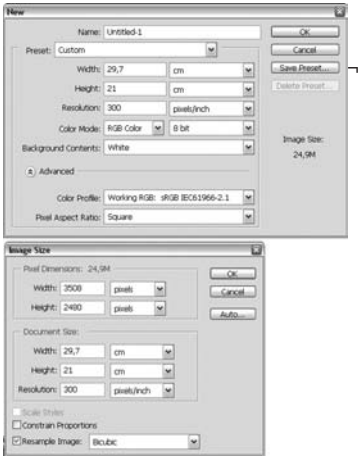


Abb | 1: Auflösungsdialog von Adobe Photoshop.

Da diese Arbeit auf den Export von hochauflösenden Daten aus vvvv fokussiert ist, soll im nächsten Kapitel ein Überblick gegeben werden, was vvvv ist und wie damit Bilder erzeugt werden können.

{2}

vvvv als Grafikwerkzeug

vvvv ist ein Vertreter der sich zur Zeit sehr schnell verbreitenden, visuellen Multimedia-Programmiersprachen (auch „grafische Programmierumgebungen“ genannt), welche eine gänzlich andere Strategie verfolgen als Programmiersprachen mit 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)*

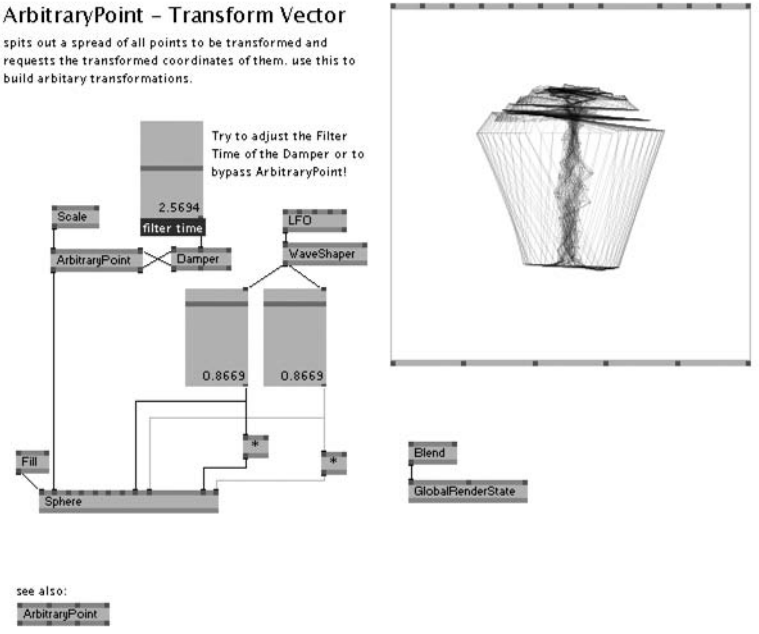
Eine visuelle Sprache ist ein räumliches Arrangement von beschrifteten grafischen Symbolen, die semantisch interpretiert werden müssen, um innerhalb der Programmierumgebung miteinander kommunizieren zu können. Auf diese 3 Definitionen bezieht sich jede Erwähnung einer "grafischen Programmiersprache"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

Die zwei bekanntesten Vertreter dieser grafischen Programmierumgebungen sind: puredata (puredata.org) und max|msp (cycling74.com).

2\1\ Über grafische Programmierumgebungen

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 die Erstellung von Designobjekten eignen. Eine Gemeinsamkeit dieser grafischen Programmierumgebungen ist, dass alle Knoten nach genau festgelegten Regeln miteinander verbunden werden können, was den Knoten erlaubt, untereinander Daten auszutauschen,
wie Abbildung 2 zeigt.

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.



Abb|2: Ein vvvv Programm mit verschiedenen Knoten zur Bildgenerierung.

Dies sind die beiden Grundeigenschaften von grafischen 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. Deswegen heißen Programme von grafischen Programmierumgebungen auch „Patches“ (engl. Flicken aber auch Steckbrett oder Schalttafel).

Andrew Begel, vom MIT Media Lab, schrieb 1996 in einer Abhandlung über eine vom MIT entwickelten grafischen Programmiersprache für Handheld-Computer über die Eigenschaften sowie Vor- und Nachteile von grafischen Programmierumgebungen. Dabei hebt er hervor, dass Zusammenhänge zwischen Knoten auf einer visuellen Logik beruhen, die das Gehirn auf einer anderen, viel direkteren Ebene als bei textbasierten Programmiersprachen interpretieren kann. Während das Vokabular einer textbasierten 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.

Weiters schreibt Begel:

> *„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." (Begel 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 ermöglichen 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 Begel 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."*
> *(Begel 1996, 10)*

Ein Nachteil ist laut Begel, dass die meisten grafischen Programmierumgebungen keine Programmierung über eine textbasierte Syntax erlauben, was für viele fortgeschrittene Programmierer eine Einschränkung darstellt. Für die in dieser Arbeit besprochenen Probleme reicht die grafische Programmierung jedoch aus.

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 Begel als Nachteil an. Ob es eine ähnliche Beschränkung wie das Deutsch Limit auch für textbasierte Programmierumgebungen gibt, bleibt allerdings offen. 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. Wie dies genau funktioniert, wird in Kapitel 5 | 2 gezeigt und von Abbildung 23 illustriert.

Dies ist ein wichtiges Feature, das jede grafische Programmiersprache besitzen sollte: 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.

2|2| Über vvvv

vvvv ist eine Eigenentwicklung der Designagentur meso (www.meso.net). Auf der Webseite vvvv.meso.net findet sich jedoch eine große Menge an Informationen und auch das Programm selbst zum kostenlosen Download. 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)*

vvvv ist also ein Werkzeug für Echtzeit-Videosynthese und zum Verbinden von externen Geräten. „Synthese" deshalb, weil vvvv Videobilder mit Hilfe von bewegten grafischen Objekten generiert. Auf der Webpage findet sich auch eine Liste mit den Funktionen ("Core Features") von vvvv, wobei nur die relevantesten Punkte hier besprochen werden:

> *1. Sophisticated objects for high level animation and problem solving*
>
> *2. Simple handling of a multitude of logical objects (Spreads)*
>
> *3. Runtime graphical programming for easy prototyping and testing*
>
> *4. High-Speed, High-Quality, High-Resolution DirectX9.0 based rendering*
>
> *5. 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 (Punkt eins und zwei) sind in dieser Auflistung die wichtigsten Punkte um vvvv für Grafik Design einsetzen zu können, wie in Kapitel 2|3 noch genauer erklärt.

Punkt 3 beschreibt die unterbrechungsfreie Arbeitsweise von vvvv. „Runtime“ bedeutet, dass es keinen Unterschied zwischen einem Programmier- und einem Ausführungsmodus gibt. Jede Veränderung im Programm 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 vvvv-Programmen erheblich.

Punkt 4 bezieht sich auf schnelles und hochqualitatives Rendering für die Darstellung auf Bildschirmen (Screen Design). Das in dieser Arbeit zu erstellende Exportmodul soll diese Aussage auf die Ausgabe von hochqualitativen und auflösungsunabhängigen Designs für Print Design erweitern. Dadurch würde vvvv nicht länger durch die gängigen Videoauflösungen (ca. 4096x4069 Pixel) eingeschränkt.

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.

Wie aus Abbildung 3 ersichtlich, besteht ein vvvv-Programm (Patch) 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. 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, für sich betrachtet, nicht besonders mächtig sind, lassen sich durch eine kreative Kombination dieser Knoten äußerst komplexe Vorgänge berechnen und darstellen.

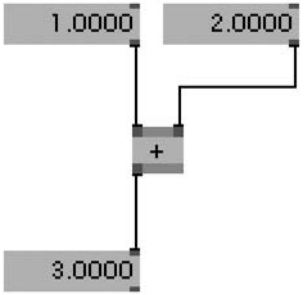


Abb | 3: Der Additionsknoten von vvvv.

Jeder Knoten, von denen es bisher über 300 verschiedene Typen gibt, bearbeitet die Daten, die von anderen Knoten übergeben werden, auf eine vorgegebene Art und Weise bevor das Ergebnis über seine Ausgänge an andere Knoten weitergereicht wird.

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

* Analyse von Daten (Bilderkennung, Audioanalyse, Sensoren…)
* Rendering und Shading von importierten 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|3| Erzeugen und manipulieren von Objekten mit vvvv

Zur Generierung von Design muss ein Designwerkzeug diverse Objekte bereitstellen, die auf einer Arbeitsfläche angeordnet werden können. Dies geschieht mit vvvv nur selten per Hand, da seine große Stärke darin besteht, große Objektmengen in regelbasierte Strukturen zu verwandeln. Die darstellbaren Grundformen werden in Kapitel 3 genau beschrieben und auch Knoten, die Positionen und Aussehen von Objekten manipulieren werden nun kurz vorgestellt. 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 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 4 zeigt ein Beispiel für einen Knoten LinearSpread (Spreads), 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 in das schwarze Ausgabefenster zeichnet. Abbildung 5 zeigt einige Knoten, die vvvv zur Erzeugung von Spreads bereithält und welche Formen diese erzeugen. Dies sind nur sehr einfache Beispiele und Komplexität wird vor allem durch Verknüpfung dieser Knoten untereinander und mit anderen Knoten erreicht.

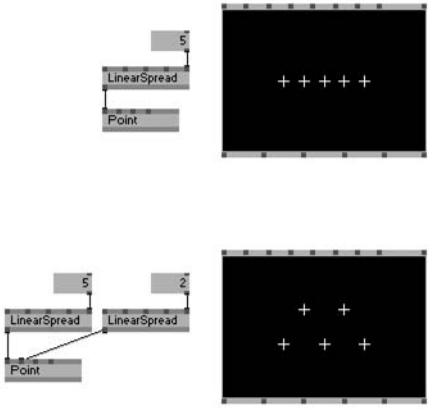


Abb | 4: Linear Spread x und Linear Spread xy.

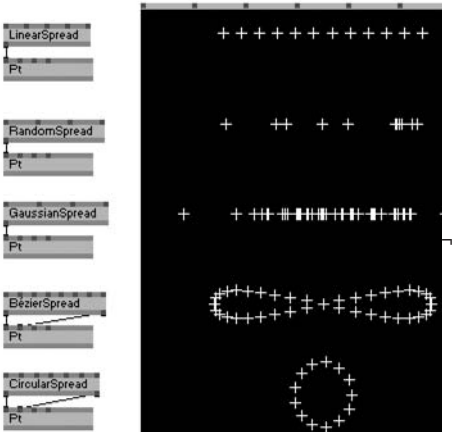


Abb | 5: Verschiedene Arten von Spreads: Linear, Random, Gaussian, Bézier, Circular, Typo.

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.

Die mit vvvv erzeugten Objekte besitzen bestimmte Eigenschaften, welche die hochauflösende Ausgabe beeinflussen können. Darauf wird im nächsten Kapitel eingegangen.

{3}

Zeichenbare Objekte in vvvv

Bevor die Modalitäten eines auflösungsunabhängigen Export aus vvvv besprochen werden können, muss erst geklärt werden, welchen Einschränkungen die unterschiedlichen Objekttypen, die mit vvvv generiert werden können, unterliegen.

Für den auflösungsunabhängigen Export muss zwischen Vektordaten und Bitmapdaten unterschieden werden: während Vektordaten ohne Qualitätsverlust unbegrenzt vergrößert werden können, ist jede Bitmap Datei, abhängig von ihrer Auflösung, nur bis zu einem gewissen Grad vergrößerbar. Detaillierte Information zu Vektor- und Bitmapdaten finden sich in „Druck & Medien Technik“ im Kapitel „Digitalisierung: Vektorgrafik oder Pixelbild“ (Teschner 2003, 8.5f) und im Kapitel „Vektorgrafik: Punkte zum Anfassen“ (Teschner 2003, 8.8ff).

3|1| *Vektordaten*

Vektordaten sind, neben Raster- bzw. Bitmapdaten die zweite gebräuchliche Methode um ein Bild zu beschreiben. Dies geschieht durch mathematische Funktionen in einem Koordinatensystem. Während eine Rastergrafik jeden einzelnen Bildpunkt speichert, besteht eine Vektorgrafik aus Linien, Kurven oder Flächen. Um z.B. einen Kreis zu definieren, benötigt eine Vektorgrafik nur zwei Werte: den Kreismittelpunkt und den Kreisdurchmesser.

Dadurch können Vektorgrafiken im Gegensatz zu Rastergrafiken stufenlos skaliert werden.

vvvv bietet ein breites Spektrum an Funktionen zur Erzeugung von Vektorobjekten an, welche sich in die Kategorien „Punkt“, „Linie“, „Fläche“ und „Körper“ einteilen lassen (vgl. Adobe 2004, 547f).

Wie in Kapitel 4 gezeigt werden wird, ist der hochauflösende Export aus vvvv nur über eine sequenzielle Vergrößerung gitterförmig angeordneter Bildausschnitte zu erreichen. Die einzelnen Vektorobjekte verhalten sich bei Vergrößerung unterschiedlich, was zu Einschränkungen der in dieser Arbeit vorgestellten Methode zum auflösungsunabhängigen Export aus vvvv führt. Deshalb ist es unabdingbar, über die Dimensionalität zu sprechen, insbesondere in Bezug auf die Darstellung und der Konstruktion eines Elements. Hier muss angemerkt werden, dass sich alle hier gezeigten Objekte in einem 3-dimensionalen Koordinatensystem (Höhe, Breite, Tiefe) befinden. Intern weisst vvvv jedem Objekt 3 Koordinaten zu, wobei Bildschirme lediglich zwei Dimensionen (Höhe und Breite) darstellen können. Der Knoten, der das Bildfenster erzeugt (Renderer) rechnet die dreidimensionalen Daten so um, dass sie auf der zweidimensionalen Bildfläche dargestellt werden können.

3|1|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)*

Dies ist eine sehr mathematische Definition und im Allgemeinen leichter verständlich scheint Euklids Definition zu sein: *„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. Implizit ermöglicht aber jeder Knoten, der ein grafisches Objekt zeichnet, auch eine Darstellung als Punktewolke der Eckpunkte. Ein Knoten. der einen gefüllten Würfel zeichnet, kann also auf eine Darstellung umgeschaltet werden, die nur die 6 Eckpunkte des Würfels zeigt.

Aus der Eigenschaft eines Punktes immer das kleinste darstellbare Objekt eines Ausgabemediums zu sein ergibt sich eine wichtige Einschränkung: Ist ein Punkt auf einem zu exportierenden Bild einen Pixel groß, wird sich dessen Größe beim Vergrößern eines Bildausschnittes nicht verändern und auch bei sehr starker Vergrößerung stets einen Pixel bzw. Punkt groß sein.

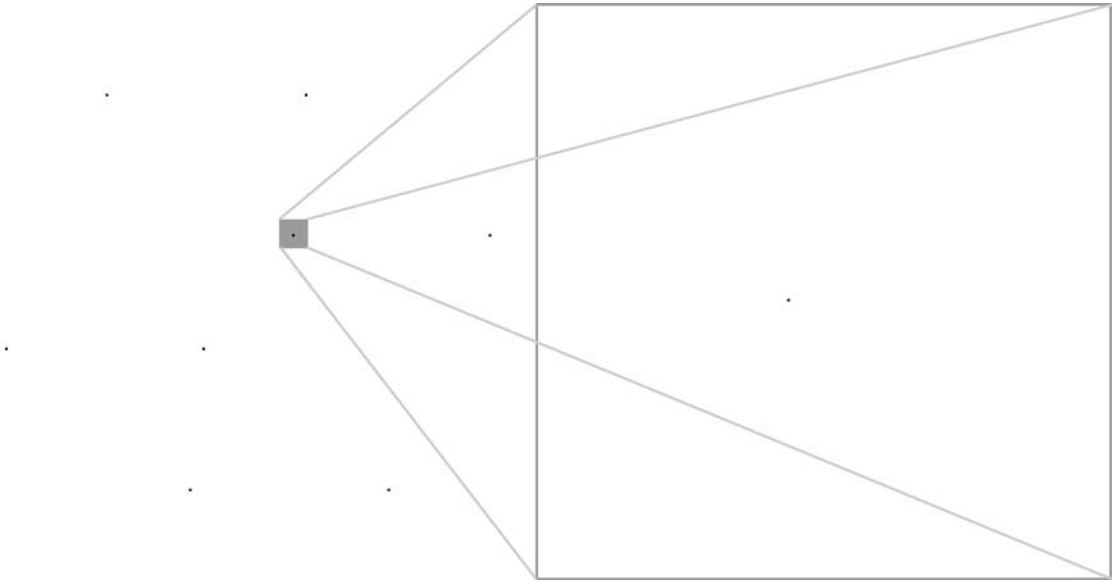


Abb | 6: Die sechs Eckpunkte eines Würfels (links) und ein vergrösserter Bildausschnitt (rechts). Die Grösse der Punkte ist identisch.

Als Beispiel soll Abbildung 6 dienen: Die 6 Eckpunkte eines Würfels. Wie aus den Bildern ersichtlich ist, verändert sich die Größe des Punktes auch bei starker Vergrößerung des Bildes nicht. Nur der Abstand zwischen den Punkten wird vergrößert. Dieses Problem der unproportionalen Vergrößerung kann nur vermieden werden, indem auf Punkte verzichtet wird und stattdessen andere Objekte (z.B. ein sehr klein skaliertes Quad (DX9)) verwendet werden.

3|1|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, und dabei keine Dicke besitzt.

vvvv kann jedes generierte Vektorobjekt auf einen Ansichtsmodus umschalten, in dem nur die Kanten zwischen Eckpunkten gezeichnet werden. So kann ein gefülltes Dreieck so gezeichnet werden, dass nur die 3 Kanten sichtabr sind. Allerdings ist die Dicke der Linien nicht beeinflussbar und hat deshalb immer nur eine Dicke von einem Pixel. Daraus ergeben sich Probleme bei der Vergrößerung von Bildausschnitten.

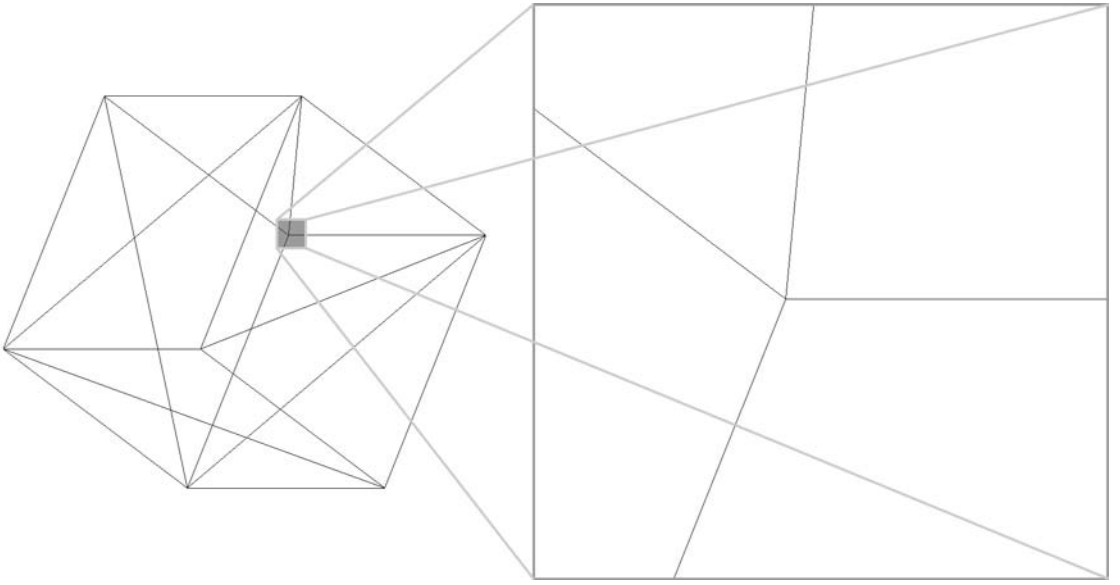


Abb | 7: Das Gitternetzmodell eines Würfels (links) und ein vergrösserter Bildausschnitt (rechts). Die Liniendicke ist identisch.

Wie aus Abbildung 7 ersichtlich beträgt die Linienstärke der Würfelkanten immer genau einen Pixel so dass das Verhältnis zwischen Linie und Weißflächen bei vergrößerten Bildausschnitten anders ist als beim unvergrößerten Ausgangsbild. Auch diese Einschränkung bei der proportionalen Vergrößerbarkeit von Linien lässt sich nur durch einen Verzicht auf diesen Darstellungsmodus und das Ersetzen durch andere Objekte (z.B. ein langgezogenes Quad (DX9)) umgehen.

3|1|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 Viereck, 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.

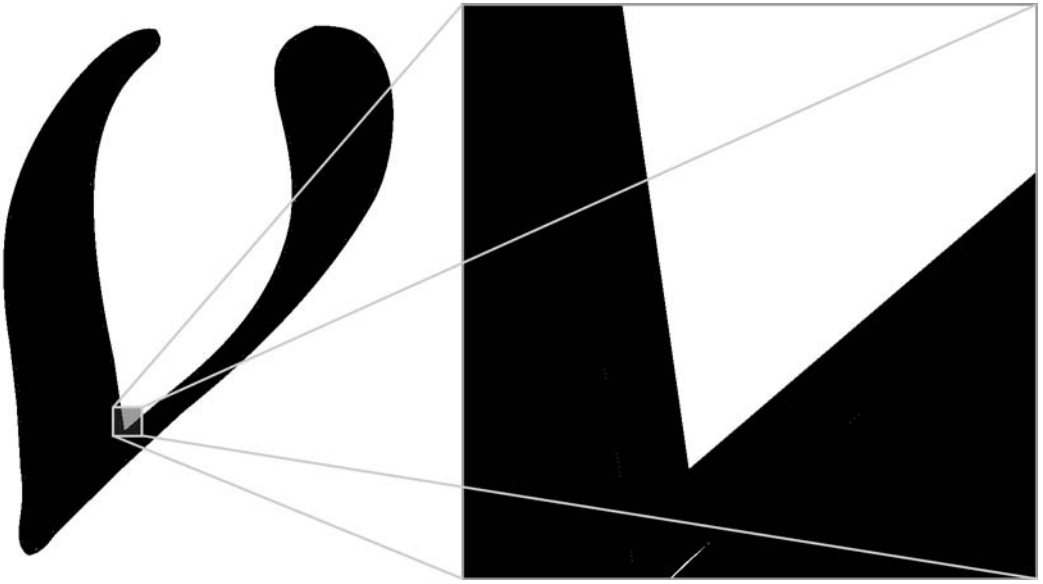


Abb | 8: Eine Fläche (links) und ein vergrösserter Bildausschnitt (rechts). Das Verhältnis von Fläche und Leeraum ist unverändert.

vvvv bietet diverse Knoten an, die sich zum Zeichnen von Flächen eignen, wobei eine Fläche immer eine gewisse Ausdehnung, aber keine Dicke besitzt, d.h. Flächen haben kein Volumen. Somit ist die Fläche ein zweidimensionales Objekt im 3-dimensionalen Raum.

Mit Flächen gibt es keinerlei Probleme beim proportionalen Vergrößern von Bildausschnitten, wie Abbildung 8 zeigt. Natürlich lassen sich bei Vergrößerungen mehr Details erkennen, als im unvergrößerten Bild. So ist z.B. dieses v nicht ganz sauber konstruiert und zwischen den Vektorlinien gibt es kleine Weißräume.

3|1|4| Körper

Körper setzten sich aus unterschiedlichen Flächen (meist Drei- oder Vierecken) zusammen, die so angeordnet werden, dass ein dreidimensionales Modell eines Objektes entsteht. In vvvv können Körper entweder generiert oder importiert werden. So kann mit einem beliebigen 3D Modelling Programm ein Modell erstellt werden, das dann in vvvv darstellbar und manipulierbar ist.

Über den Knoten VertexBuffer (EX9.Geometry Join) in Verbindung mit Mesh (EX9.Geometry Join) stellt vvvv Funktionen bereit, die die Erstellung von Körpern aus Punktewolken ermöglichen. So können komplexe 3D-Objekte direkt generiert und punktgenau manipuliert werden. Zudem existiert ein Knoten, der 3-dimensionale Textobjekte erzeugen kann. Dies funktioniert mit allen installierten Fonts und Symbol-Fonts. Jeder Körper hat eine Oberfläche, die ein gewisses Volumen umschließt und deshalb lassen sich alle gefüllten Körper problemlos vergrößern, wie Abbildung 9 zeigt.

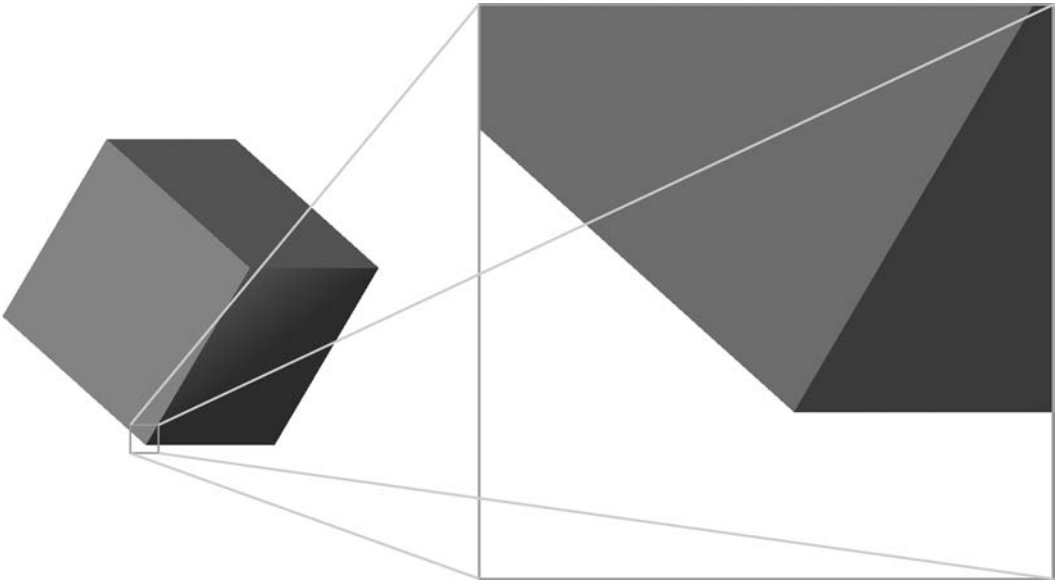


Abb | 9: Ein Würfel (links) und ein vergrösserter Bildausschnitt (rechts). Das Verhältnis von Fläche und Leeraum ist unverändert.

3|2| *Bitmap Daten*

vvvv kann Bitmapdaten nur als Oberflächen (Texturen) auf Vektorobjekten darstellen.

Das 3D Objekt dient dabei als Skelett über das ein Bild gelegt wird. Da Bitmap Dateien immer in einer gewissen Grundauflösung vorliegen, können diese nicht ohne Qualitätsverlust skaliert werden. Beim Verkleinern gehen Informationen verloren und beim Vergrößern gibt es keine Möglichkeit, mehr Informationen (also zusätzliche Details) auf der Bildfläche darzustellen als jene im Ursprungsbild enthaltenen. (vgl. Adobe 2004, 547)

Sollten also in Bildern, die mit der in dieser Arbeit gezeigten Methode hochauflösend exportiert werden sollen, Texturen verwendet werden, so gilt es einen wichtigen Umstand zu beachten: die Textur verliert immer dann an Qualität, wenn sie über ihre ursprüngliche Größe hinaus vergrößert wird.

Dazu ein Beispiel:

Eine Textur mit einer Größe von 1024 x 1024 Pixeln soll auf einem Quadrat dargestellt werden. Dieses Quadrat ist Teil eines Bildes, das in einer Größe von 5120 x 5120 Pixeln exportiert werden soll. Sobald das Quadrat über die Größe der darauf liegenden Textur vergrößert wird, kommt es zu einem Qualitätsverlust. Deswegen sollte dieses nicht mehr als 1/5tel des gesamten Bildes einnehmen.

Wird diesem Aspekt der eingeschränkten Vergrößerbarkeit von Bitmapdaten Rechnung getragen, gibt es keine Probleme bei der Verwendung von Texturen für den hochauflösenden Export aus vvvv.

{4}

Bildexport mit vvvv

41| Als Ganzes

Es gibt nur einen Knoten in vvvv der ein Bild erzeugen kann und das ist der „Renderer“.

Diesen gibt es in 5 Varianten:

1. TTY: Ein Terminalfenster, das nur Text darstellen kann und über das Fehlermeldungen und andere Informationen angezeigt werden können.
2. HTML: Ein HTML Browser mit dem sich Internetseiten anzeigen lassen.
3. GDI: Zeichnet 2d Strichgrafiken über das „Graphic Device Interface“ von Windows
4. Flash: Kann Flash Filme (.swf) darstellen
5. DX9: Der Renderer, der auf das DirectX Interface von Windows zugreift und hardwarebeschleunigt 3D Objekte rendern kann.

   Auf diesen beziehen sich alle folgenden Ausführungen.

Der Renderer (DX9) kann die in einem Patch definierten Objekte darstellen und, in Kombination mit zwei anderen Knoten auf einen Datenträger speichern. Der Ausgangspin DX9 Out des Renderers muss mit dem Source Pin eines DX9.Texture (EX9.Texture) Knotens verbunden werden. Anschließend muss dessen Texture Out Pin mit dem Texture Pin des Writer (EX9.Texture) Knotens verbunden werden. Hier muss das Dateiformat und ein Dateiname definiert werden. Nach klicken auf den Save Pin, wird das Bild gespeichert. Diesen Aufbau zeigt Abbildung 10.

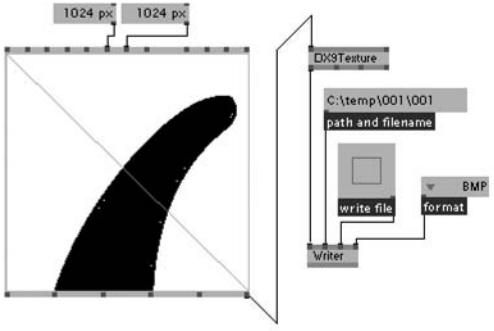


Abb | 10: Aufbau zum Speichern des Inhalts eines Renderers.

Um die Pixelgröße des zu speichernden Bildes festzulegen, dienen die Backbuffer Width und Backbuffer Height Pins am Renderer (DX9) Knoten. vvvv akzeptiert an diesen beiden Pins Werte von 1 bis 10000 Pixeln aber die maximale Ausgabegröße wird zusätzlich durch die verwendete Hardware, im Speziellen der Grafikkarte, eingeschränkt, wie ein Entwickler von vvvv in einem privaten E-Mail (siehe Anhang) erklärt:

> *„2048x2048 Pixel als Beschränkung für die maximale Größe einer Textur sind eine reine Hardwarebeschränkung und treffen auch nur für ATI- Karten zu. Die meisten nvidia Karten scheinen eine maximale Auflösung von 4096x4096 zu haben.*

Vielleicht gibt es noch andere Grafikkarten Hersteller, die andere Beschränkungen in ihre Produkte einbauen. Schwer herauszufinden da so etwas kaum bei den technischen Spezifikationen im Internet dabei steht. Es handelt sich dabei um den Wert den die Grafikkarten und deren Treiber für die „caps“ (= capabilities) „MaxTextureWidth“ und „MaxTextureHeight“ zurückliefern. In der neuen vvvv Version (Beta 9) wird dieser Wert der Grafikkarte am zweiten Pin des „Device (Auto)“ Knotens angezeigt.“ (Joreg)

Diese Beschränkungen entstehen durch die Tatsache, dass Hardware immer mit einer endlichen Anzahl von Transistoren umgehen muss. Somit gibt es ganz klare Begrenzungen in der Anzahl der Dinge, die softwaretechnisch unterscheiden werden können und somit eine Obergrenze für die Anzahl von Pixeln in einer Textur. (vgl. Thompson 2003, 137)

Sebastian Oschatz, dessen E-Mail sich ebenfalls im Anhang befindet, erklärt diese Limitierungen so: ¬ 40

„4096x4096 scheint das zu sein, was heutige Hersteller maximal zulassen. Immerhin 16MB Bilddaten, die in jedem Frame bewegt werden müssen, wenn man mit der Textur arbeitet. 8192x8192 sind dann schon 64MB -- kein Spielehersteller glaubt, dass sich mit den damit erzielbaren Frameraten ein Blumentopf gewinnen ließe. Also spart man die Leitungen, Zählwerke und Adressierungslogiken ein. Alles wertvolle Transistoren.“

An dieser Stelle ist anzumerken, dass sämtliche genannten Pixelwerte Potenzen von 2 sind.

2048 = 211, 4096 = 212

Zweierpotenzen sind in der Hardwaretechnik wichtig, weil sie die Zahlen beschreiben, die mit einer minimalen Anzahl von Adressleitungen (Bussen) eindeutig adressiert werden können (vgl. Thompson 2003, 138). Also eine technische Optimierung.

Mit 9 Leitungen können 512 Pixel adressiert werden (29 = 512). Sollen 513 Pixel adressiert werden, bräuchte es schon 10 Leitungen. Mit gleichem Aufwand könnten jedoch bis zu 1024 Pixel adressiert werden. Relativ gesehen, sind also 513 teurer als 1024.

Da jede Grafikkarte eine endliche Anzahl an Adressleitungen besitzt, und viele davon für andere Aufgaben (z.B. Geometrie- und Befehlsspeicher) verwendet werden (vgl. Fernando 2003, 97) stellt die Grafikkarte nur eine begrenzte Anzahl von Adressleitungen pro Textur zur Verfügung. Wie bereits erwähnt ist diese Obergrenze für die Bildgröße nicht bei allen Grafikkarten gleich und gewisse Karten können auch sehr große Texturen (4069 Pixeln entsprechen 34,5 cm bei 300dpi) rendern. Und in Zukunft wird es ohne Zweifel möglich sein, noch größere Texturen zu rendern. Trotzdem wird es immer eine Obergrenze geben und somit ist die Auflösungsunabhängigkeit keinesfalls gegeben. Es muss also ein Weg gefunden werden, um die Limitierungen der Hardware zu umgehen.

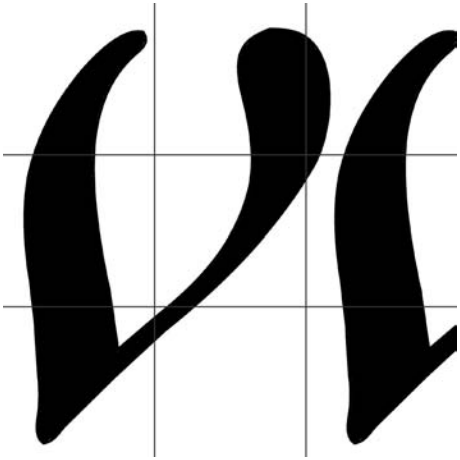
4|2| *In Teilen*

Wie in Kapitel 3 gezeigt, können die meisten von vvvv erzeugten Objekte beliebig vergrößert werden ohne an Qualität zu verlieren. Allerdings verhindern die in Kapitel 4|1 erörterten Hardwareeinschränkungen den Export von beliebig großen Bildern.

Diese Einschränkungen können nur überwunden werden, indem Bildteile vergrößert gerendert werden. Durch Verschieben des Bildausschnittes kann ein Bild in mehreren Einzelteilen gespeichert werden, die zusammengesetzt dem zu exportierenden Einzelbild entsprechen.

So lässt sich die maximale Größe der exportierbaren Bilder vervielfachen. Je höher die Vergrößerung der einzelnen Bildausschnitte, desto mehr Einzelteile hat das exportierte Bild. Somit kann jedes aus Vektorobjekten zusammengesetzte Motiv in beliebiger Größe exportiert werden. Der beschränkende Faktor ist nur noch die Größe der Festplatten, wobei ein 10 x 10 Meter großes Farbbild mit 300 dpi „nur“ 39GB groß wäre. Somit ist das Limit eher theoretisch.

Eine mögliche Einteilung zeigt Abbildung 11. Dabei wird das Bild in 3 x 3 quadratische Teile zerlegt. Ob und wie sich diese Bildaufteilung mit vvvv realisieren lässt, ist das Thema des nächsten Kapitels.



Abb|11: Mögliche Aufteilung eines Motives in gleich grosse Teile.

{5} Export von Bildausschnitten mit vvvv

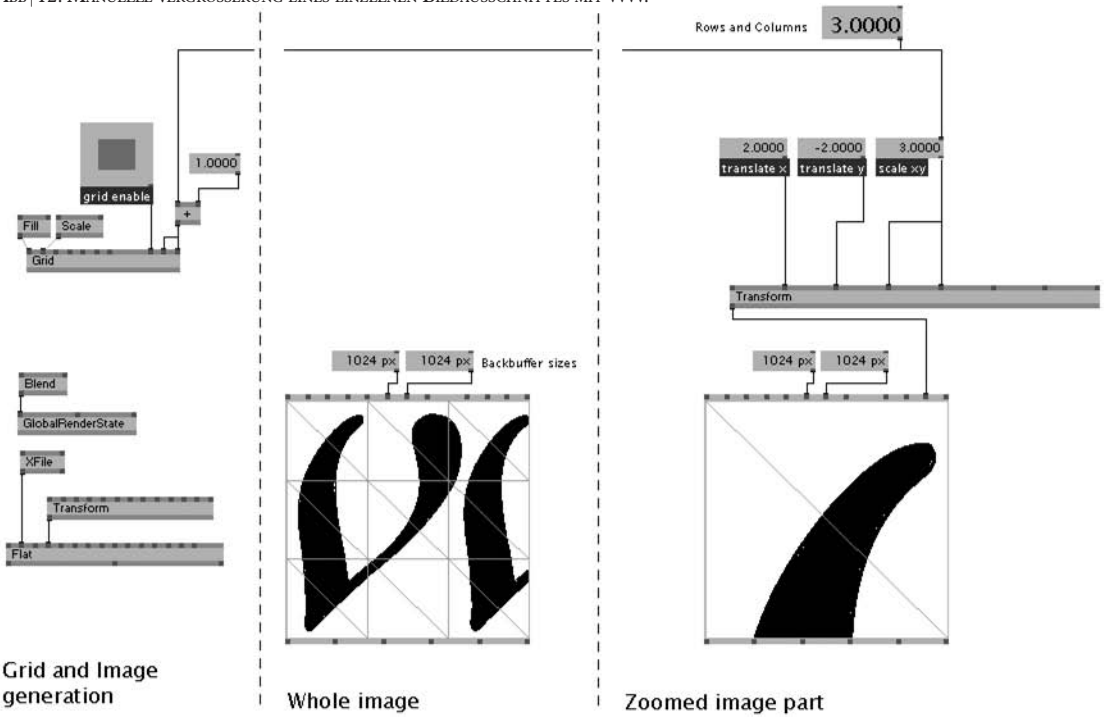
Die Vergrößerung von definierten Bildausschnitten ist mit vvvv mit wenigen Knoten zu lösen, wie der Patch in Abbildung 12 (nächste Seite) zeigt. Er besteht aus 3 Sektionen: Bildgenerierung, einem Renderer des Gesamtbildes und dem Teil, der einzelne Bildausschnitte vergrößert.

Im Bildgenerierungsteil wird ein 3D Modell importiert und transformiert sowie das Gitter gezeichnet, das die Bildaufteilung visualisiert. Dieses Gitter kann mit dem „grid enable“ Button ein- und ausgeblendet werden. Im Mittelteil befindet sich nur ein Renderer (DX9) der die im Bildgenerierungsteil erzeugten Objekte darstellt.

Der rechte Teil vergrößert einen bestimmten Bildausschnitt wofür ein einziger Knoten ausreicht. Mit dem Transform (Transform 2d) Knoten kann über die Pins TranslateX, TranslateY, ScaleX und ScaleY genau der Bildausschnitt des rechten Renderer (DX9) Fensters eingestellt werden.

Dazu muss Transform Out mit dessen View Transform verbunden werden. Die Vergrößerung sollte für X und Y identisch sein, da es sonst Verzerrungen gibt. Es ist nicht verwunderlich, dass der Wert für die Skalierung der Anzahl an Reihen und Spalten („Rows and Columns“) entspricht. Bei einem Drittel des Bildausschnittes ist die Vergrößerung gleich 3 wie der rechte Bildausschnitt im Beispiel (Abb. 12) zeigt.

Abb | 12: Manuelle Vergrösserung eines einzelnen Bildausschnittes mit vvvv.



So kann also der Bildausschnitt eingestellt werden. Jedoch erweist sich diese Methode als sehr mühsam, da die X und Y Position für jeden Bildausschnitt manuell eingestellt werden muss. Eine komfortablere Lösung sollte diesen Nachteil beseitigen.

vvvv besitzt einen Knoten, der eine schnelle Navigation in Gittern erlaubt: GridSplit (2d). Dieser wird auf den Transform (Transform 2d) Knoten aufgesetzt und kontrolliert somit den Bildausschnitt.

Da GridSplit (2d) die Transformationsinformationen für die Skalierung nicht in einem Format liefert, das für die Transformation des Renderer (DX9) Knotens geeignet ist, müssen einige Umrechnungen

stattfinden. Dazu dienen die Knoten UniformScale (Transform), Inverse (Transform) und / (Value). Dadurch werden die Unstimmigkeiten korrigiert, was der Aufbau in Abbildung 13 illustriert.

Indem der Wert des Index Pins von GridSplit (2d) verändert wird, kann der Bildausschnitt gewählt werden. Jeder Bildausschnitt hat eine Indexnummer, die sich aus der Position des Ausschnittes errechnen lässt. Die Ausschnitte sind von oben links (Indexnummer = 0) bis unten rechts (Indexnummer = n - 1) sequenziell durchnummeriert. n ist die Anzahl der Bildausschnitte, im vorliegenden Beispiel also 3 x 3 = 9. Soll z.B. der mittlere Bildausschnitt gewählt werden, muss der Index Pin von GridSplit (2d) auf 4 gestellt werden.

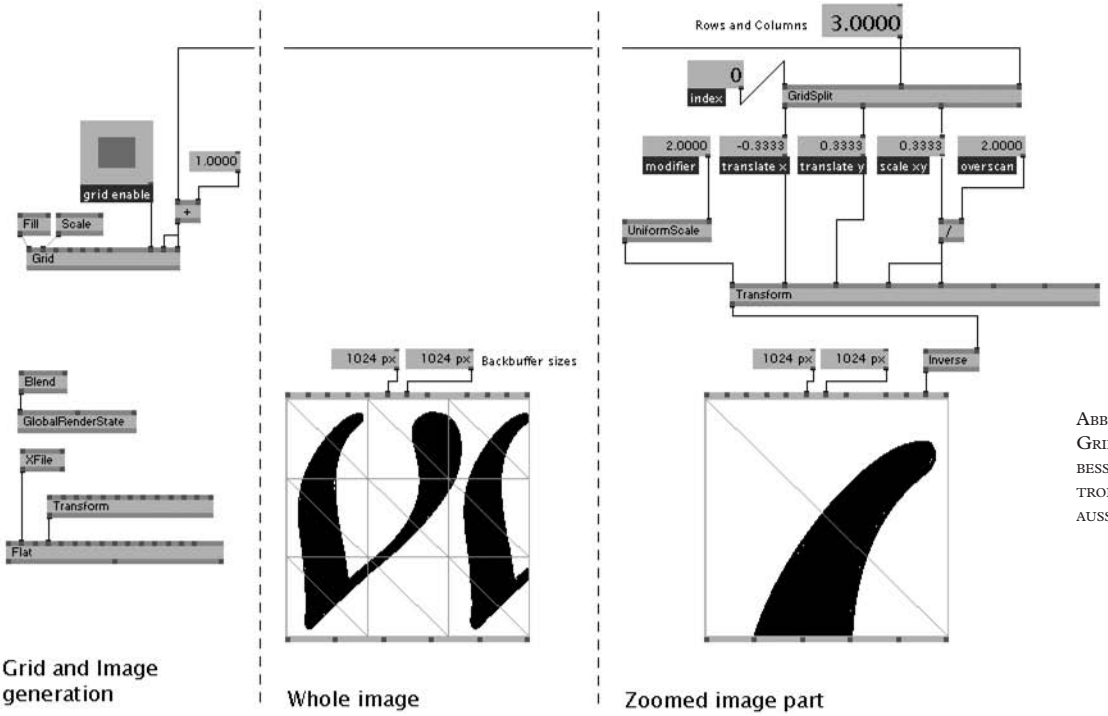


Abb | 13: GridSplit zur besseren Kontrolle des Bildausschnittes.

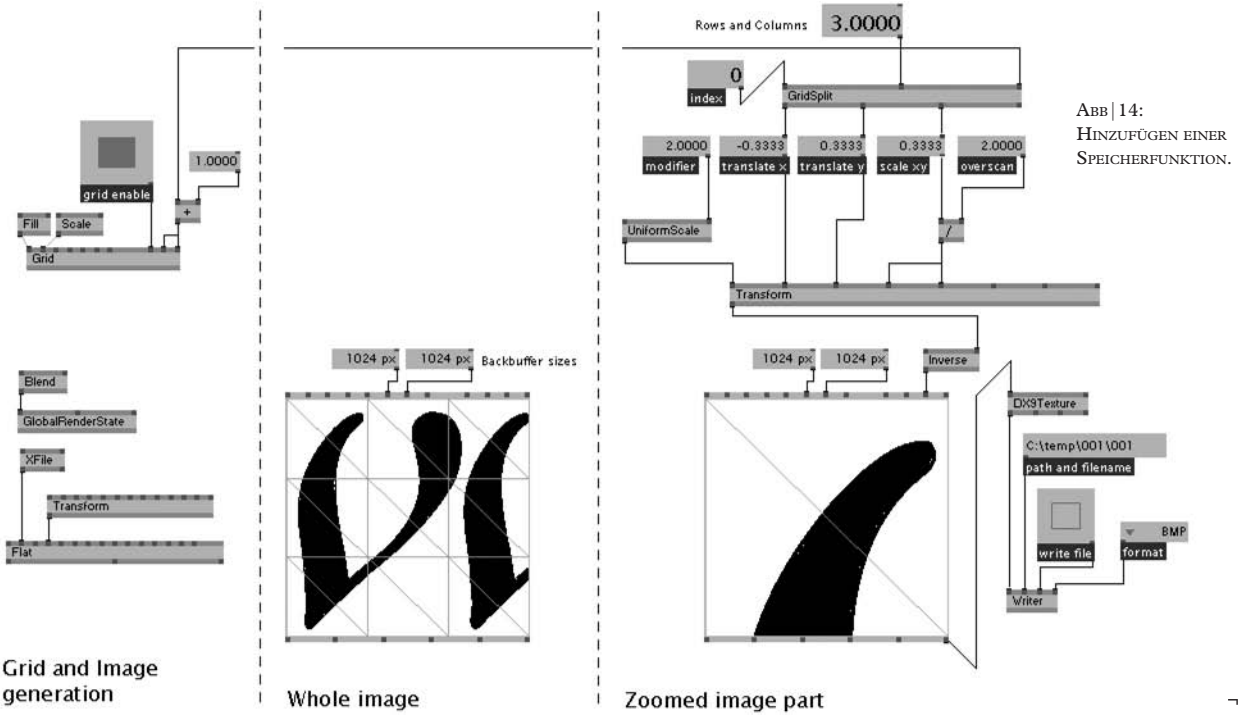


ABB | 14: HINZUFÜGEN EINER SPEICHERFUNKTION.

Da die Bildausschnitte nun schnell und einfach angesprungen werden können, muss der Patch noch um eine Speicherfunktion erweitert werden um den dargestellten Bildausschnitt auf einen Datenträger speichern zu können. Dazu werden nur zwei weitere Knoten (DX9Texture (EX9.Texture) und Writer (EX9.Texture)) benötigt, wie Abbildung 14 zeigt. Wie diese Knoten zu verbinden sind, ist aus dem Patch ersichtlich und wurde zudem bereits in Kapitel 4 | 1 erklärt. Wichtig ist hier nur, dass der Dateiname auf einen Pfad verweist, der bereits existiert, da der Writer (EX9.Texture) Knoten keine Verzeichnisse erstellen kann. Die Struktur des Dateinamens muss der Form „[Pfad]\[Dateiname]“ entsprechen, wobei beim Speichern die am Writer über den File Format Pin eingestellte Dateiendung angehängt wird. Ein Beispiel: wenn der File Name „C:\temp\vvvv.jpg“ ist und bei File Format „bmp“ eingestellt ist, wird die gespeicherte Datei als „vvvv.jpg.bmp“ im Pfad „C:\temp\“ gespeichert.

Der Patch ist jetzt schon so weit, dass Bildausschnitte gewählt und auf Knopfdruck gespeichert werden können. Allerdings muss der Dateiname für jedes Bild manuell eingegeben werden, was das Exportieren von vielen Einzelteilen sehr mühsam macht. Der nächste Schritt muss also sein, die Auswahl des Bildausschnittes mit dem Dateinamen zu verknüpfen. Ein etwas aufwendigerer Schritt als die vorhergehenden, da dem Patch sieben neue Knoten hinzugefügt werden müssen.

Im Wesentlichen wird dem Patch das Zählwerk Counter (Animation) hinzugefügt, welches durch die Bildausschnitte der Reihe nach durchschalten kann. Zudem kann der Ausgabewert verwendet werden um einen Dateinamen zu erzeugen.

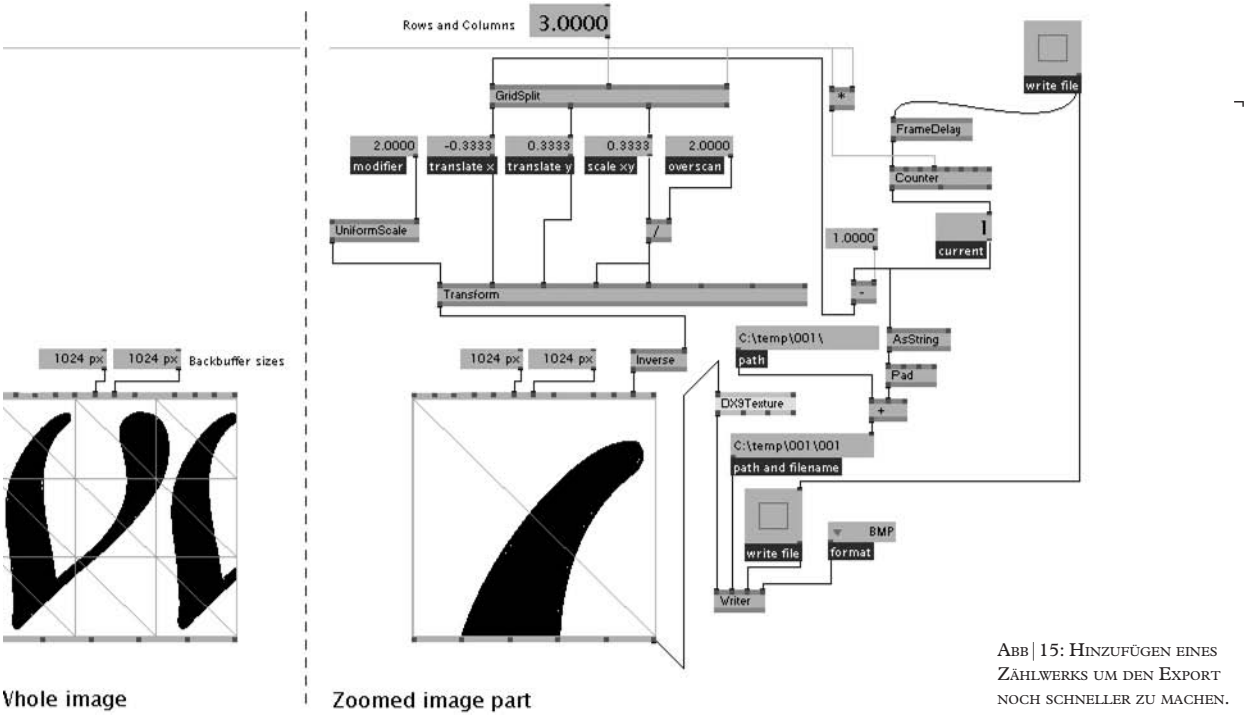


Abb | 15: Hinzufügen eines Zählwerks um den Export noch schneller zu machen.

Die Anzahl der Bildausschnitte errechnet sich indem die Anzahl von Reihen und Spalten multipliziert wird, im Beispiel 3 x 3 = 9. Das Zählwerk wird immer um 1 erhöht, wenn dessen Up Pin auf 1 gesetzt wird und springt auf 1 wenn das Maximum (9), also die Gesamtzahl der Bildausschnitte erreicht ist.

Da die Dateinamen mit 001 beginnen sollen, der erste Bildausschnitt jedoch die Indexnummer 0 hat, muss vom Output des Counter (Animation) 1 subtrahiert werden um den korrekten Ausschnitt zu erhalten.

Die Knoten AsString (Value), Pad (String) und + (String) verwandeln den Output des Zählwerks in einen Dateinamen indem die Indexnummer korrekt formatiert und mit dem statischen Speicherpfad kombiniert wird.

Der FrameDelay (Animation) Knoten hat eine Verzögerungsfunktion, die verhindert, dass der Wert vom Counter (Animation) um 1 erhöht wird bevor das Bild geschrieben wurde. Zuerst wird also gespeichert und dann weitergezählt. Somit können mehrere Teile eines Bildes schnell angesprungen und gleichzeitig gespeichert werden, was den Export von großen Bildern erheblich beschleunigt.

5|1| Konzeption eines Exportmodules

Im vorherigen Kapitel wurde ein Prototyp eines Patches angefertigt, der alle benötigten Funktionen enthält um ein Bild in beliebiger Größe exportieren zu können. Diese Funktionalität soll nun in ein Modul umgewandelt werden, das schnell und einfach mit einem beliebigen Bildgenerierungspatch kombiniert werden kann. Dieses Unterkapitel soll die Frage klären, wie dieses Modul aussehen könnte. Welche Eingaben wird es von Benutzern und Benutzerinnen erwarten und was soll es zurückliefern. Dabei wird größter Wert darauf gelegt, dass das Modul so einfach wie möglich bedient werden kann. Das Ziel ist es, das zu entwickelnde Modul zu einem integralen Bestandteil der offiziellen vvvv Distribution zu machen um es allen zugänglich zu machen.

5|1|1| Benutzerdefinierte Eingänge

Im Laufe der Entwicklung des Exportmodules wurde klar, dass die Eingangs Pins in zwei Klassen unterteilt werden können: Daten, die vom Bildgenerierungspatch, im Speziellen dem Renderer, geliefert werden müssen und Pins über die der Exportvorgang gesteuert wird.

Um das zu exportierende Bild korrekt an das Exportmodul übergeben zu können, müssen die Renderer (DX9) Knoten gewisse Einstellungen gemeinsam haben. Diese sind:

* render pass: Definiert, in welchen Renderer Fenstern ein Objekt dargestellt wird.
* bg color: Hintergrundfarbe des Renderers.
* depth buffer: Behandlung von Tiefeninformationen. Beeinflusst die Darstellung von Objekten bei Überlappungen.
* renderer transforms: Alle Transformationen die mit dem View Transform Pin des Renderer (DX9) im Bildgenerierungspatch verbunden sind.Sind diese nicht identisch, zeigen das exportierte Bild und der Bildgenerierungspatch nicht den selben Bildausschnitt.

Abb | 16: Eingänge des konzipierten Exportmodules.

Die anderen Pins bestimmen die anderen Funktionen des Exportmodules. Vor allem die Menge und Art und Größe der zu exportierenden Dateien. Die Pins im Überblick:

* enable: Aktiviert und deaktiviert den Renderer (DX9) im Exportmodul.
* size: Pixelanzahl pro Seite jedes exportierten Einzelbildes.
* rows: Reihen, in die ein Bild eingeteilt wird. Entspricht der Bilderanzahl in einer Reihe.
* cols: Spalte, in die ein Bild eingeteilt wird. Entspricht der Bilderanzahl in einer Spalte.
* GO!: Startet den Exportvorgang und schreibt alle Einzelteile auf einen Datenträger.
* Path: Der Ort auf dem Datenträger, an dem die Files gespeichert werden sollen.
* Filename Prefix: Statischer Teil des Dateinamens. Bei allen exportierten Bildern identisch.
* FileName Padding: Jedes Bild erhält beim Speichern eine eindeutige und fortlaufende Nummer. Dieser Input Pin definiert die Anzahl an Stellen, die diese Nummer haben muss. 3 bedeutet, dass eine dreistellige Zahl (z.B. 001 oder 134) an den Dateinamen bzw. Prefix angehängt wird.
* extension: Das Format, in dem die Bilder abgespeichert werden sollen.
  Zur Auswahl stehen BMP, JPG, PNG, DDS, DIB, HDR, PFM. Die Art wie die Bilder weiterbearbeitet werden sollen, bestimmt meistens die Auswahl des Dateityps.

Nachdem nun geklärt worden ist, welche Eingangspins ein Exportmodul haben sollte, muss geklärt werden, was es zurückliefern soll.

5|1|2| Ausgänge

Da das Modul Einzelbilder sequenziell auf einen Datenträger schreibt, kann der Exportprozess einige Zeit in Anspruch nehmen. Deswegen sollten Statusinformationen zur Überwachung des Programmablaufs zurückgeliefert werden. Folgende sinnvolle Ausgangspins konnten identifiziert werden:

Abb | 17: Ausgänge des konzipierten Exportmodules.

* texture out: Gibt den aktuellen Bildausschnitt als Textur aus. Diese kann in einem Rendererfenster dargestellt werden und so können bereits während dem Exportvorgang die Einzelbilder auf Fehler überprüft werden.
* working: Pin steht auf 1 wenn ein Exportvorgang läuft.
* currently saving: Zeigt den Dateinamen (ohne Erweiterung) des momentan bearbeiteten Bildausschnittes an.
* # images: Liefert die Gesamtanzahl der zu exportierenden Bilder zurück.
* % done: Der fertig gestellte Prozentsatz.

Ein Modul, das die oben definierten Ein- und Ausgangspins besitzt, könnte in jedem Knoten als Modul aufgerufen werden. Abbildung 18 zeigt, wie dieses Modul in einem leeren Patch aussehen würde. Allerdings sind die Ein- und Ausgänge in diesem Beispiel noch nicht funktionsfähig. Es ist lediglich eine Hülle für den in Kapitel 5 gezeigten Patch um Bildausschnitte zu vergrößern und abzuspeichern.

Aufgabe des nächten Kapitels wird es sein, alle bisherigen Erkenntnisse zur Implementierung eines voll funktionsfähigen und universell einsetzbaren Exportmodules zu nutzen.

Abb | 18: Das noch funktionslose Exportmodul in einem leeren Patch.

5|2| *Implementierung des Modules*

Die im bereits besprochenen Exportpatch (Abb. 15) verwendeten Knoten sind in 3 Kategorien eingeteilt:

5|2|1| Bildgenerierung

Die Erzeugung von zu rendernden Objekten. Dieser Abschnitt hat keine Verbindungen mit dem Exportteil und benötigt auch keine. Wenn ein Bild im Renderingteil angezeigt wird, trifft dies auch auf die exportierten Bilder zu. Somit wird dieser Teil nicht im Exportmodul enthalten sein. Deswegen müssen sich später alle Knoten zur Bildgenerierung in dem zu exportierenden Patch befinden. In diesem kann dann der Knoten für den Exporter angelegt werden um ein hochauflösendes Bild zu exportieren.

5|2|2| *Rendering*

Ein Renderer (DX9) und alle daran angeschlossenen Knoten wie Transformationen, Hintergrundfarbe, usw. werden in dieser Kategorie zusammengefasst. Hier werden die Einstellungen des Bildgenerierungsteils in ein Bild umgesetzt. Das zu exportierende Bild muss, bis auf die Größe, mit dem hier dargestellten identisch sein. Gewisse an den Renderer angeschlossenen Knoten (siehe dazu Kapitel 5|2|1) müssen an das Exportmodul angeschlossen werden.

5|2|3| *Exportteil*

Das Exportmodul wird im Wesentlichen aus den hier verwendeten Knoten bestehen, da der Großteil der geforderten Funktionalität bereits implementiert ist. Darum werden alle Knoten aus dieser Kategorie in einen leeren Patch kopiert und mit den in Kapitel 5|1|1 und 5|1|2 definierten Ein- und Ausgängen ausgestattet. Diese können dann mit den entsprechenden Knoten verbunden werden so dass die volle Funktionalität des Exportteiles wiederhergestellt wird. Wird dieser Patch in das „modules“ Verzeichnis von vvvv gespeichert, kann der Exportknoten sofort in jedem beliebigen Bildgenerierungspatch als einfacher Knoten angelegt werden und von dort Einzelteile auf die Festplatte speichern. Allerdings kann das Modul noch nicht selbstständig ein ganzes Bild exportieren und deswegen soll diese Einschränkung im nächsten Kapitel aufgehoben werden.

5|2|4 Writer (EX9.Texture Grid)

Der offizielle Name, den das Exportmodul trägt ist Writer (EX9.Texture Grid). Dieser entspricht den Namenskonventionen für vvvv Module und wurde zusammen mit den Entwicklern festgelegt. Unter diesem Namen findet sich das Modul auch in der offiziellen Distribution, die es auf der Entwicklerhomepage (vvvv.meso.net) zum kostenlosen Download gibt.

Im Gegensatz zum Patch in Kapitel 5 (Abb. 15) und im Prototyp des Exportmodules aus Kapitel 5|2|3 muss das fertige Modul alle Einzelteile eines Bildes automatisch exportieren können. Dazu muss dem Patch eine Schleife hinzugefügt werden, die den sequenziellen Export aller Bildausschnitte kontrolliert.

In Pseudocode kann der Ablauf eines Exportvorganges so dargestellt werden:

1. Die Gesamtanzahl der Teile berechnen
2. Größe des Bildausschnittes berechnen
3. Bildausschnitt auswählen
4. Filenamen generieren
5. Datei abspeichern
6. Gesamtfortschritt berechnen und Statusinformationen ausgeben
7. Zurück zu 3 bis das Bild in allen Einzelteilen exportiert wurde.

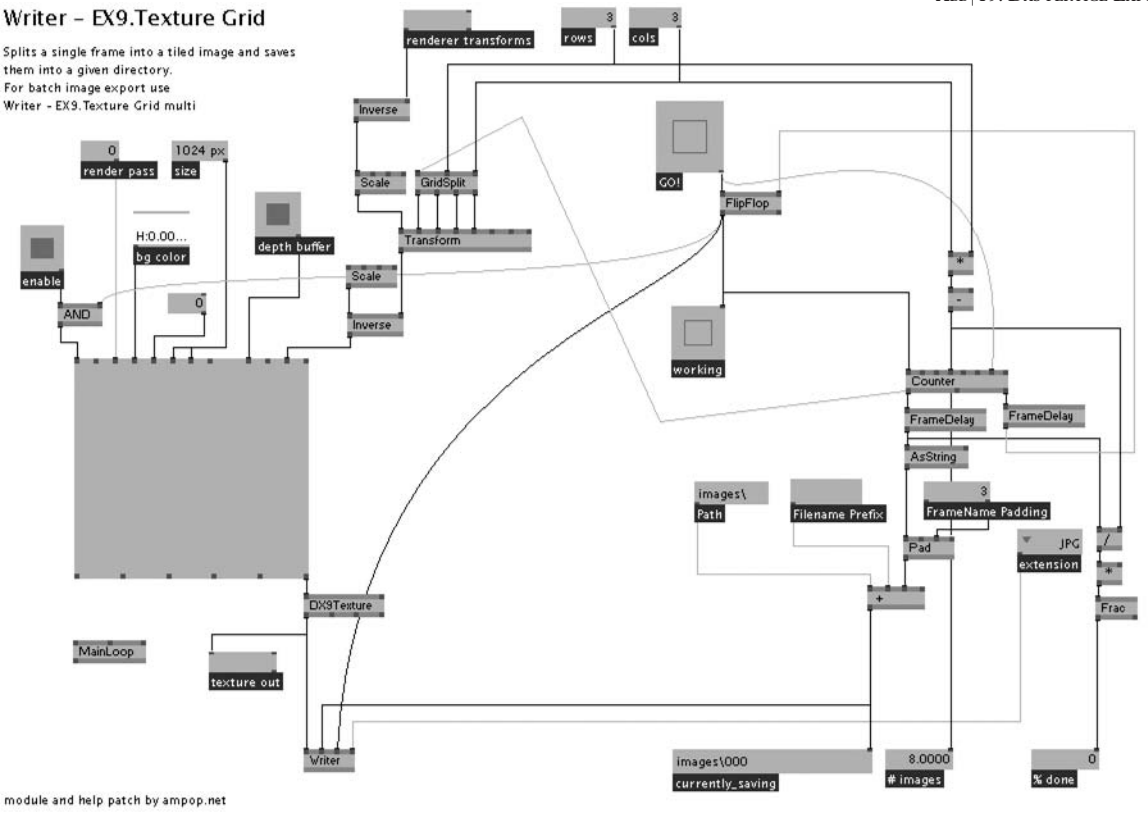


Abb | 19: Das fertige Exportmodul.

Abgesehen von 3 Knoten für die Fortschrittsberechnung in Prozent wurden dem Modul nur zwei Knoten hinzugefügt: FrameDelay (Animation) und FlipFlop (Animation). Ersterer verzögert nur die Weitergabe eines Eingangswertes für exakt einen Programmschritt (Frame). Der FlipFlop ist ein logischer Baustein, der seinen Ausgangswert auf 1 stellt, wenn irgend ein Wert außer 0 über den Set Pin in den Knoten geschickt wird. Der Ausgangswert bleibt so lang auf 1, bis irgend ein Wert außer 0 an den Reset Pin geschickt wird. In diesem Fall springt der FlipFlop wieder zurück und liefert am Ausgang 0. Diese Schritte können beliebig oft wiederholt werden, deswegen der Name.

Wird der Exportvorgang gestartet, schaltet der FlipFlop (Animation) auf 1 und die Speicherung der Einzelteile beginnt. Nach dem Speichern zählt der Counter (Animation) um 1 weiter und der nächste Bildausschnitt wird gespeichert. Der Counter hat jedoch ein definiertes Maximum (die Anzahl der zu exportierenden Bilder) bei dessen Überschreitung er auf den Startwert zurück springt. Dabei wird am Overflow Pin des Counters 1 ausgegeben. Passiert dies, wird noch abgewartet bis das letzte Einzelbild abgespeichert wurde und dann wird an den Reset Pin des FlipFlop (Animation) 1 geschickt. Dieser springt zurück auf 0 und so ist der Exportvorgang beendet.

Dies ist alles, was das Exportmodul können muss um schnell und einfach ein Bild in beliebig vielen Einzelteilen zu exportieren.

Im Anhang befindet sich ein Tutorial, das die Einbindung des Writer (EX9.Texture Grid) dokumentiert und im nächsten Kapitel wird erklärt, wie sich die damit exportierten Bilder wieder zu einem einzelnen, ¬ 57
größeren Bild zusammenfügen lassen.

{6} Das Zusammensetzen der Teile

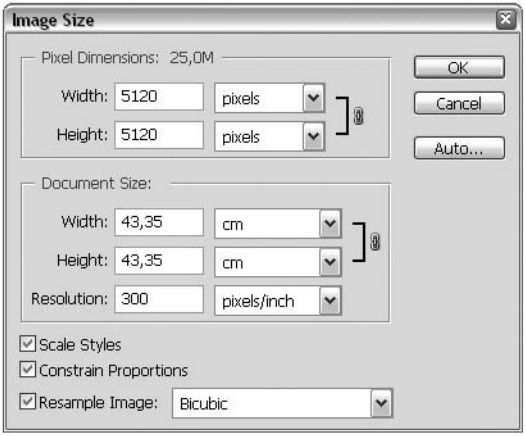
Da der gezeigte Writer (EX9.Texture Grid) eine mehr oder weniger große Anzahl von Einzelbildern generiert, müssen diese zuerst zu einem einzelnen Bild zusammengesetzt werden, bevor das exportierte Bild weiter verarbeitet oder gedruckt werden kann.

Im Folgenden werden 3 Methoden vorgestellt, die dazu eingesetzt werden können: ¬ 58

6|1| Manuell

Um die Einzelbilder manuell zu einem einzelnen Bild zusammenzufügen muss zuerst in einem beliebigen Bildbearbeitungsprogramm eine neue Datei angelegt werden, die mindestens genauso groß ist wie die Gesamtgröße der zusammenzusetzenden Einzelbilder. Wurden z.B. 5x5 Bilder mit jeweils 1024x1024 Pixeln exportiert, muss dieses Bild 5120x5120 Pixel groß sein. Bei einer Druckauflösung von 300dpi würde dies einer Ausgabegröße von 43,35 cm entsprechen, wie Abbildung 20 zeigt.

Abb | 20: Dialog zur Einstellung der Bildgrösse in Photoshop.



In dieses leere Bild müssen dann die Einzelbilder der Reihe nach eingefügt werden. Dazu empfiehlt es sich, vorher einen Raster aus Hilfslinien zu definieren (siehe Abbildung 21) um die Einzelbilder leichter anordnen zu können. Sind schließlich alle Teile eingefügt, muss die Datei nur noch gespeichert werden.

Diese manuelle Herangehensweise eignet sich nur, wenn nur sehr wenige Bilder zusammenzufügen sind und auch nur, wenn diese Bilder aus wenigen Teilen bestehen, da der Zeitaufwand beträchtlich ist.

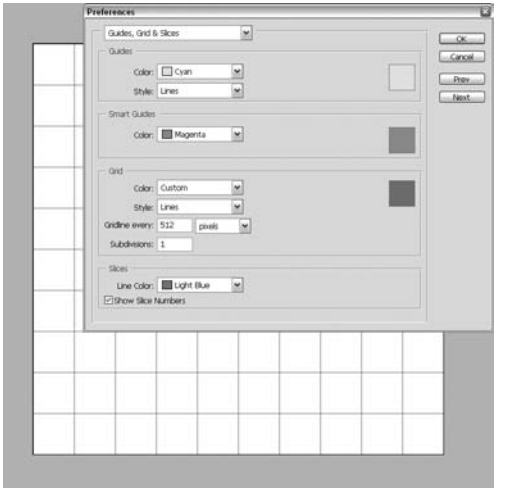


Abb | 21: Bildrastereinstellungen in Photoshop.

6\2\ Automatische Stapelverarbeitung

Anstatt für jedes exportierte Bild alle Einzelteile manuell zusammenzufügen, besteht auch die Möglichkeit mit einem Bildbearbeitungsprogramm, das die Automatisierung von Abläufen unterstützt, eine Stapelverarbeitung (Batch) zu erstellen. So muss ein Handlungsablauf nur einmal vorexerziert werden und kann dann beliebig oft auf andere zusammenzusetzende Bilder angewendet werden. Hier gibt es jedoch beträchtliche Einschränkungen, da die Ausgangssituation bei allen zusammenzusetzenden Bilder exakt gleich sein muss.

Werden z. B. für ein Projekt 5x5 Bilder exportiert und für ein anderes 7x7, muss für die neue Aufgabe eine neue Stapelverarbeitung geschrieben werden. Außerdem ist vor allem bei sehr großen Bildern schon das einmalige manuelle Zusammensetzen eine zeitaufwendige Arbeit.

Die Dateinamen müssen auch immer gleich sein und (zumindest bei Adobe Photoshop) die Einzelteile müssen immer im selben Ordner liegen. Deshalb müssen nach Beendigung der Stapelverarbeitung neue Daten manuell in den Quellordner kopiert und die Stapelverarbeitung neu gestartet werden. So kann immer nur ein Bild ohne Benutzerinteraktion zusammengesetzt werden.

6|3| *Ein leistungsfähigeres Tool*

Um die oben aufgezählten Nachteile einer in einem Bildbearbeitungsprogramm realisierten Stapelverarbeitung zu umgehen, muss ein Programm gefunden werden, das schnell und komfortabel eine beliebige Anzahl von Bildern (in beliebig vielen Ordnern) mit variabler Anzahl von Einzelteilen zusammensetzen kann. Es muss möglich sein, eine Ordnerstruktur einzulesen und alle dort in Einzelteilen gespeicherten Bilder vollautomatisch zu kombinieren, wobei das Programm selbstständig die Ausgabegröße jedes Bildes bestimmen müsste. Bei sehr vielen oder sehr großen Bildern kann es zu längeren Wartezeiten kommen und deshalb muss es möglich sein, das Programm im Hintergrund ablaufen zu lassen, so dass die benutzende Person ungehindert weiterarbeiten kann, während Bilder vom Programm bearbeitet und abgespeichert werden.

Um einen schnellen und unterbrechungsfreien Programmablauf zu ermöglichen sollten sich die Benutzerinteraktionen auf folgende 4 Aktionen beschränken:

1. Pfad zu den Bildern setzen
2. Ausgangsdatentyp einstellen
3. Ablaufpriorität des Programmes einstellen
4. Das Zusammensetzen starten

Da die Reihenfolge der Bilder eindeutig sein muss, wird es gewisse Konventionen für Ordner- und Dateinamen geben müssen. Alle Ordner und Dateien müssen so benannt sein, dass das Programm eindeutig identifizieren kann, welches Ausgangsbild an welche Stelle im zusammengesetzten Bild gehört. Dabei muss es unerheblich sein, welche Pixelgröße die Ausgangsdateien haben und wie viele Dateien im jeweiligen Ordner zu einem großen Bild zusammengesetzt werden sollen.

Ein Beispiel:

In Ordner 001 befinden sich 81 Dateien mit jeweils 1024 x 1024 Pixeln Größe.

In Ordner 002 befinden sich 16 Dateien mit jeweils 512 x 2048 Pixeln Größe.

In Ordner 003 befinden sich 16 Dateien mit jeweils 2048 x 512 Pixeln Größe.

Nach dem vollautomatischen Zusammensetzen sollen die Bilder folgende Größen haben:

Bild 001: 9216 x 9216 Pixel (77 x 77 cm bei 300 dpi)

Bild 002: 2048 x 8192 Pixel (17 x 68 cm bei 300 dpi)

Bild 003: 8192 x 2048 Pixel (68 x 17 cm bei 300 dpi)

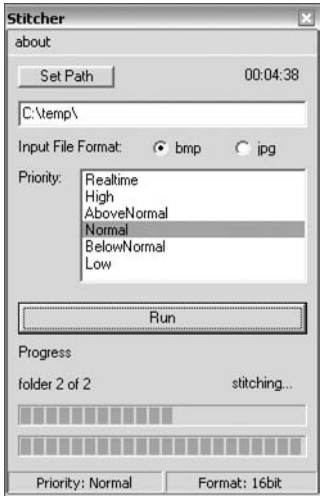
Es ist wichtig dass das Gitter, das die Aufteilung des Bildes in Einzelteile bestimmt, immer gleich viele Spalten und Reihen (2 x 2, 3 x 3, 4 x 4, usw.) hat. Ansonsten könnte das Programm nicht selbstständig entscheiden, wie die Einzelteile wieder zu einem korrekten Ausgangsbild zusammengesetzt werden müssen. Die Anzahl von Spalten und Reihen müsste deswegen für jedes Bild explizit angegeben werden.

Darum wird die im Beispiel gezeigte Methode empfohlen um das Seitenverhältnis des zusammenzusetzenden Bildes festzulegen: die Größe der Ausgangsbilder bestimmt das Seitenverhältnis des zusammengesetzten Bildes. Dies ist beim Export mit dem Writer (EX9.Texture Grid) über die Backbuffer Width und Backbuffer Height am Renderer (DX9) einzustellen. Eine weitere Methode um nicht-quadratische Bilder zu exportieren wird im Anhang A|3 kurz erwähnt.

Somit sind die Anforderungen an ein Programm definiert das gut mit dem vorgestellten Exportmodul Writer (EX9.Texture Grid) zusammenarbeiten kann und im folgenden Unterkapitel wir ein geeignetes Programm vorgestellt.

6|3|1| Stitcher

Da trotz umfangreicher Recherchen keine Software gefunden werden konnte, welche die oben beschriebenen Funktion bereitstellt, musste dieses Programm neu entwickelt werden. Dank der Hilfe von Alexander Dejaco war es dem Autor dieser Arbeit möglich, dies zu tun. Das Ergebnis ist ein leicht zu bedienendes Programm, das eine beliebige Anzahl von, in Einzelteile zerlegte, Bilder vollautomatisch zusammensetzen und auf einem Datenträger abspeichern kann. Abbildung 22 zeigt die Benutzeroberfläche dieses Programmes.



Abb|22: Die Oberfläche des „Stitcher“.

Das Programm wurde mit Borland Delphi in der Programmiersprache Pascal geschrieben und es ist etwas über 500 Zeilen lang. In Anhang A|1 und auf dem beiliegenden Datenträger befindet sich der vollständige Quellcode. Es werden keine nicht-standard Module benötigt um das Programm kompilieren zu können. Das Programm ist prozedural geschrieben und beinhaltet 16 Prozeduren und Funktionen, die durch Benutzeraktionen und von anderen Funktionen bzw. Prozeduren aufgerufen werden.

Sobald das Programm geöffnet wird, wird sofort die Prozedur FormActivate ausgeführt, welche die Funktion (loadPathFromReg) aufruft. Es werden lediglich Grundeinstellungen geladen und einige Variablen gesetzt.

Sobald das Programm offen ist, erwartet es eine Aktion. Dazu gibt es folgende Prozeduren:

* about1Click: zeigt ein Informationsdialogfeld an
* btnSetPathClick: Pfad zu den zusammenzusetzenden Bildern
* RadioBmpClick: aktivieren, wenn bmp Bilder zusammengesetzt werden sollen
* RadioJpgClick: aktivieren, wenn jpg Bilder zusammengesetzt werden sollen
* ListPriorityClick: stellt die Prozesspriorität ein
* btnRunClick: startet das Zusammensetzen

Beim Setzen des Pfades muss irgendein Bild in der Ordnerstruktur angewählt werden. Der Stitcher bestimmt dann selbstständig die Anzahl der Ordner und der jeweils darin enthaltenen Bilder. Nach dem Klick auf den „Run“Button (btnRunClick) wird überprüft, ob die Ordner und Dateien den vorgegebenen Namenskonventionen entsprechen, ansonsten wird eine Fehlermeldung ausgegeben.

Es ist wichtig, dass alle Ordner sequenziell nummeriert sind, 3-stellige Namen haben und dass der erste Ordner „001“ heißt. In weiterer Folge heißen die Ordner demnach „002“, „003“, „004“, usw. Für die Dateien gilt das selbe: Datei 1 muss „001.bmp“ bzw. „001.jpg“ heißen. Dabei müssen in allen Ordnern die Dateien das gleiche Dateiformat (bmp oder jpg) haben! Die Anzahl an Dateien kann für jeden Ordner unterschiedlich sein, so lang diese Anzahl quadratisch ist. Ein zusammengesetztes Bild kann demnach aus 4, 9, 16, 25, 36, 49, 64, 81, 100, … Bildern bestehen.

Diese Konventionen sind notwendig, um eine eindeutige Zuordenbarkeit aller Bilder zu gewährleisten. Werden diese eingehalten, bestimmt der Stitcher die Größe des zu erzeugenden Bildes und arbeitet sich dann Bild für Bild durch den ersten Ordner wobei jedes Einzelteil geladen und in einem neuen Bild an der korrekten Position eingefügt wird. Sind alle Bilder eines Ordners im neuen Bild eingefügt, wird dieses abgespeichert und der nächste Ordner in Angriff genommen. Für den nächsten Ordner wird wieder die Ausgabegröße berechnet, die Bilder eingefügt und dann das zusammengesetzte Bild abgespeichert bis alle Ordner abgearbeitet sind. Dabei bekommt jedes zusammengesetzte Bild den selben Namen wie der Quellordner, nur mit der Erweiterung „bmp“.

Der untenstehende Teil des Quellcodes ist die zentrale Schleife im Programm:

```
ocount := foldercount; // Berechnet die Zahl der Ordner
  ProgressBar1.Max := ocount; // Der Fortschrittsbalken
  ProgressBar1.Position := 0;
  ProgressBar.Position := 0;

  for o := 1 to ocount do // Ein Ordner nach dem anderen wird hier abgearbeitet
  begin
    bmset := false;
    LblFolder.Caption := 'folder ' + IntToStr(o) + ' of ' + IntToStr(ocount);
    ProgressBar.Position := 0;
    GetCount(o); // Ermittelt die Anzahl der Dateien im Ordner 'o'
    checkcount; // Überprüft die Namenskonventionen
    if (error = true) then // Ausgabe einer Fehlermeldung bei falschen Dateinamen
    begin
      LblDone.Caption := '';
      LblTime.Caption := '';
      exit;
    end;
    Calc(o);  // Die Positionierung des Teilbildes im grossen Bild                ¬ 65
    Save(o); // Abspeichern des zusammengesetzten Bildes 'o'
    ProgressBar1.Position := o;
    Application.ProcessMessages;
  end;
```

Die Schleife in Pseudocode:

1. Anzahl der Ordner bestimmen
2. Den nächsten Ordner einlesen und die Anzahl und Größe der enthaltenen Bilder bestimmen.

   All das macht die Funktion GetCount.
3. Die von GetCount ermittelten Daten mit checkcount überprüfen
4. Calc aufrufen, welches das Bild zusammensetzt.
5. Das Bild mit Save speichern.
6. Zurück zu Schritt 2 oder die Schleife beenden wenn alle Bilder zusammengesetzt worden sind.

Die eigentliche Bearbeitung des Bildes erfolgt in Schritt 4 durch die Prozedur Calc. Von dieser werden alle Bilder eines Ordners nacheinander geladen und in das Ausgabebild (SecondBM) eingefügt.

Die Variable i ist die Nummer des aktuellen Bildes und wurzel die Quadratwurzel aus der Gesamtanzahl von Bildern eines Ordners. Befinden sich z.B. 16 Bilder in einem Ordner, ist die Wurzel = 4.

```
SecondBM.Height := StrToInt(FloatToStr(h * wurzel)); // Höhe des Ausgabebildes
SecondBM.Width := StrToInt(FloatToStr(w * wurzel)); // Breite des Ausgabebildes

x := i mod wurzel;                                  // Berechnung der x Position
if x = 0 then
  x := (wurzel - 1) * w
else
  x := (x - 1) * w;

y := ((i - 1) div wurzel) * h;              // Berechnung der y Position

SecondBM.Canvas.Draw(x, y, JPG);                                          ¬ 66
```

h und w sind die Pixelgröße (z.B. h = 1024 und w = 512) der zusammenzufügenden Bilder. Die Größe des Ausgabebildes errechnet sich durch Multiplikation der Höhen und Breiten der Quellbilder mit der Anzahl von Bildern in jeder Spalte und Reihe.

Dann wird die x Position berechnet. i ist die Nummer des aktuellen Bildes und x ist der Rest aus der Division von i durch wurzel. Diese Form der Division wird auch Modulo-Funktion genannt. Wenn das Bild Nummer 1 bearbeitet wird, dann wird gerechnet: 1 / 3 = 0 mit 1 Rest. x ist dann also 1.

Bei i = 2 ist x = 2, bei i = 3 ist x = 0 und bei i = 4 ist x = 1.

Dieses Muster setzt sich fort, so dass die Folge 0, 1, 2, 0, 1, 2, … entsteht.

Wenn wurzel = 4 ist, ergibt sich die Folge 0, 1, 2, 3, 0, 1, …

Somit ist bekannt, in welche Spalte im Ausgabebild das aktuelle Einzelteil einzusetzen ist. Diese Position muss allerdings in Pixeln angegeben werden, um das Einzelteil an die korrekte Stelle im Ausgangsbild einfügen zu können.

Dazu muss überprüft werden, ob x = 0 ist. Trifft dies nicht zu, wird die Spaltennummer (abzüglich 1), in der sich das Einzelteil befindet, mit der Pixelgröße der Einzelteile multipliziert: x = (x - 1) * w

Das erste bearbeitete Bild (i = 1, x = 1) wird demnach ganz links im Ausgabebild eingefügt:

(1 – 1) * 512 = 0 * 512 = 0

Egal wie groß die Einzelteile sind, durch 0 dividiert wird das erste Bild immer an der x Position 0 stehen.

Für das zweite Bild (i = 2, x = 2) wird w mit 1 multipliziert. Das bedeutet, dass Bild 2 immer genau an jener Stelle eingefügt wird, an der Bild 1 endet.

Diese Methode der Positionsberechnung eignet sich für alle Spalten außer der letzten ganz rechts. Bei x gleich 0 würde der Wert -w zurückgeliefert und das Einzelteil außerhalb der Bildfläche eingefügt.

Deswegen wird dieser spezielle Fall von der if-Abfrage abgefangen und die Berechnung

x = (wurzel - 1) * w durchgeführt.

So kann die x Position für jedes beliebige Einzelteil berechnet werden und deshalb muss nur noch bestimmt werden auf welcher Höhe die Teile eingefügt werden müssen.

Dazu wird eine ganzzahlige Division von i – 1 durch wurzel durchgeführt und diese dann mit der Pixelgröße h multipliziert. Eine ganzzahlige Division liefert zurück, wie oft eine Zahl durch eine andere dividierbar ist. So können z.B. 0, 1 und 2 nicht durch 3 dividiert werden, wenn nur ganze Zahlen benutzt werden, da die Zahl vor den Kommastellen immer kleiner als 1 ist. Wird jede Zahl der Folge 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 ganzzahlig durch 3 dividiert, entsteht die Folge 0, 0, 0, 1, 1, 1, 2, 2, 2, 3. Hat ein zusammenzusetzendes Bild also 3 Reihen und Spalten, werden die ersten 3 Bilder ganz oben im Ausgabebild eingesetzt. Die nächsten 3 Bilder dann genau dort, wo die erste Reihe endet.

Nachdem die x und y Positionen eines Einzelteils bestimmt worden sind, kann dieses ins zu exportierende Bild eingefügt werden. Dieser Ablauf wird für alle Dateien im aktuellen Ordner wiederholt! Am Ende wird die Prozedur verlassen und das Programm springt zurück in die Hauptprozedur btnRunClick. ¬ 68
Diese ruft Save auf, das Bild wird gespeichert und dann kehrt die Hauptschleife zu ihrem Ursprung zurück. Der nächste Ordner wird bearbeitet bis alle Bilddateien in der ausgewählten Ordnerstruktur zusammengesetzt worden sind.

Somit kann eine beliebige Anzahl von Bildern vollautomatisch zusammengesetzt werden.
Die gezeigte Lösung „Stitcher“ ist für die Verwendung mit dem Writer (EX9.Texture Grid) konzipiert und nur dadurch ist ein effizienter hochauflösender Export möglich. Wäre das Zusammenfügen der Einzelteile zu kompliziert oder zeitaufwendig, brächte das Exportmodul keinen großen Nutzen.

{7} Conclusio

Die hier gezeigte Methode ermöglicht es vvvv, ein neues Anwendungsgebiet abzudecken: Print Design. Ohne die vorgestellte Erweiterung in Form eines Exportmodules ist dies nicht möglich. Das im Zuge dieser Arbeit entstandene Modul Writer (EX9.Exture Grid) wurde ausführlich dokumentiert und es wurde gezeigt, dass hochauflösender Export aus vvvv möglich ist. Durch die Entscheidung der Entwickler, das Exportmodul in die offizielle vvvv Distribution zu integrieren, konnte eine wichtige Prämisse dieser Arbeit erfüllt werden: etwas zu schaffen, das Anderen hilft.

Alle kreativen Köpfe die sich mit Design beschäftigen, sollten Interesse daran haben, ihre kreativen Freiheiten zu erweitern. vvvv besitzt viele Funktionen die über die Möglichkeiten von klassischen Designtools hinausgehen. Der Beweis für diese Behauptung soll vom beiliegenden Bildband geliefert werden, der vierundsechzig unbearbeitete vvvv- Grafiken enthält. Allerdings repräsentieren diese Bilder den vollen Funktionsumfang von vvvv nur sehr unzureichend.

Auf der Webseite der Entwickler (vvvv.meso.net) findet sich eine grössere Auswahl an Beispielbildern, die einen weiteren Überblick über die Möglichkeiten von vvvv geben. Hier nur ein winziger Ausschnitt aus den „Screenshots of the days“:

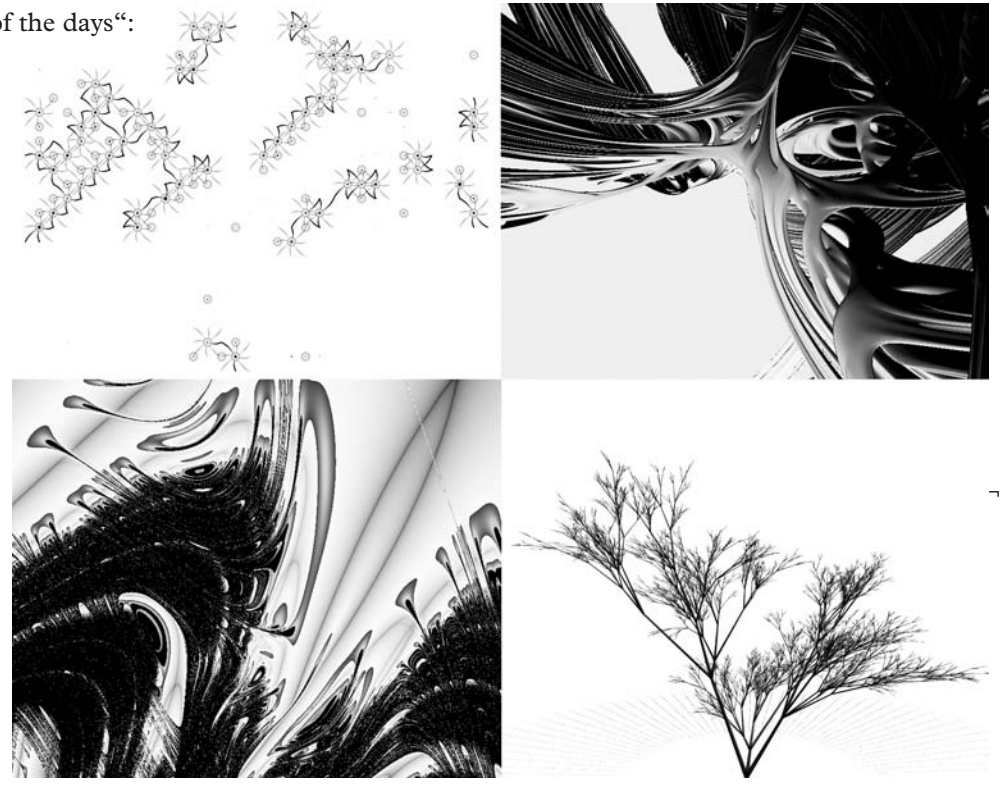


Abb | 23: Vier Beispiele aus den „Screenshots of the Days“ der vvvv Homepage. Davon basieren drei auf fraktalen Algorithmen.

Neben den ästhetischen Möglichkeiten gibt es weitere Vorteile beim Einsatz von vvvv. So können etwa externe Datenquellen verwendet werden um Designs zu beeinflussen. Texte können direkt aus Datenbanken importiert werden, oder Wetterdaten aus dem Internet das Aussehen von Objekten steuern. Die Grafiken im beiliegenden Buch basieren zum Teil auf solchen externen Daten. Auch der Fließtext im Bildband wurde von vvvv aus dem Internet abgerufen und in die Textfelder eingesetzt. Es gibt also genügend Gründe, die den Einsatz von vvvv für die Erzeugung von Print Design interessant machen.

Da sich diese Arbeit jedoch nicht mit der Bilderzeugung, sondern ausschliesslich mit dem Bildexport befasst, ist der Bildband eine Demonstration der Funktionalität des Writer (EX9.Exture Grid) Knotens in Kombination mit dem „Stitcher“: Sehr viele Bilder wurden mit hoher Auflösung aus verschiedenen Bildgenerierungspatches exportiert. Dabei gab es keinerlei Probleme und die Arbeit ging sehr schnell voran. Einige Messwerte die während der Arbeit am Bildband gemacht wurden:

* Integration des Exportmodules in einen Bildgenerierungspatch: 2 - 5 Minuten
* Speicherung eines Bildes mit dem Exportmodul (8192 x 8192 px): 50 Sekunden
* Zusammenfügen von 10 Bildern mit dem Stitcher (alle 8192 x 8192 px): 65 Sekunden

Der Writer (EX9.Exture Grid) Knoten ist seit „vvvv33beta8.1“ fixer Bestandteil der offiziellen Distribution. Die in dieser Arbeit gezeigten vvvv Programme sind mit dieser Beta Version entwickelt worden und funktionieren nur mit dieser Programmversion.

„vvvv33beta9“ wurde während der Ausarbeitung dieser Arbeit veröffentlicht und konnte deswegen nicht berücksichtigt werden. Das Exportmodul wurde jedoch bereits auf die neuere Version angepasst und ist im Installationspaket (vvvv.meso.net) enthalten. Es waren jedoch nur kleinere Modifikationen notwendig um die Patches auf die neue Version zu portieren und die Unterschiede sind dabei so gering, dass alle hier gemachten Aussagen weiterhin Gültigkeit haben.

Der „Stitcher“ ist noch unveröffentlicht. Auf stitcher.ampop.net wird jedoch in Kürze eine Webseite eingerichtet, die sich mit hochauflösendem Export aus vvvv befassen wird. Dort kann dann der „Stitcher“ kostenlos heruntergeladen werden.

Um den Exportprozess weiter zu optimieren, sollen auch Erweiterungen auf der Webseite vorgestellt werden. Im Anhang A|3 finden sich einige Entwürfe hierzu. Neben bereits gelösten Problemen, die nur dokumentiert werden müssten, gibt es noch ungelöste Probleme, wie etwa den sequenziellen Export von Animationen in hoher Auflösung. So könnte eine Plattform geschaffen werden um die Etablierung von vvvv im Designalltag zu beschleunigen.

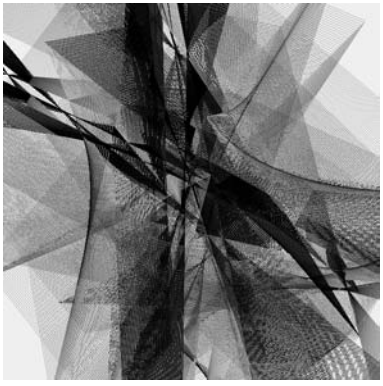
Die in dieser Arbeit gezeigte Methode zum Export von Bildausschnitten ist mit vielen anderen grafischen Programmierumgebungen kompatibel. Z.B. kann die gezeigte Methode (in Kombination mit dem „Stitcher“) auch auf puredata | gem und max | msp angewendet werden um aus diesen Programmen auflösungsunabhängig exportieren zu können.

Zusammenfassend kann also gesagt werden, dass eine Methode entwickelt wurde, um mit vvvv beliebig große Einzelbilder zu exportieren, was das mögliche Einsatzgebiet der Software erweitert.
Die gezeigte Lösung wurde in die offizielle Distribution integriert und gehört dadurch zum Funktionsumfang der Standardversion von vvvv. Somit ist die 100-prozentige Verfügbarkeit des Exportmodules gewährleistet, was allen vvvv- Benutzerinnen und Benutzern die Möglichkeit gibt, beliebig große Bilder zu erzeugen. Darüber hinaus lässt sich die gezeigte Technik auch auf ähnliche grafische Programmierumgebungen übertragen.

{a} Anhänge

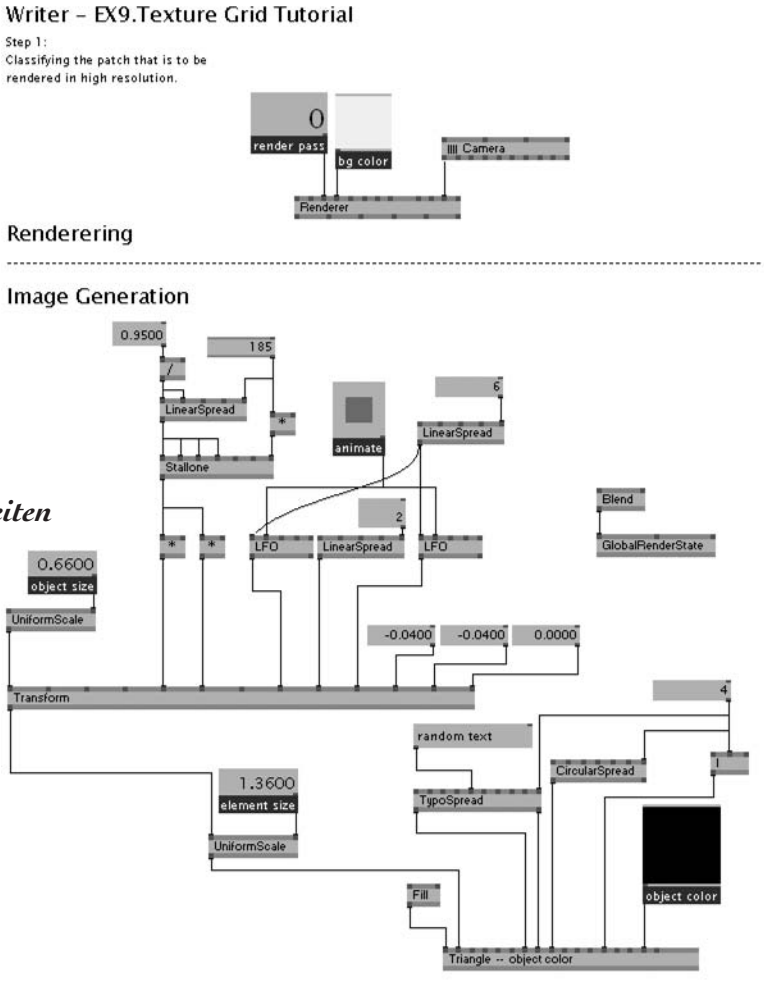
A|1| Writer Tutorial

A|1|1 Zu exportierendes Bild



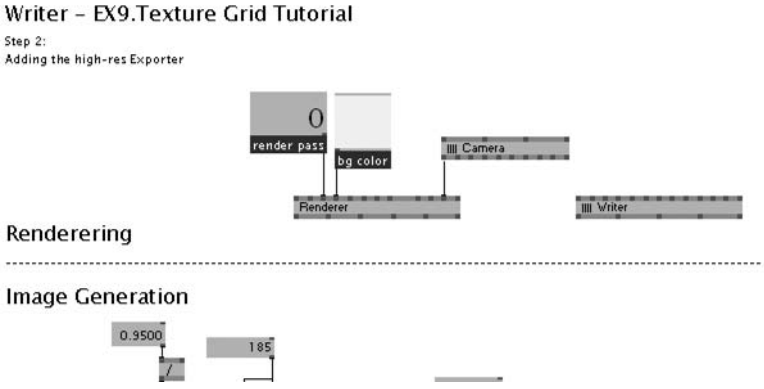
Abb|A|1: Das zu exportierende Bild

A|1|2 Bildgenerierungspatch vorbereiten



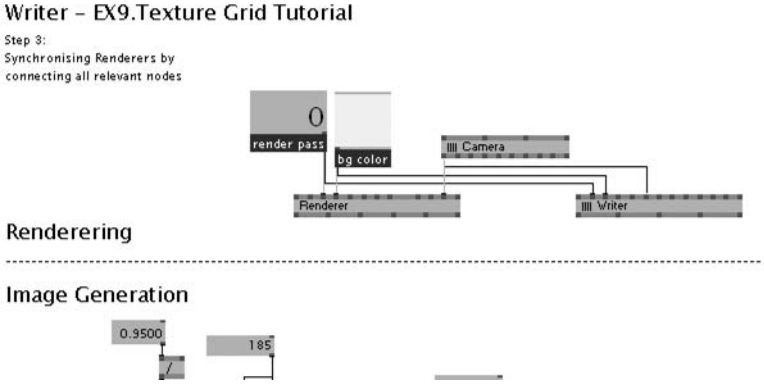
Abb|A|2: Der umzubauende Patch mit Renderer und Bildgenerierungsknoten.

A\1\3 Writer hinzufügen



Abb|A|3: Das Exportmodul wird im Patch als einfacher Knoten angelegt.

A\1\4 Connections ziehen



Abb|A|4: Die Einstellungen vom Renderer müssen an das Exportmodul übergeben werden.
Wird der Go! Pin aktiviert, werden die Einzelteile des Bildes auf einen Datenträger geschrieben.

A|1|5 Steuerknoten und Grundeinstellungen

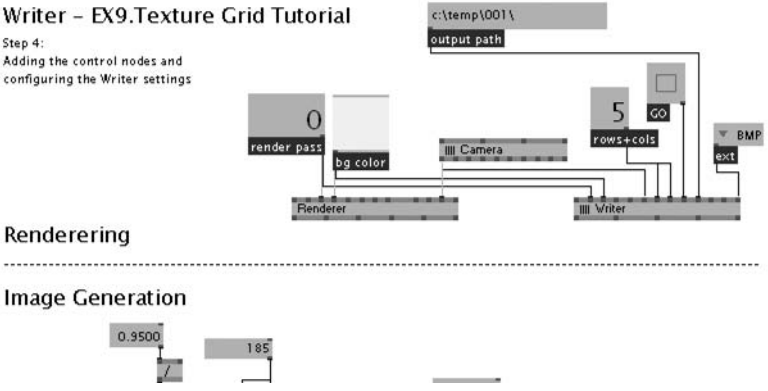


Abb | A | 5: Zusätzliche Kontrollknoten sollten angelegt werden um die restlichen Einstellungen des Exportmodules leichter überblicken zu können.

A|1|6 GO klicken und schon werden die Bilder in den Zielpfad gespeichert

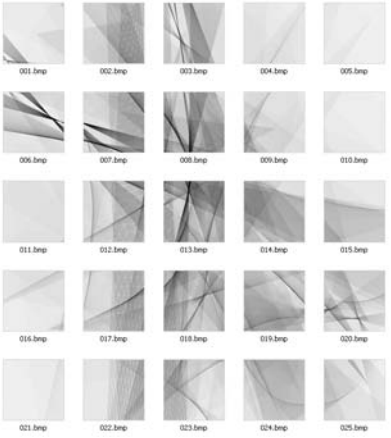


Abb | A | 6: Die exportierten Bilder im Windows Explorer.

A|2| Stitcher Quellcode

```
//  Program:      stitcher ***
//  Author:       Thomas Hitthaler, thomas@ampop.net
//  Supervisor:   Alexander Dejaco, lex@will-hier-weg.de
//  Creation:     2005-06-12  (yyyy-dd-mm)
//  Last Change:  2005-10-14  (yyyy-dd-mm)
//  Purpose:      the program reads jpg or bmp images and stiches them together
//                e.g. if we have 9 files, they are stiched together like this 123
//                                                                            456
//                                                                            789
//  Usage:        Stitcher.exe
//  Input:        The input data has to match a certain structure,
//                the path to a file in egrep would be
//                ^[0-9][0-9][1-9]\\[0-9][0-9]((.bmp)|(.jpg))$
//                the folder must be numbered from 001...999
//                and the files in each folder from 001-999
```



```
unit Main;


interface


uses
Windows, Messages, SysUtils, Graphics, Forms, registry, Classes, Controls, Dialogs, Menus,
StdCtrls, ComCtrls, Jpeg;
```

```
type
TfrmMain = class(TForm)

//  *************** UI declarations ********************

    MainMenu1: TMainMenu;
    about1: TMenuItem;
    LblDone: TLabel;
    RadioJpg: TRadioButton;
    RadioBmp: TRadioButton;
    ProgressBar: TProgressBar;
    ProgressBar1: TProgressBar;
    LblFolder: TLabel;
    OpenDialog: TOpenDialog;
    StatusBar1: TStatusBar;
    Label1: TLabel;
    Label3: TLabel;
    LblTime: TLabel;
    Label2: TLabel;
    Button1: TButton;
    ListPriority: TListBox;
    EditPath: TEdit;
    Button2: TButton;
    procedure FormDestroy(Sender: TObject);
    procedure about1Click(Sender: TObject);
    procedure RadioBmpClick(Sender: TObject);
    procedure RadioJpgClick(Sender: TObject);
    procedure btnRunClick(Sender: TObject);
    procedure FormActivate(Sender: TObject);
    procedure ListPriorityClick(Sender: TObject);
    procedure btnSetPathClick(Sender: TObject);
```

```
// ***************** declarations (variables, functions, procedures) **********************

  private
    count, ocount, h, w, wurzel: integer;
    exepath, format, ordner, filen: string;
    bmset: boolean;
    SecondBM, BM: TBitmap;
    JPG: TjpegImage;
    path: string;
    procedure calc(o: integer);
    function CalcTime: string;
    procedure save(o: integer);
    procedure checkcount;
    function foldercount: integer;
    procedure getCount(o: integer);
    procedure InitBMForCalcOnce;
    function loadPathFromReg: string;
    { Private declarations }
  public
    { Public declarations }
  end;

var
  frmMain:        TfrmMain;
  error, running: boolean;
  starttime:      TTime;
  pixFormat:      TPixelFormat;

implementation

{$R *.dfm}
```

```
// ***************** when the program opens *********************
// * Name:           * FormActivate
// * Event:          * on activation of the form
// * Parameters:     * Sender
// * Return Values: * none
// * Description:   * some inital values are set when the program is run

procedure TfrmMain.FormActivate(Sender: TObject);
begin
  running := false;
  exepath := ExtractFilePath(Application.ExeName);
  path := loadPathFromReg;
  pixformat := pf16bit;
  if path = '' then
  begin
    path := exepath;
  end;
  EditPath.Text := path;
end;

// ***************** loading settings for FormActivate ********************
// * Name:           * loadPathFromReg
// * Parameters:     * none
// * Return Values: * string
// * Description:   * function loads the 'FilePath' of the last session from the registry

function TfrmMain.loadPathFromReg: string;
var
  Reg: TRegistry;
begin
  Reg := TRegistry.Create;
  try
    Reg.RootKey := HKEY_CURRENT_USER;
    if Reg.OpenKey('\software\stitcher', false) then
    begin
      result := Reg.ReadString('FilePath');
    end;
  finally
    Reg.Free;
    inherited
  end;
end;
```

```
// ***************** When the „RUN“ button is klicked *********************
// * Name:           * btnRunClick
// * Event:          * on clicking the run button
// * Parameters:     * Sender
// * Description:    * this is the main procedure
//                   * the file format is read in, the start-values are set
//                   * the calculation functions are called in steps:
//                   * ‘getcount’, ‘checkcount’, ‘calc’ and finally ‘save’

procedure TfrmMain.btnRunClick(Sender: TObject);
var
  o: integer;
begin
  SecondBM := TBitmap.Create;
  JPG := TJpegImage.Create;
  BM := TBitmap.Create;
  BM.PixelFormat := pf16bit; // the selected pixel-format is set
  JPG.PixelFormat := jf24Bit;
  starttime := now; // starttime
  error := false;
  if Radiobmp.Checked = true then // check format                    ¬ 81
    format := ‘bmp’
  else
    format := ‘jpg’;
  ocount := foldercount; // call procedure foldercount and store to ocount
  ProgressBar1.Max := ocount; // set progressbar values
  ProgressBar1.Position := 0;
  ProgressBar.Position := 0;
  for o := 1 to ocount do // iterative going through all folders
  begin
    bmset := false; // it is set true when the output bitmap is initialized
    LblFolder.Caption := ‘folder ‘ + IntToStr(o) + ‘ of ‘ + IntToStr(ocount);
    ProgressBar.Position := 0;
    GetCount(o); // stores the number of files in the current folder ‘o’ into ‘count’
    checkcount; // checks the count for wrong entries
    if (error = true) then // if an error was found reset values and exit procedure
    begin
      LblDone.Caption := ‘’;
      LblTime.Caption := ‘’;
      exit;
    end;
```

```
    Calc(o);  // calculation is called for the current folder 'o'
    Save(o); // the save procedure is called for the current folder 'o'
    ProgressBar1.Position := o;
    Application.ProcessMessages;
  end;
  LblDone.Caption := 'done';
  LblTime.Caption := CalcTime; // the return value of calcTime is showed as a label
  LblFolder.Caption := '';
  Application.ProcessMessages;
end;



// ********** Sub procedures called by btnRunClick and childs **********

// * Name:          *  foldercount
// * Parameters:    *  none
// * Return Values: *  cnt: integer
// * Description:   *  returns the number of folders  in the selected file structure

function TfrmMain.foldercount: integer;
var
  i, j, cnt: integer;
  ordner: string;
  stop: boolean;
begin
  stop := false;
  cnt := 0;
  i := 0;
  repeat
    i := i + 1;
    ordner := IntToStr(i);
    for j := 1 to 3 - length(ordner) do
      ordner := '0' + ordner;
    if DirectoryExists(path + ordner) then
      cnt := cnt + 1
    else
      stop := true;
  until stop;
  result := cnt;
end;
```

```
// * Name:          *  getCount
// * Parameters:    *  o: integer (current folder-number)
// * Description:   *  counts the files in the specific folder and stores in 'count'

procedure TfrmMain.getCount(o: integer);
var
  i, j: integer;
begin
  LblDone.Caption := 'reading files...';
  Application.ProcessMessages;
  i := 0;
  repeat
    i := i + 1;
    ordner := IntToStr(o);
    for j := 1 to 3 - length(ordner) do
      ordner := '0' + ordner;
    filen := IntToStr(i);
    for j := 1 to 3 - length(filen) do
      filen := '0' + filen;
  until not FileExists(path + ordner + '\' + filen + '.' + format);
  count := i - 1;
end;

// * Name:          * checkcount
// * Description:   * the procedure checks the 'count' value on errors occured
// *                * (e.g. wrong file format or incorrect structure)

procedure TfrmMain.checkcount;
var
  i: integer;
begin
  if (count = 0) then
  begin
    MessageDlg('error' + #13 + #10 + 'possible wrong file format', mtError, [mbOK], 0);
    error := true;
  end;
  if (TryStrToInt(FloatToStr(sqrt(count)), i) = false) then
  begin
    MessageDlg('The File-count of the structure is incorrect!', mtError, [mbOK], 0);
    frmMain.Close;
  end;
end;
```

```
// * Name:          * calc
// * Parameters:    * o: integer (current folder-number)
// * Return Values: * none
// * Description:   * the procedure which stitches the images in the specific folder together
//                  * Two standard Tbitmaps are used, one file in the directory is read into
//                  * 'BM' and then drawed into 'SecondBM' at the correct position.
//                  * the result image is going to be x * y size where x = y
//                  * The raws are first filled

procedure TfrmMain.calc(o: integer);
var
  i, j, y, x: integer;                                                    ¬ 84
begin
  LblDone.Caption := 'stitching...';
  Application.ProcessMessages;
  wurzel := StrToInt(FloatToStr(sqrt(count))); // the sqrt of the filecount is stored
  ProgressBar.Max := count;
  for i := 1 to count do // the path to the current file wanted is calculated
  begin
    ProgressBar.Position := i;
    ordner := IntToStr(o);
    for j := 1 to 3 - length(ordner) do
      ordner := '0' + ordner;
    filen := IntToStr(i);
    for j := 1 to 3 - length(filen) do
      filen := '0' + filen;
    Showmessage(path + ordner + '\' + filen + '.' + format);
    try
      if (format = 'jpg') then                                  // which format is to read in
        jpg.LoadFromFile(path + ordner + '\' + filen + '.' + format) // current file is loaded
      else
        BM.LoadFromFile(path + ordner + '\' + filen + '.' + format); // current file is loaded
```

```
    except
      on e: exception do
      begin
        MessageDlg('error (' + e.Message + ')' + #13 + #10 + '' + #13 + #10 +
        'possible wrong file format', mtError, [mbOK], 0);
        ProgressBar.Position := 0;
        ProgressBar1.Position := 0;
        LblDone.Caption := 'error occured';
        LblFolder.Caption := '';
        Application.ProcessMessages;
        exit;
      end;
    end;

    if bmset = false then
      InitBMForCalcOnce; // the output image is initialized before the first image
                         // can be loaded into it
    y := ((i - 1) div wurzel) * h; // position of current image
    x := i mod wurzel;             // in the output image is calculated
    if x = 0 then
      x := (wurzel - 1) * w                                                    ¬ 85
    else
      x := (x - 1) * w;
    if (format = 'jpg') then
      SecondBM.Canvas.Draw(x, y, JPG) // the current image is drawn into the
    else                              // output image at calculated position
      SecondBM.Canvas.Draw(x, y, BM);
  end;
end;
```

```
// * Name:          * InitBMForCalcOnce
// * Parameters:    * none
// * Return Values: * none
// * Description:   * 'SecondBM' which is used as output for the stitched image initialized

procedure TfrmMain.InitBMForCalcOnce;
begin
  try
    if (format = 'jpg') then
    begin
      h := JPG.Height;
      w := JPG.Width;
    end else
    begin
      h := BM.Height;
      w := BM.Width;
    end;
  except
    on e: exception do
      MessageDlg('error (' + e.Message + ')' + #13 + #10 + '' + #13 + #10 +
      'possible wrong file format', mtError, [mbOK], 0);
  end;
  SecondBM.Height := StrToInt(FloatToStr(h * wurzel)); // the size of the input files is
  SecondBM.Width := StrToInt(FloatToStr(w * wurzel)); // multipied by the number of rows
  bmset := true;                                      // and columns of the output image
end;
```

```
// * Name:          * save
// * Parameters:    * o: integer (current folder-number)
// * Return Values: * none
// * Description:   * the procedure stores the stitched image on the harddrive

procedure TfrmMain.save(o: integer);
var
  name: string;
  i: integer;
begin
  LblDone.Caption := 'saving...';
  Application.ProcessMessages;
  name := IntToStr(o);
  for i := 1 to 3 - length(name) do
    name := '0' + name;
  SecondBM.SaveToFile(path + name + '.bmp');
  BM.Width := 0;
  SecondBM.Width := 0;
end;
```



```
// * Name:          * CalcTime
// * Parameters:    * none
// * Return Values: * time: string
// * Description:   * calculates the time used for the stitching and returns it as string

function TfrmMain.CalcTime: string;
var h, m, s, ms: word;
  time: string;
begin
  decodetime(now - starttime, h, m, s, ms);
  if m < 10 then time := '0' + inttostr(m) else time := inttostr(m);
  time := time + ':';
  if s < 10 then time := time + '0' + inttostr(s) else time := time + inttostr(s);
  time := time + ':';
  ms := ms div 10; {Zweistellige Millisekunden}
  if ms < 10 then time := time + '0' + inttostr(ms) else time := time + inttostr(ms);
  result := time;
end;
```

```
// ***************** When the program is closing *********************
// * Name:          * release
// * Event:         * when the Stitcher is closed
// * Parameters:    * Sender
// * Return Values: * none
// * Description:   * resets the width of the bitmaps after a calculation

procedure TfrmMain.FormDestroy(Sender: TObject);
begin
  SecondBM.Free;
  BM.Free;
  JPG.Free;
end;

// ***************** Button Clicks and Interface Programming  *********************
// the about-box
procedure TfrmMain.about1Click(Sender: TObject);
begin
  MessageDlg('"stitcher"' + #13 + #10 + '' + #13 + #10 +
  'http://stitcher.amopop.net', mtInformation, [mbOK], 0);
end;

// the format buttons-handler
procedure TfrmMain.RadioBmpClick(Sender: TObject);
begin
  if RadioJpg.Checked = true then
  begin
    RadioJpg.Checked := false;
    RadioBmp.Checked := false;
  end;
end;

// the format buttons-handler
procedure TfrmMain.RadioJpgClick(Sender: TObject);
begin
  if RadioBmp.Checked = true then
  begin
    RadioBmp.Checked := false;
    RadioJpg.Checked := false;
  end;
end;
```

```
// changes the priority of the process if changed in the list box
procedure TfrmMain.ListPriorityClick(Sender: TObject);
const
  BELOW_NORMAL_PRIORITY_CLASS = $4000;
  ABOVE_NORMAL_PRIORITY_CLASS = $8000;
begin
  case ListPriority.ItemIndex of
    0: begin
        SetPriorityClass(GetCurrentProcess, REALTIME_PRIORITY_CLASS);
        StatusBar1.Panels[0].Text := 'Priority: RealTime';
      end;
    1: begin
        SetPriorityClass(GetCurrentProcess, HIGH_PRIORITY_CLASS);
        StatusBar1.Panels[0].Text := 'Priority: High';
      end;
    2: begin
        SetPriorityClass(GetCurrentProcess, DWORD(ABOVE_NORMAL_PRIORITY_CLASS));
        StatusBar1.Panels[0].Text := 'Priority: AboveNormal';
      end;
    3: begin
        SetPriorityClass(GetCurrentProcess, NORMAL_PRIORITY_CLASS);
        StatusBar1.Panels[0].Text := 'Priority: Normal';
      end;
    4: begin
        SetPriorityClass(GetCurrentProcess, DWORD(BELOW_NORMAL_PRIORITY_CLASS));
        StatusBar1.Panels[0].Text := 'Priority: BelowNormal';
      end;
    5: begin
        SetPriorityClass(GetCurrentProcess, IDLE_PRIORITY_CLASS);
        StatusBar1.Panels[0].Text := 'Priority: Low';
      end;
  end;
end;
```

```
// * Name:          * btnSetPathClick
// * Event:         * on clicking the 'set path' button
// * Parameters:    * Sender
// * Description:   * opens a dialog to select a image in the structure
//                  * the path is read in and stored to the registry for the next session.

procedure TfrmMain.btnSetPathClick(Sender: TObject);
var
  Reg: TRegistry;
  err: boolean;
begin
  err := false;
  Reg := TRegistry.Create;
  OpenDialog.InitialDir := exepath + '001\';
  MessageDlg('Please select a image of the structure!', mtInformation, [mbOK], 0);
  OpenDialog.Execute;
  path := Copy(OpenDialog.FileName, 0, length(OpenDialog.FileName) - 11);
  if not (Copy(path, length(path), 1) = '\') then
  begin
    MessageDlg('Pfad ungültig' + #13 + #10 +
    'Please select a image of the structure!', mtError, [mbOK], 0);
    err := true;
  end;
  if err = true then
    exit;
  editPath.Text := path + '     ';
  try
    Reg.RootKey := HKEY_CURRENT_USER;
    if Reg.OpenKey('\software\stitcher', True) then
    begin
      Reg.WriteString('FilePath', path);
      Reg.CloseKey;
    end;
  finally
    Reg.Free;
    inherited;
  end;
end;
```

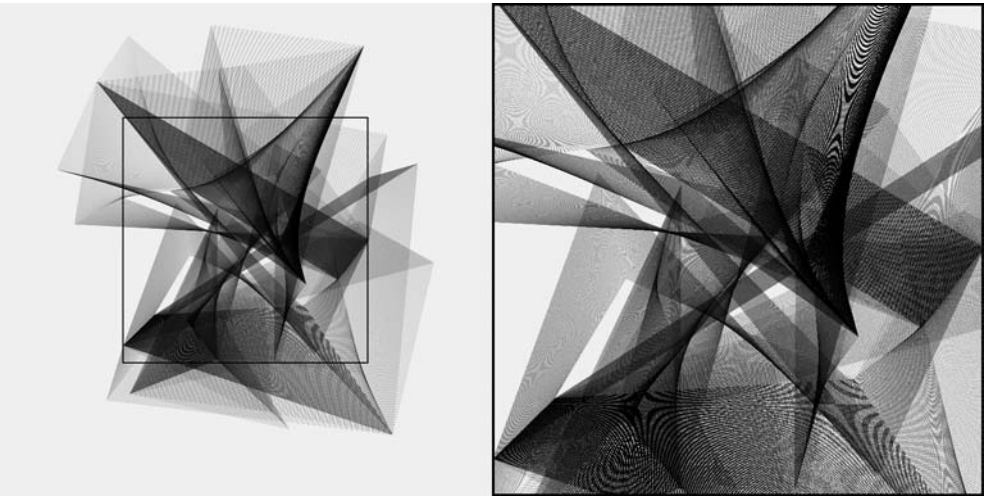

end.

A|3| *Erweiterungen*

A|3|1| *Qualitätsverbesserungen*

Sollen Treppen vermieden werden, sollte jedes exportierte Bild in doppelter Grösse gerendert werden und dann mit einem Bildbearbeitungsprogramm um 50% verkleinert werden.

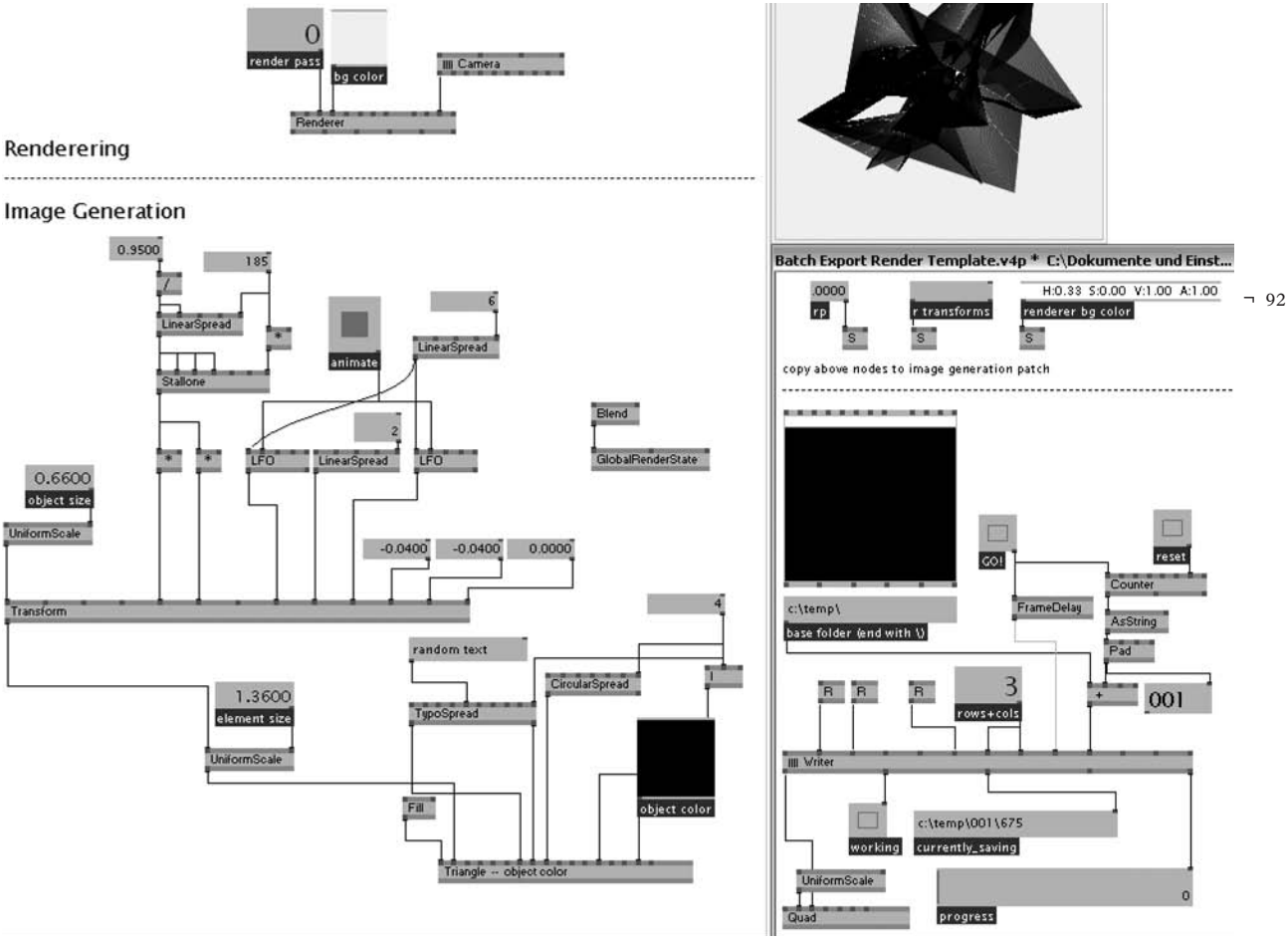
Die meisten Bildbearbeitungsprogramme können dabei die Qualität der Bilder beträchtlich steigern indem alle Kanten geglättet werden.



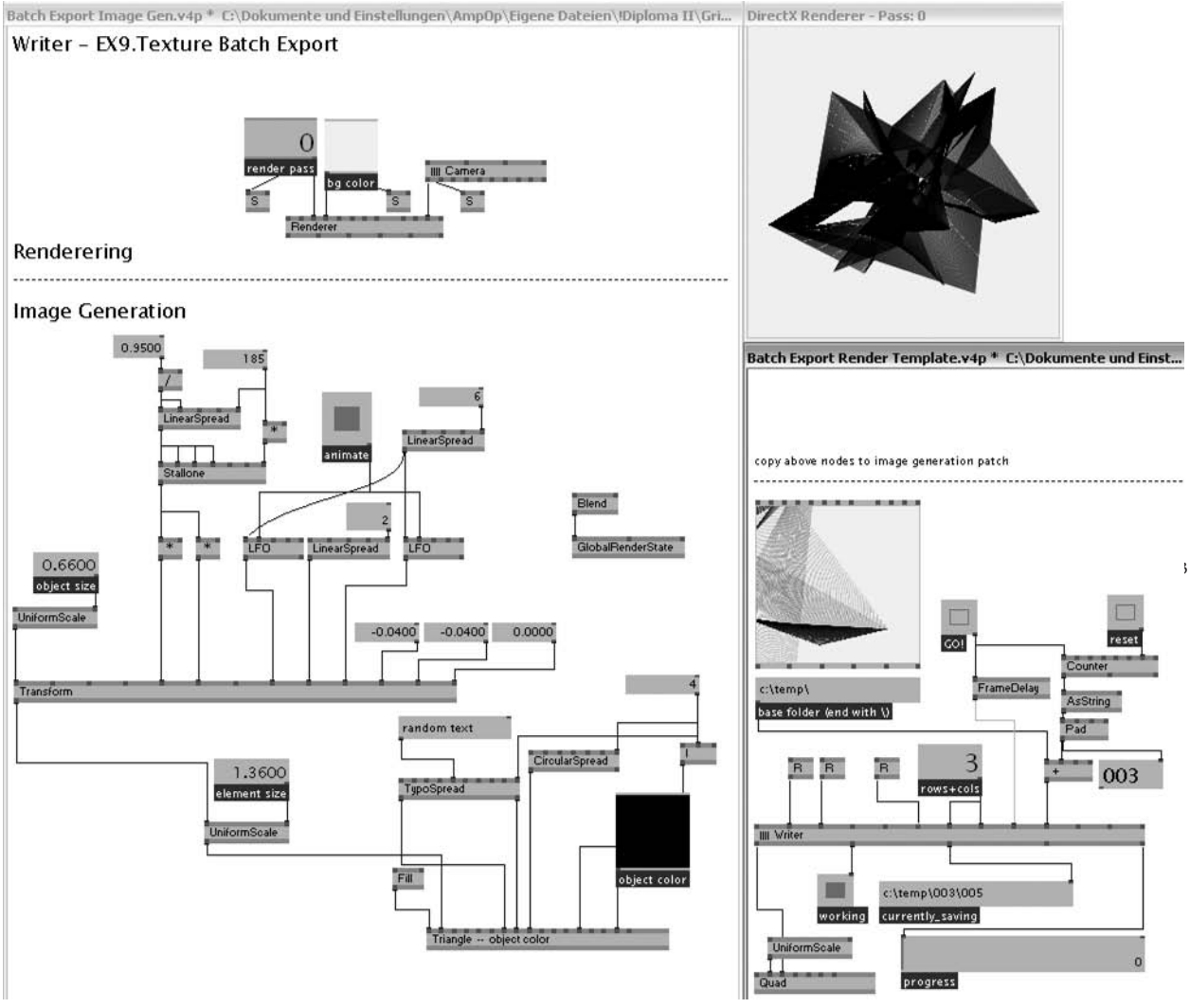
Abb|A|7: Das bikubisch verkleinerte Bild (links) und ein Ausschnitt des unbearbeiteten Bildes (rechts). Letzteres weisst Treppen bei ungeraden Linien auf, während das verkleinerte Bild geglättet worden ist und dadurch alle Treppen entfernt wurden.

A|3|2| Batch export

Auf dem beiliegenden Datenträger befindet sich das „Batch Export Render Template“, welches noch schneller als im Tutorial von Anhang 1 gezeigt, mit einem vorhandenen Bildgenerierungspatch kombiniert werden kann. Zudem besitzt es die Fähigkeit, nach jedem Exportvorgang ein anderes Verzeichnis auszuwählen. So können sehr schnell hintereinander viele Bilder in Einzelteilen exportiert werden.



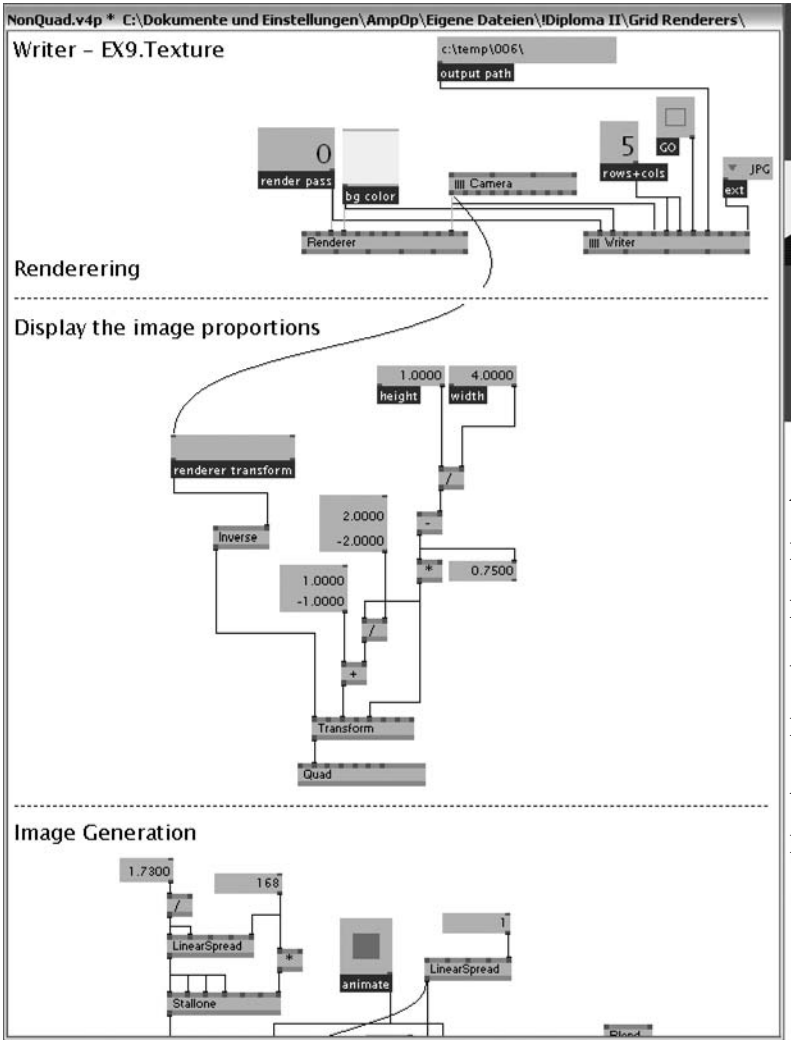
Abb|A|8: Bildgenerierungspatch (links) und das Exporttemplate (rechts). Nur die obersten Knoten müssen in den Bildgenerierungspatch verschoben werden bevor mit dem hochauflösenden Export von beliebig vielen Bildern begonnen werden kann.



Abb|A|9: Das Exporttemplate in Aktion.

A|3|3| *Um nicht quadratische Bilder zu exportieren*

Durch eine kleine Änderung im Writer (EX9.Texture Grid) könnten nicht-quadratische Seitenverhältnisse auch direkt an dessen Renderer (DX9) Knoten eingestellt werden.



Allerdings bietet diese Methode keine zuverlässige Vorschau der Einstellungen, was zu unerwünschten Resultaten führen kann.
Die in Abbildung A|10 verwendete Methode zur Kontrolle des Bildausschnittes ist sicherer.

Abb|A|10: Die Erweiterung zum Anzeigen des korrekten Bildausschnittes, abhängig vom eingegebenen Seitenverhältnis.

A|4| Private E-Mail Korrespondenz mit vvv Entwicklern

A|41| Von: joreg [mailto:joreg@meso.net]

Gesendet: Freitag, 14. Oktober 2005 20:41

halo thomas.

ich probier mich mal stichwortartig in einer erklärung:
die 2048x2048 als beschränkung für die maximale grösse einer textur sind eine reine hardware-beschränkung und treffen so auch nur für ati-karten zu. die meisten nvidia karten scheinen eine maximale auflösung von 4096x4096 zu haben. vielleicht gibt es noch andere grafikkarten hersteller, die andere beschränkungen in ihre produkte einbauen. schwer herauszufinden. sowas steht kaum bei den tech specs im internet. es handelt sich dabei um den wert den die graphikkarten/treiber für die “caps” (=capabilities) MaxTextureWidth und MaxTextureHeight zurückliefern.

in der neuen vvvversion siehst du den wert deiner graphikkarte am zweiten pin eines Device (Auto) knotens.

die antialiasing problematik ist die: in directx funktioniert das antialiasing nicht mit rendertarget-texturen (so heisst das directx-intern, wenn man einen dxtextureknoten anlegt.) also auch hier erstmal eine directx-beschränkung. irgendeinen umständlichen trick gab es da aber...ich denke in einer zukünftigen version würden wir uns das nochmal anschaun.. erstmal doppelt so gross rendern und dann 50% verkleinern. wird auch alles smoother...

die 2er-potenz sache ist eine historische. irgendwie scheint das speichermanagment leichter zu sein, wenn man immer nur mit solchen einheiten zu tun hat. heute können die meisten graphikkarten in vielen fällen auch mit texturen beliebiger grösse umgehen.

sonst noch wer?

alles gute.

joreg.

A|4|2| Von: Sebastian Oschatz [mailto:oschatz@meso.net]

Gesendet: Sonntag, 16. Oktober 2005 19:13

Hallo thomas,

Ja hier auch nochmal eine ergänzung zu joregs antwort. Alles natürlich
ein riesiges thema, zu dem es uferlos viel literatur gibt. Beste
grüsse s.

Da hardware ja immer mit einer endlichen anzahl von transistoren umgehen muss, gibt es immer ganz klare begrenzungen in der anzahl der dinge, die man dann softwaretechnisch unterscheiden kann. ¬ 97

Ein 6502 (wie im C64) verwendet hat 16 Adressleitungen, kann also 65536 Bytes adressieren.
Ein 8086 hat 2?? Adressleitungen kann also XX Bytes adressieren (recherchieren)
640k should be enough for everybody (bill gates; recherchieren)

Ein Pentium hat 32 Adressleitungen, kann also 4294967296 (4GB) adressieren. (auch wenn windows sich eine klaut, und man daher nur die hälfte benutzen kann)
Für jedes Bit, das gleichzeitig bearbeitet werden muss, braucht man eine eigene Leitung und eigene Schaltkreise, was sich in erhöhter komplexität niederschlägt.

Da die grafikkarte ja nur so schnell ist, weil die bits kurze und breite wege auf der karte zurücklegen gibt es eine natürliche limitierung in der anzahl an pixeln, die man sinnvoll bewegen kann. 4096x4096 scheint das zu sein, was heutige hersteller maximal zulassen. Immerhin 16MB bilddaten, die in jedem frame bewegt werden müssen, wenn man mit der textur arbeitet. 8192x8192 sind dann schon 64MB -- kein spielehersteller glaubt, dass sich mit den damit erzielbaren frameraten ein blumentopf gewinnen liesse. Also spart man die leitungen, zählwerke und adressierungslogiken ein.
Alles wertvolle transistoren.

Lagert man speicher auf das motherboard aus, hat man verloren (viele laptops tun das ja) Die geschwindigkeiten die man über den grafikkartenstecker übertragen kann (sei es auch ein superschneller pci-express bus) sind um grössenordungen niedriger, als die, die man direkt auf der grafikkarte zurücklegen kann.

Im wesentlichen eine konsequenz der für heutige belange zu niedrigen lichtgeschwindigkeit (übungsaufgabe: in wievielen centimetern muss man seine logik unterbringen, wenn man z.b. mit 3.6GHz Taktfrequenz arbeiten will)
Die Dinge eng zusammenzurücken ist die eine möglichkeit (die andere ist parallelisieren)
Deswegen sind auch cache-speicher so extrem wichtig. Sie rücken diejenigen bytes, die man häufiger braucht näher an den prozessor ran. Wenn der cache speicher voll ist, bricht die geschwindigkeit rapide ein. Das könnte man vermutlich ganz gut in vvvv untersuchen: mal messungen durchführen, wieviele polygone einer bestimmten texturgrösse man mit einer bestimmten framerate anzeigen kann. Ich tippe es ergibt sich eine treppenstufenkurve. Jede treppenstufe weist darauf hin, das irgendein cache vollläuft.

Zweierpotenzen sind in der hardwaretechnik wichtig, weil sie die zahlen beschreiben, die mit einer *minimalen* anzahl von leitungen eindeutig adressiert werden können. Also eine technische optimierung. Mit 9 leitungen kann ich 512 pixel adressieren (2^9=512). Will ich 513 pixel adressieren, brauche ich 10 leitungen. Mit gleichem aufwand könnte ich also auch 1024 benutzen. Relativ gesehen, sind also 513 teurer als 1024.

Wäre aber spannend zu wissen, wie sich das mit der geschwindigkeit verhält.
Ich hätte immernoch die vermutung, dass 513x513 genauso schnell ist wie 1024x1024.

Beste grüsse,
sebastian oschatz

w w w . m e s o . n e t

.

t ++49 69 24 000 321
f ++49 69 24 000 330
niddastrasse 84hh . 60329 frankfurt main de

A|5| Literaturverzeichnis

ADOBE, Creative Team (2004): Classroom in a Book: Adobe Illustrator CS2. Berkeley, CA: Adobe Press.

BAEZA-YATES, Ricardo (2005): Visual Programming.
URL: http://www.dcc.uchile.cl/~rbaeza/cursos/vp/todo.html (06.11.2005)

BECK, Ernst-Georg (2005): Einführung in die Mikroskopie, Mikroskopische Übungen. Breisach: Becsoft.
URL: http://www.biokurs.de/skripten/bs11-3.htm (06.11.2005)

BEGEL, Andrew (1996): LogoBlocks: A Graphical Programming Language for Interacting with the World. (unveröffentliche Bachelor-Arbeit. Cambridge, MA: Epistemology and Learning Group, MIT Media Laboratory). URL: http://www.cs.berkeley.edu/~abegel/mit/begel-aup.html (06.11.2005)

BRUCKMANN (1981): Bruckmann's Handbuch der Drucktechnik. München: Verlag F. Bruckmann KG.

BURNETT, M. | GOLDBERG, A. | LEWIS, T. (1994): Visual Object-Oriented Programming: Concepts and Environments. Upper Saddle River, NJ: Prentice-Hall.

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

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

GOLIN, E. J. | REISS S. P. (1990): The Specification of Visual Language Syntax. In: Journal Of Visual Languages and Computing. 1(2):141--157, 1990. Amsterdam: Elsevier.

JAKOB, Tanja (2004): Illustrator CS. München: Hanser Fachbuchverlag

MESO (2005a): Propaganda. URL: http://vvvv.meso.net/tiki-index.php?page=Propaganda (06.11.2005)

MESO (2005b): vvvv : Spreads. URL: http://vvvv.meso.net/tiki-index.php?page=Spreads (06.11.2005)

MYERS, B. A. (1990): Taxonomies of Visual Programming and Program Visualization.
In: Journal Of Visual Languages and Computing. 01(2): 97--123, 1990. Amsterdam: Elsevier.

WOLFRAM Research, Inc (2005a): Point. URL: http://mathworld.wolfram.com/Point.html (06.11.2005)

WOLFRAM Research, Inc (2005b): Line. URL: http://mathworld.wolfram.com/Line.html (06.11.2005)

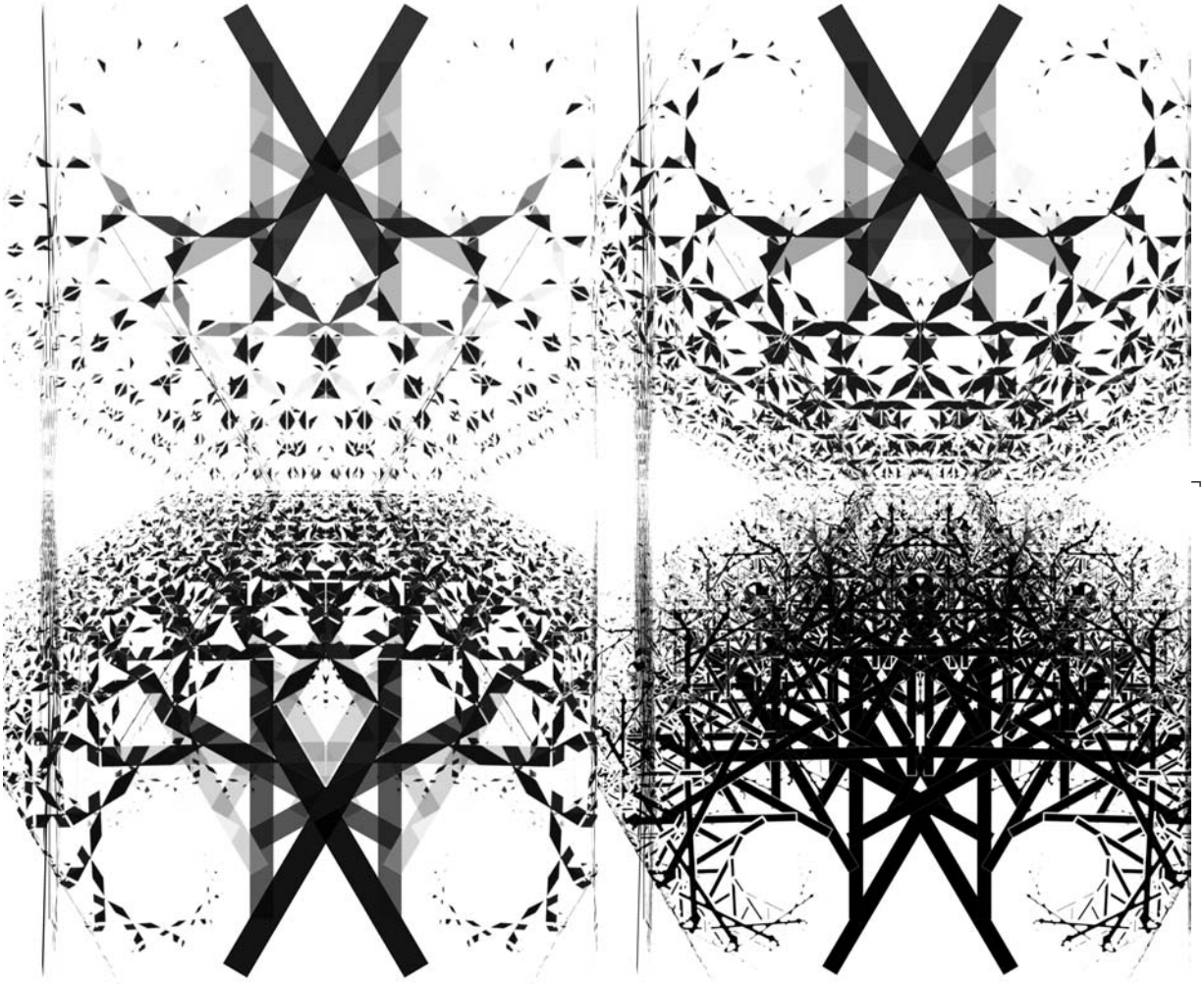
TESCHNER, Helmut (2003): Druck & Medien Technik. Fellbach: Fachschriften-Verlag.

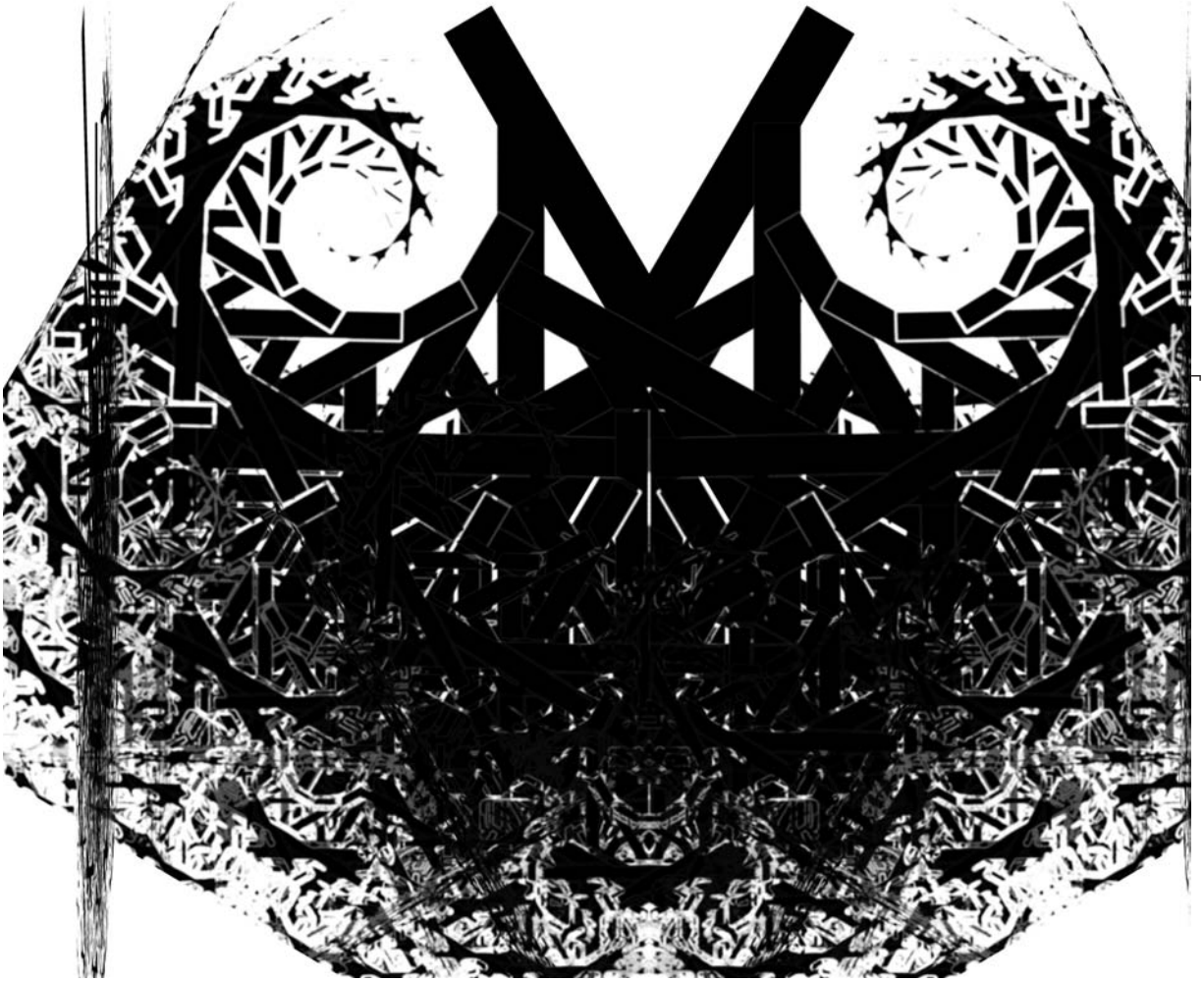
THOMPSON, Robert | FRITCHMAN T., Barbara (2003): PC Hardware in a Nutshell, 3rd Edition.
Sebastopol, CA: O'Reilly Media, Inc.

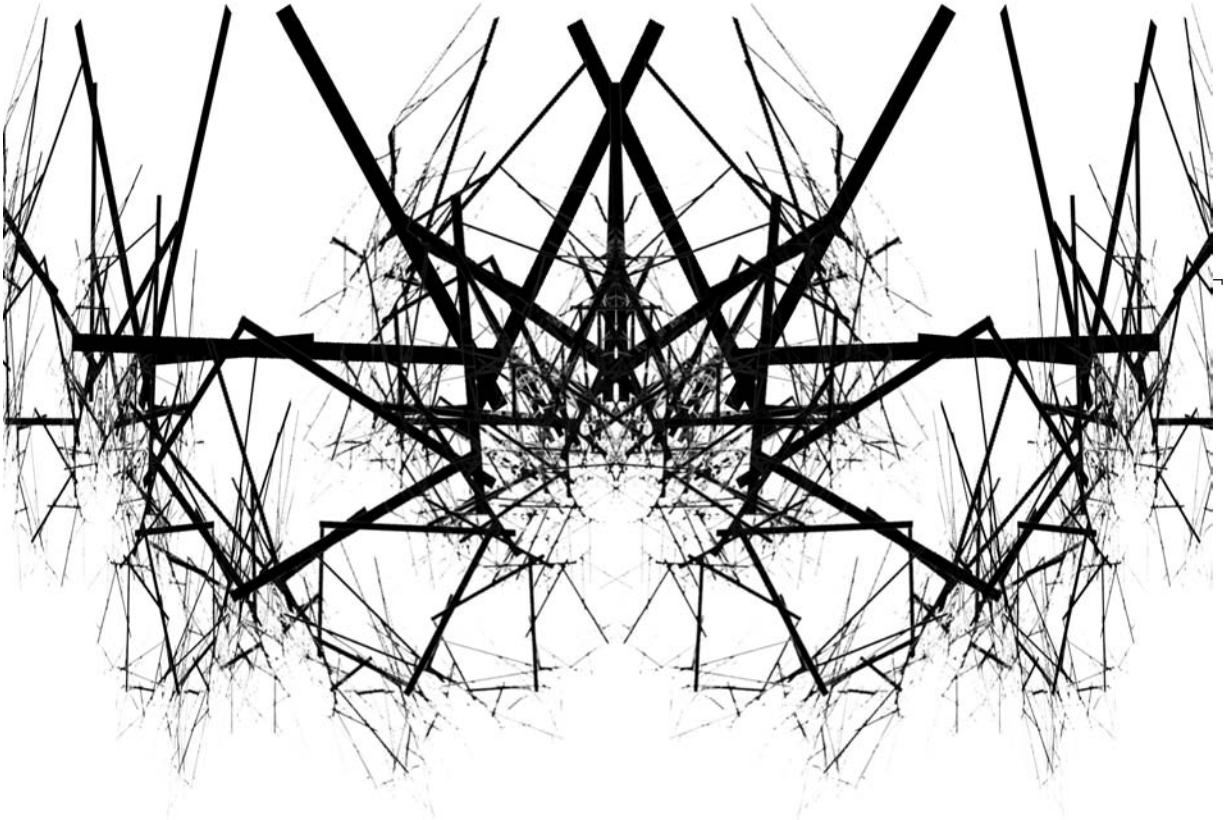
A|6| Abbildungsverzeichnis

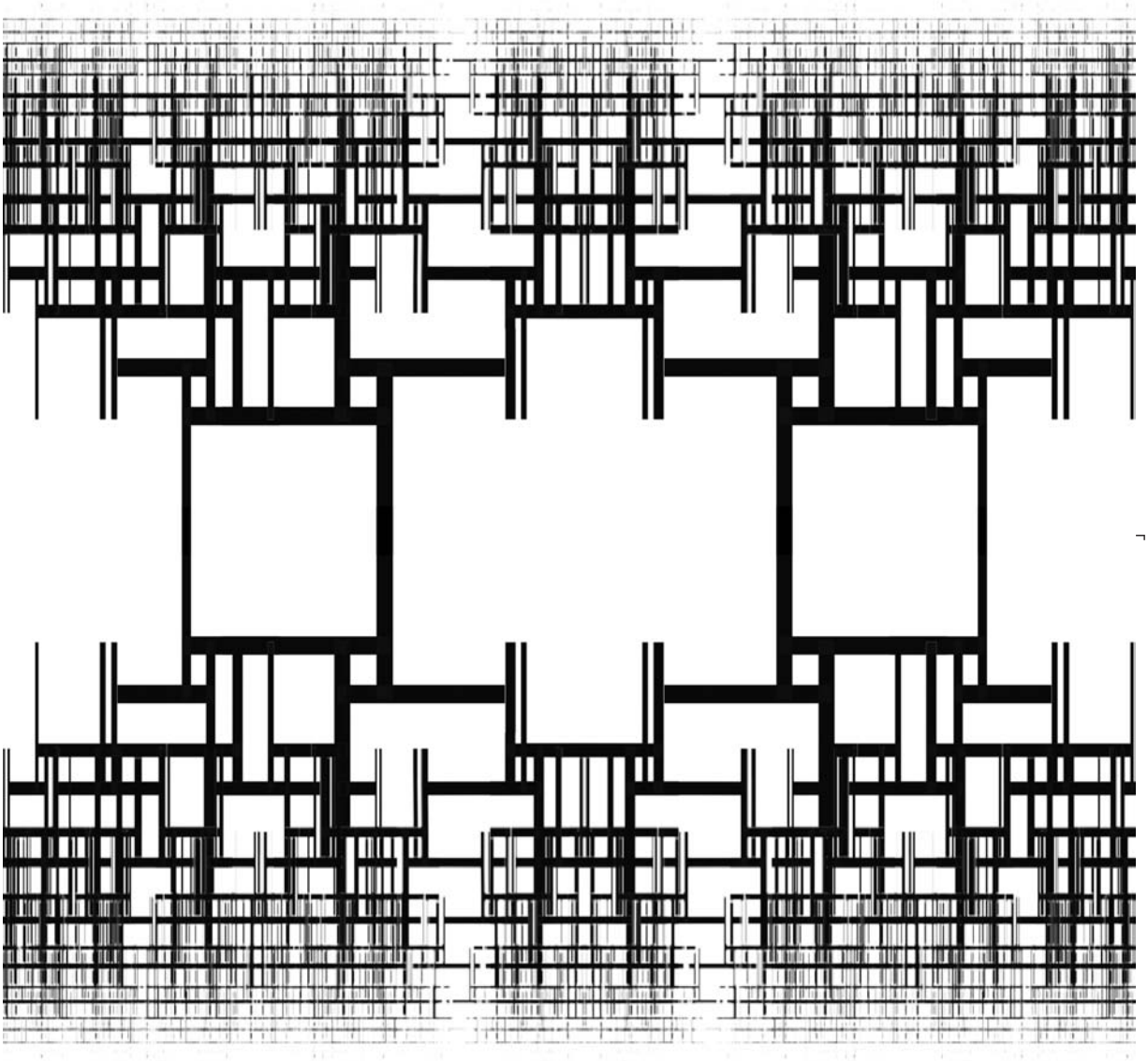
Mit Ausnahme von Abbildung 23, die eine Collage von 4 Grafiken von unbekannten Urhebern ist, wurden alle Abbildungen vom Autor angefertigt.

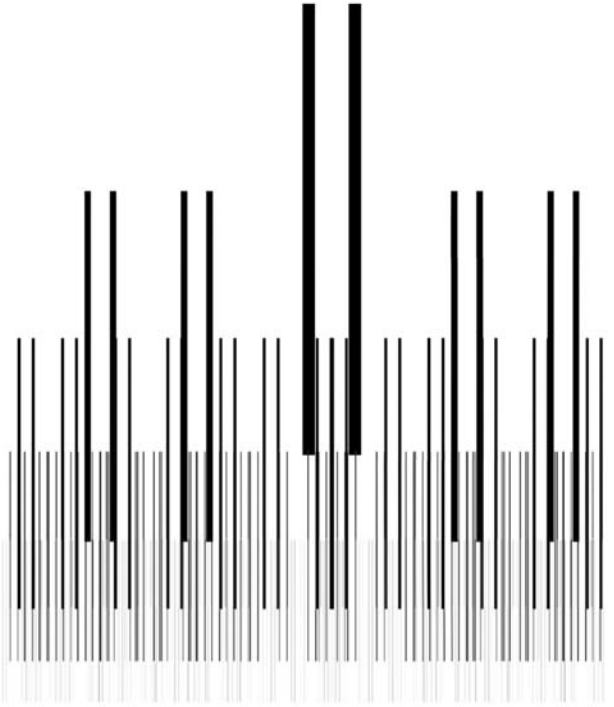
{b} Bilder

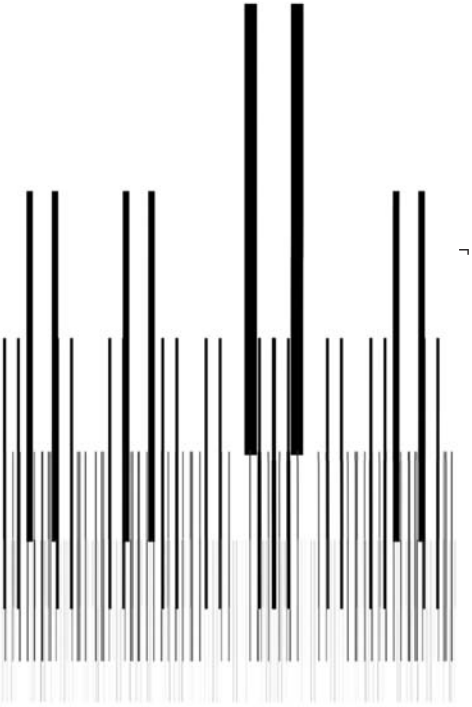


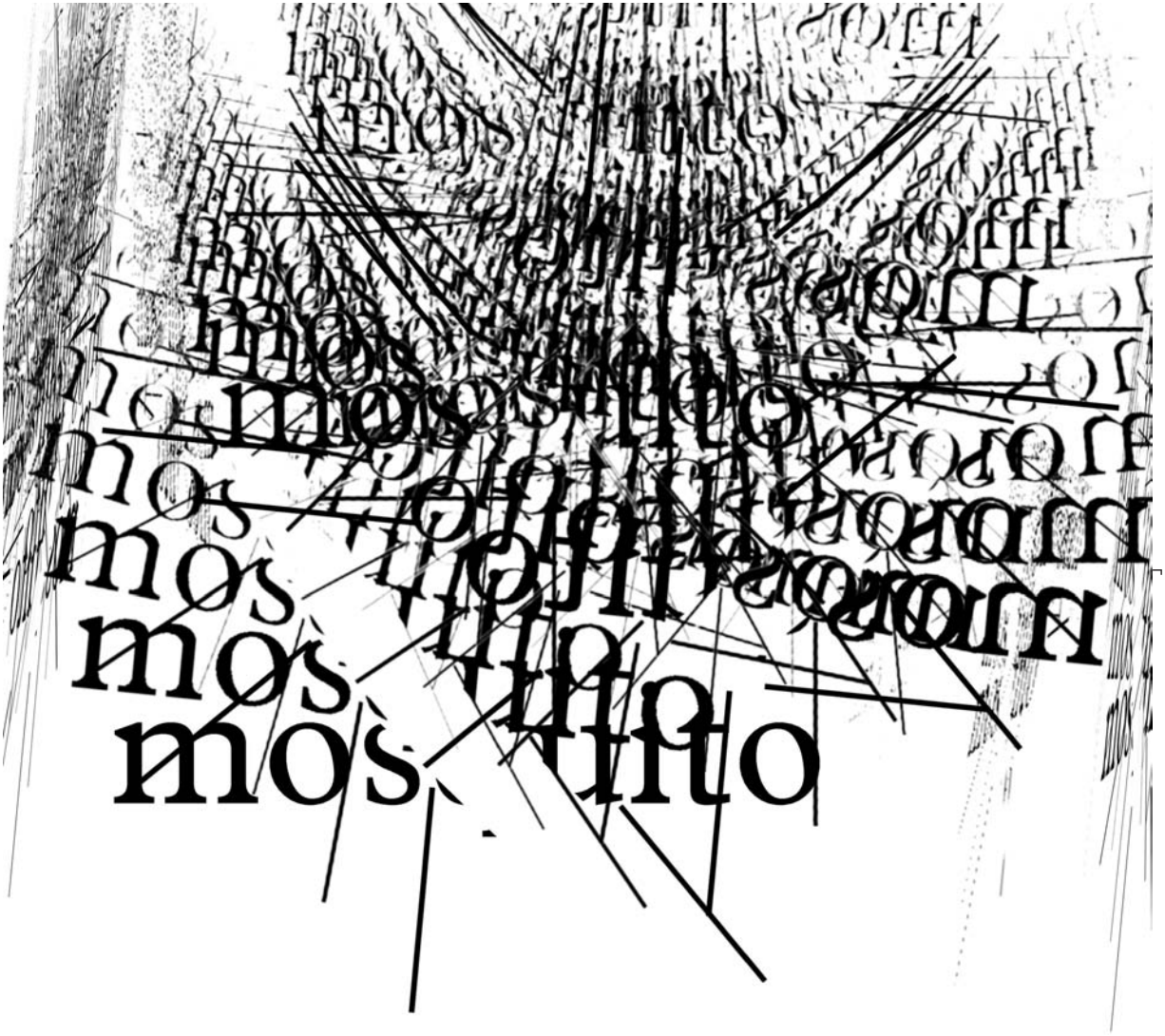


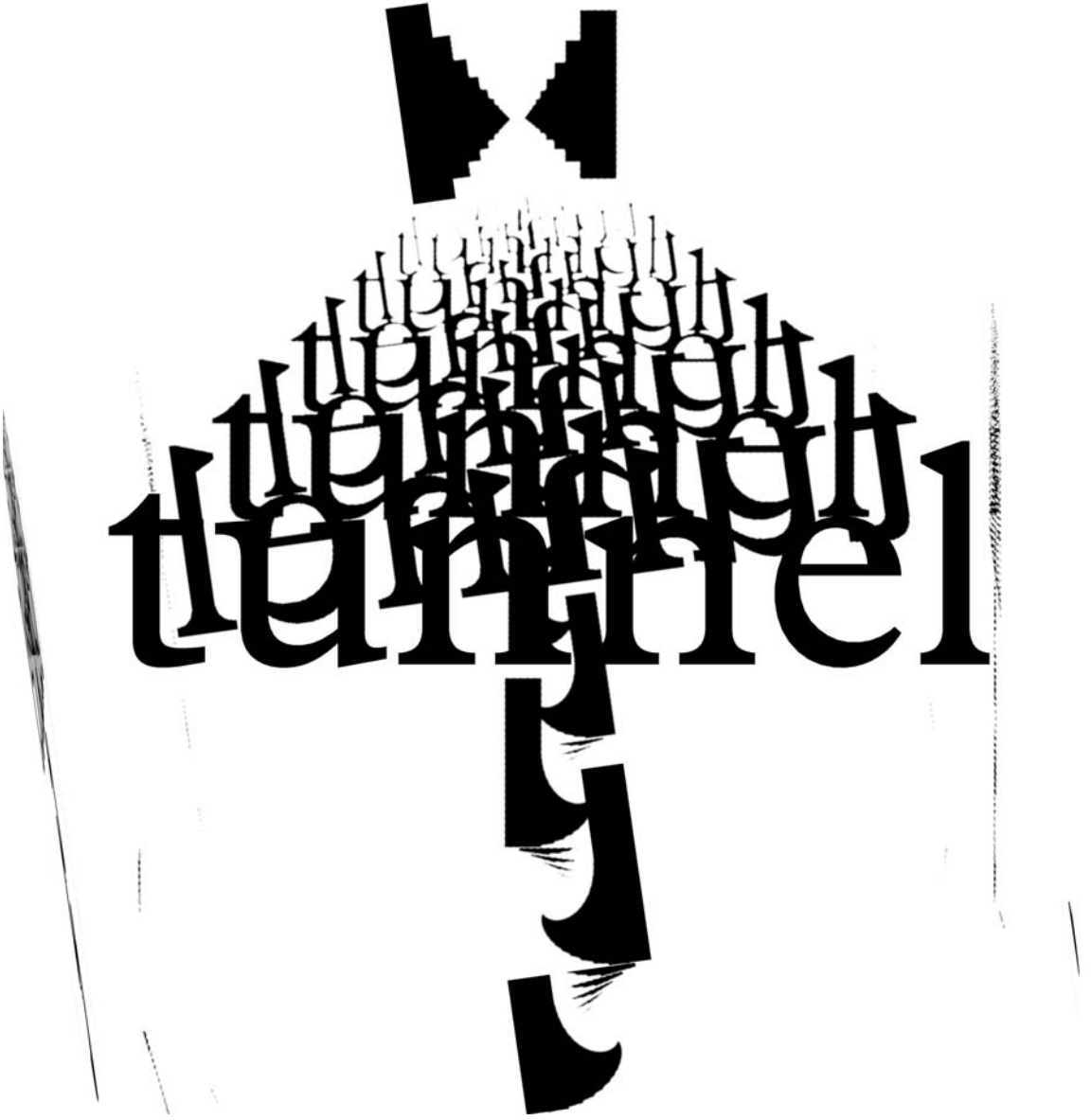


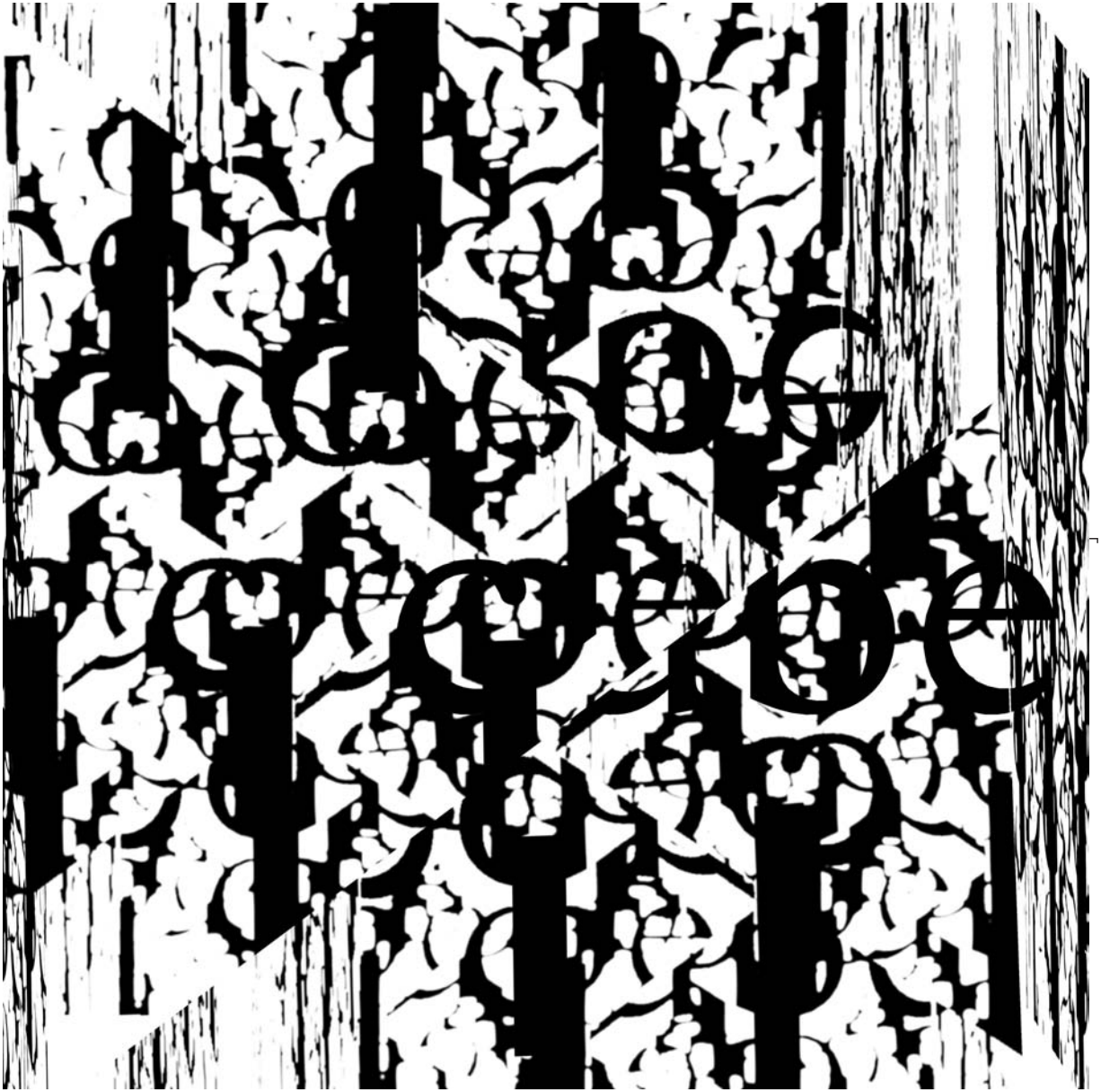


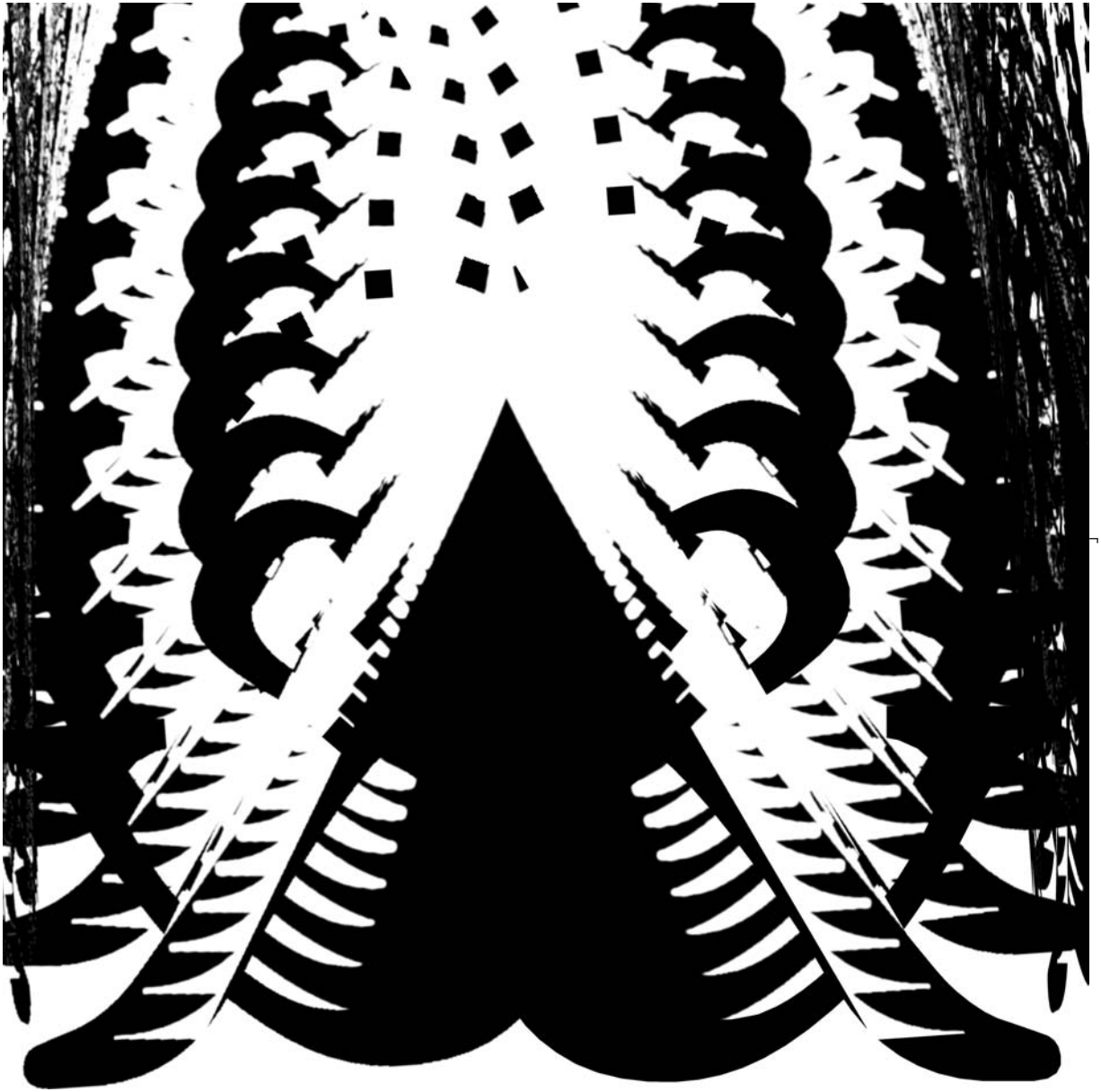


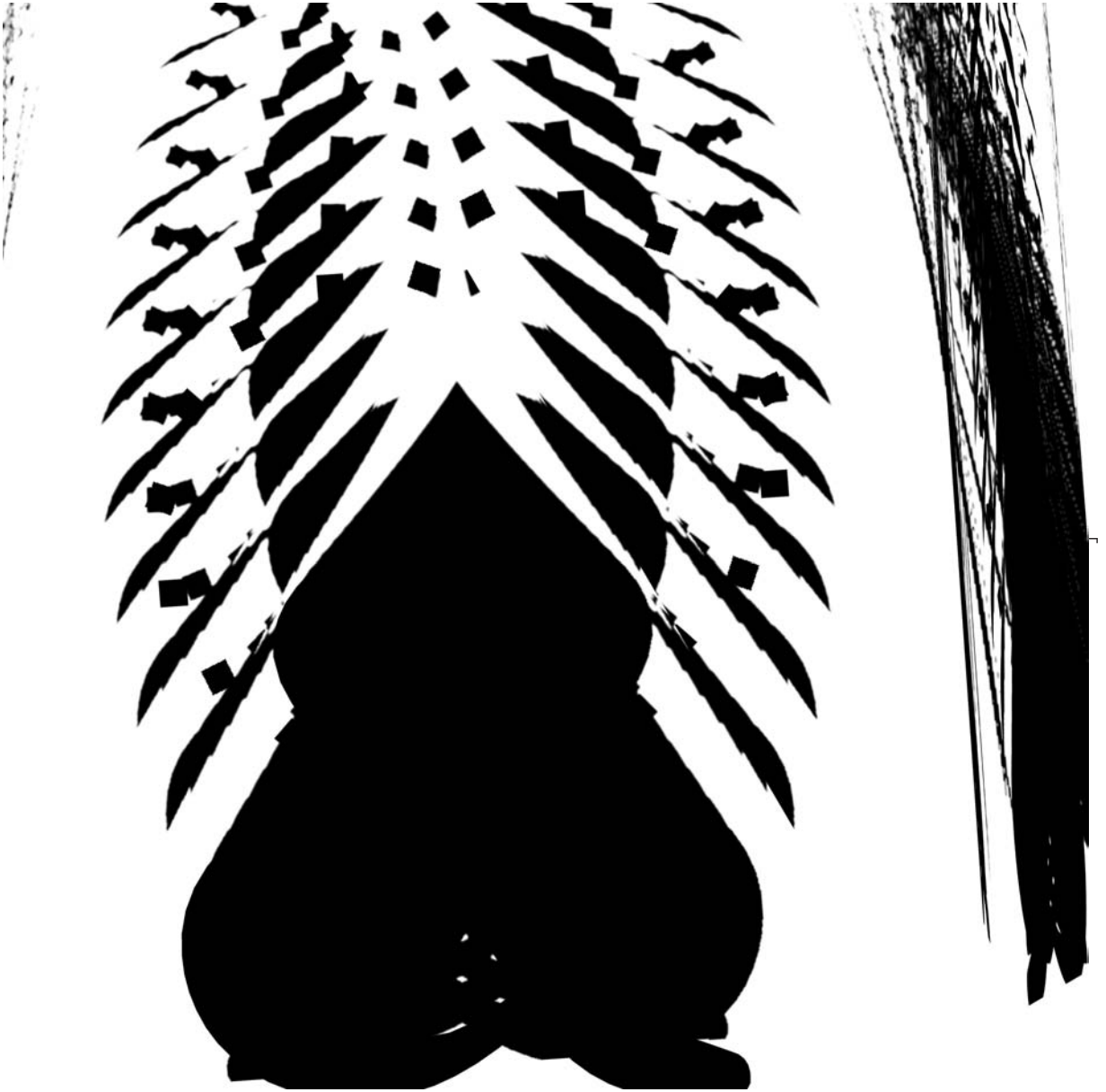


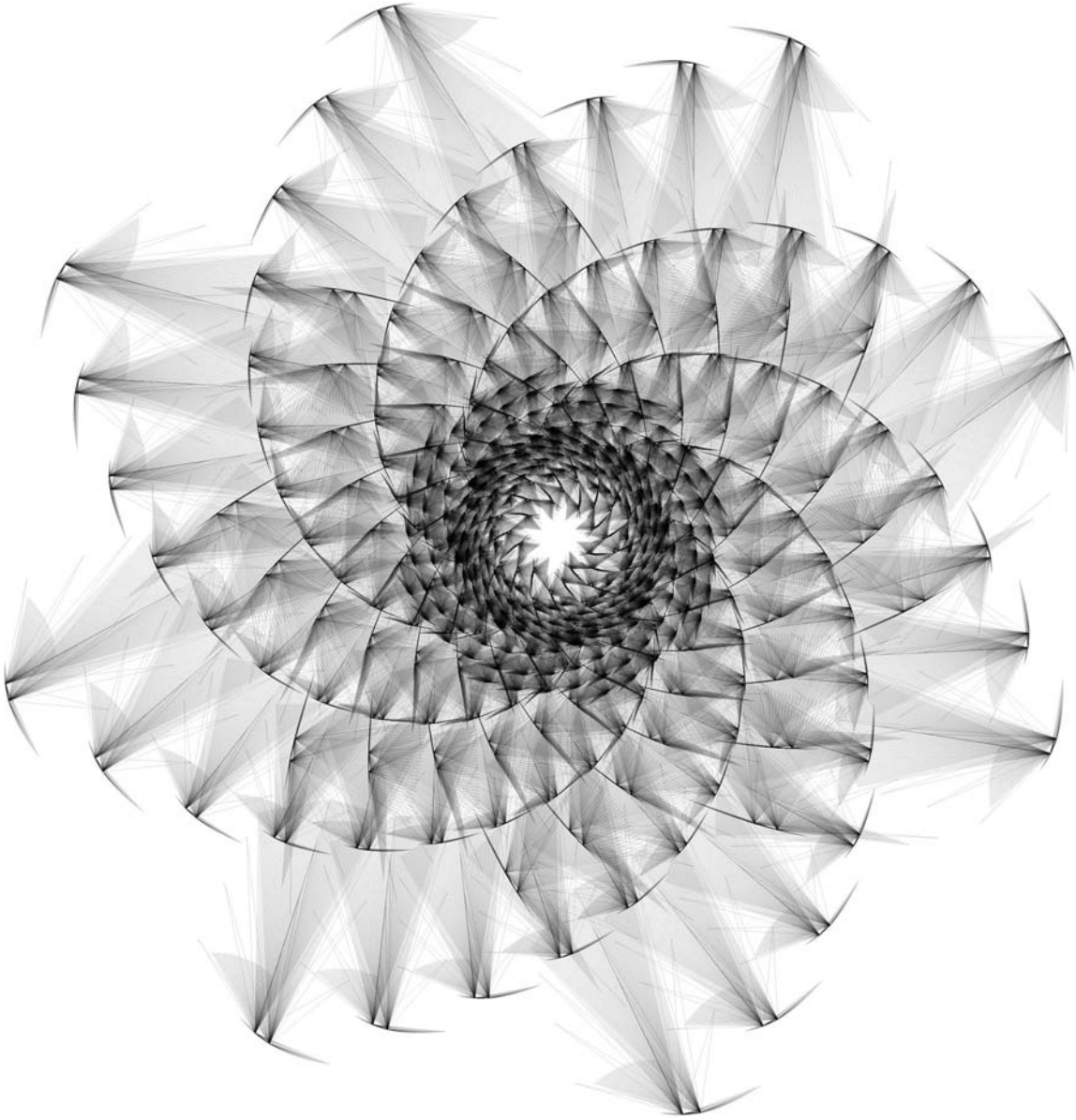

mos
mos
mos
mos

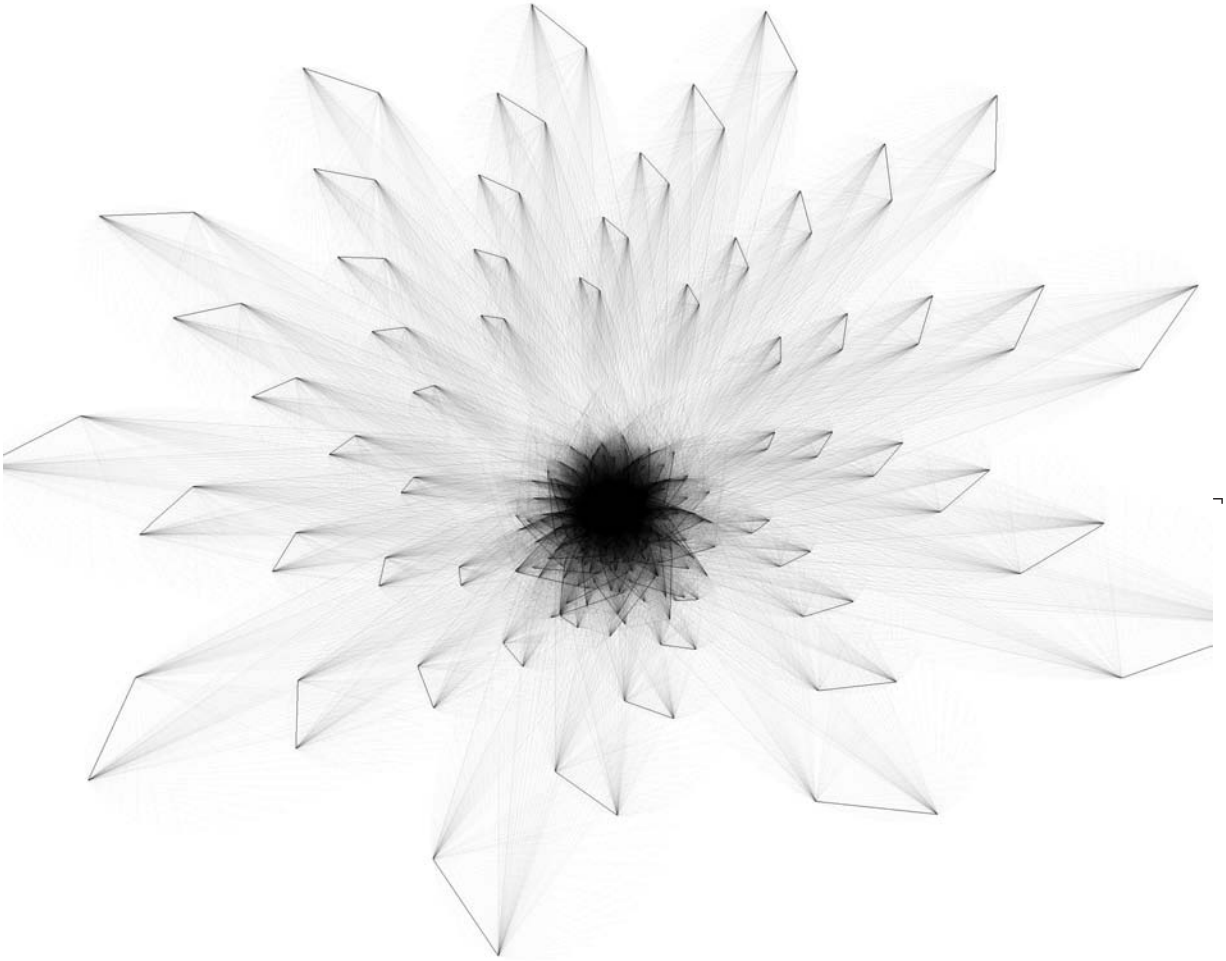

tunnel

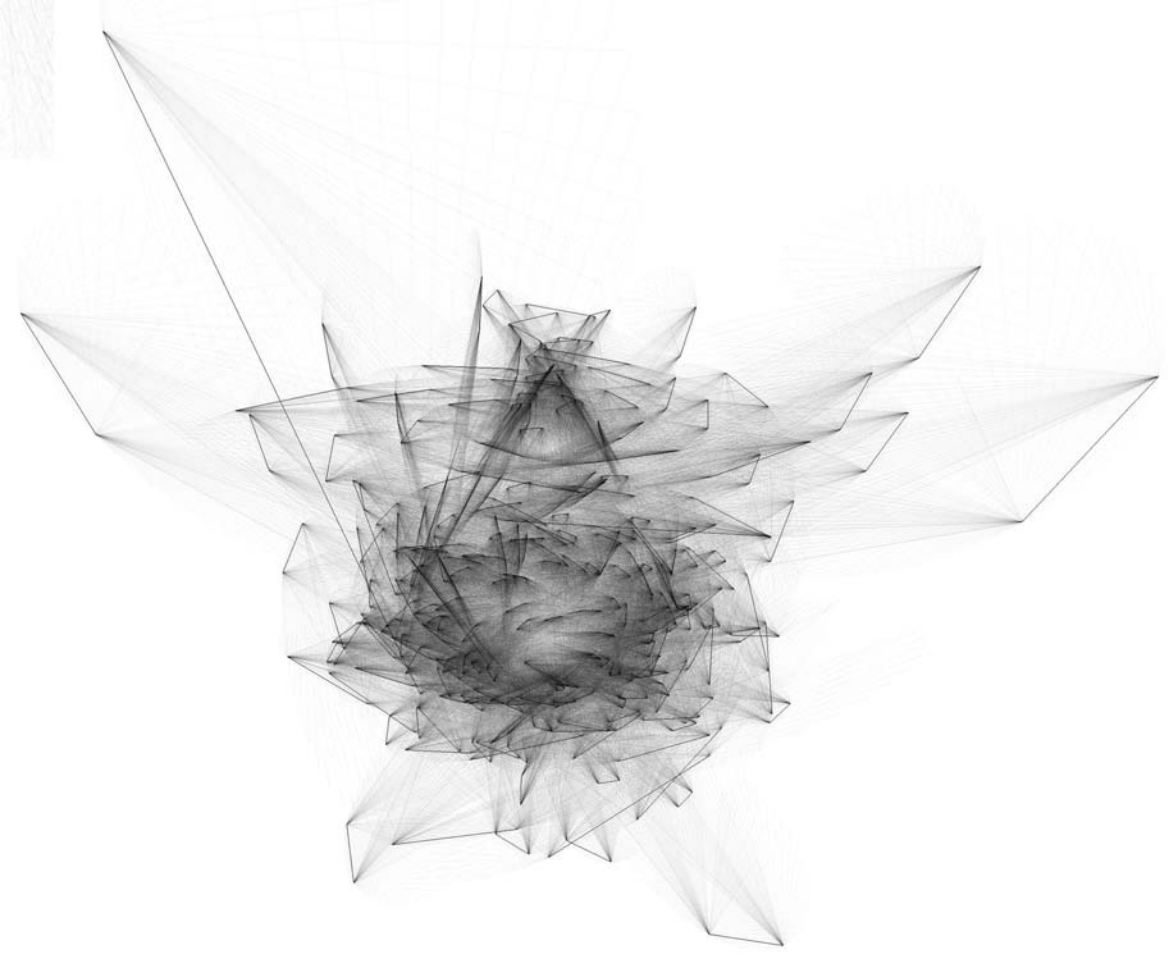


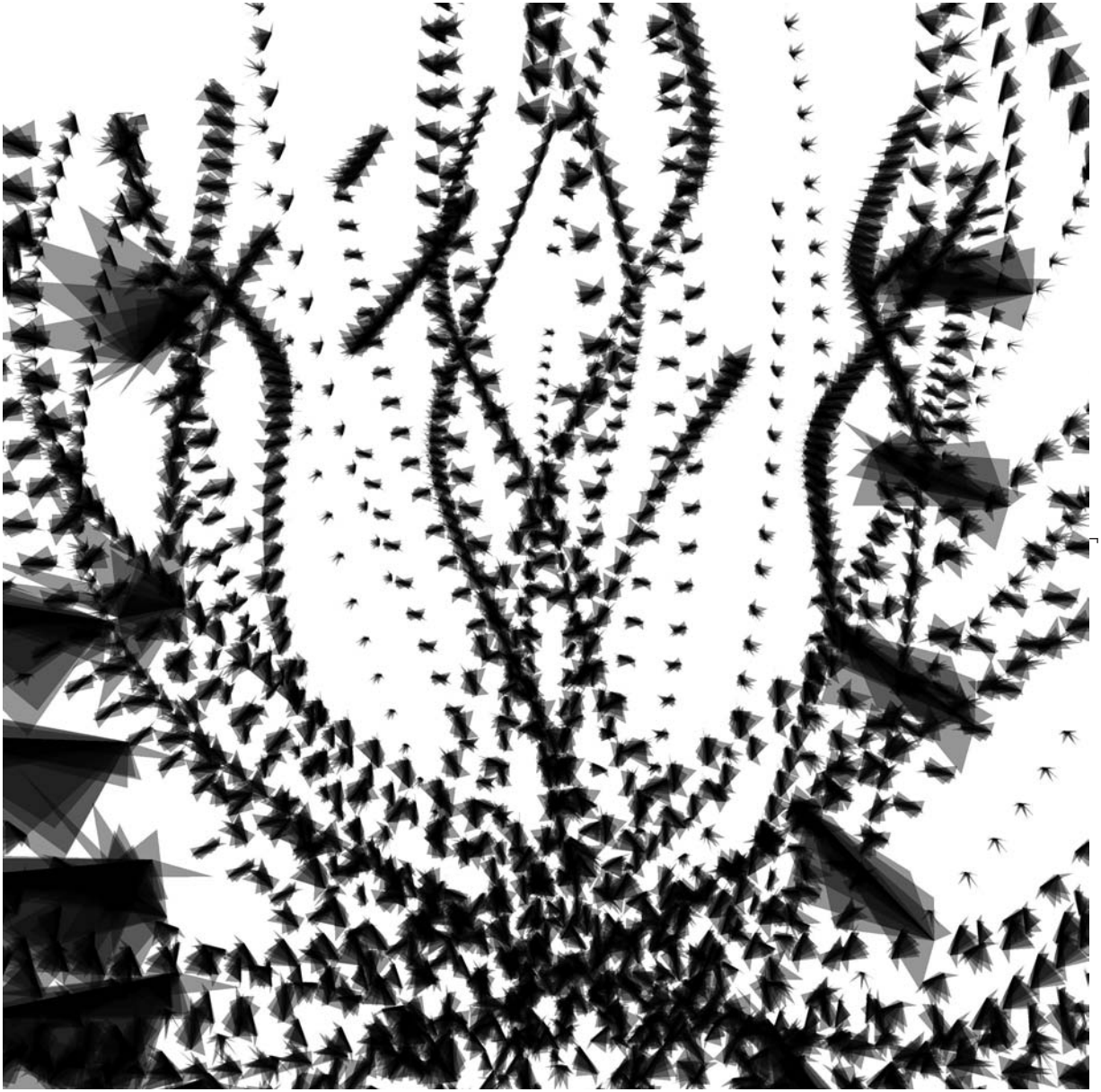


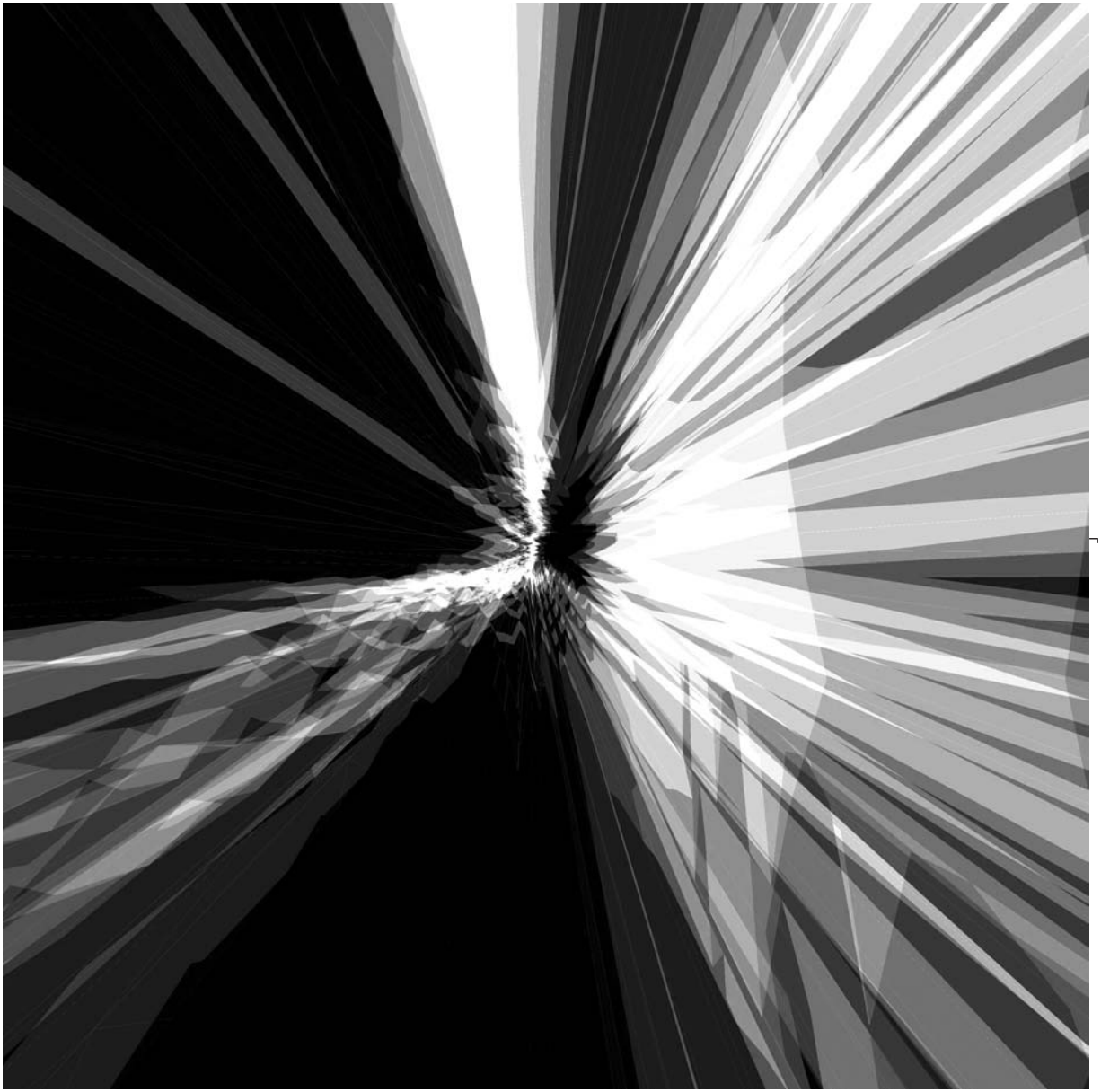


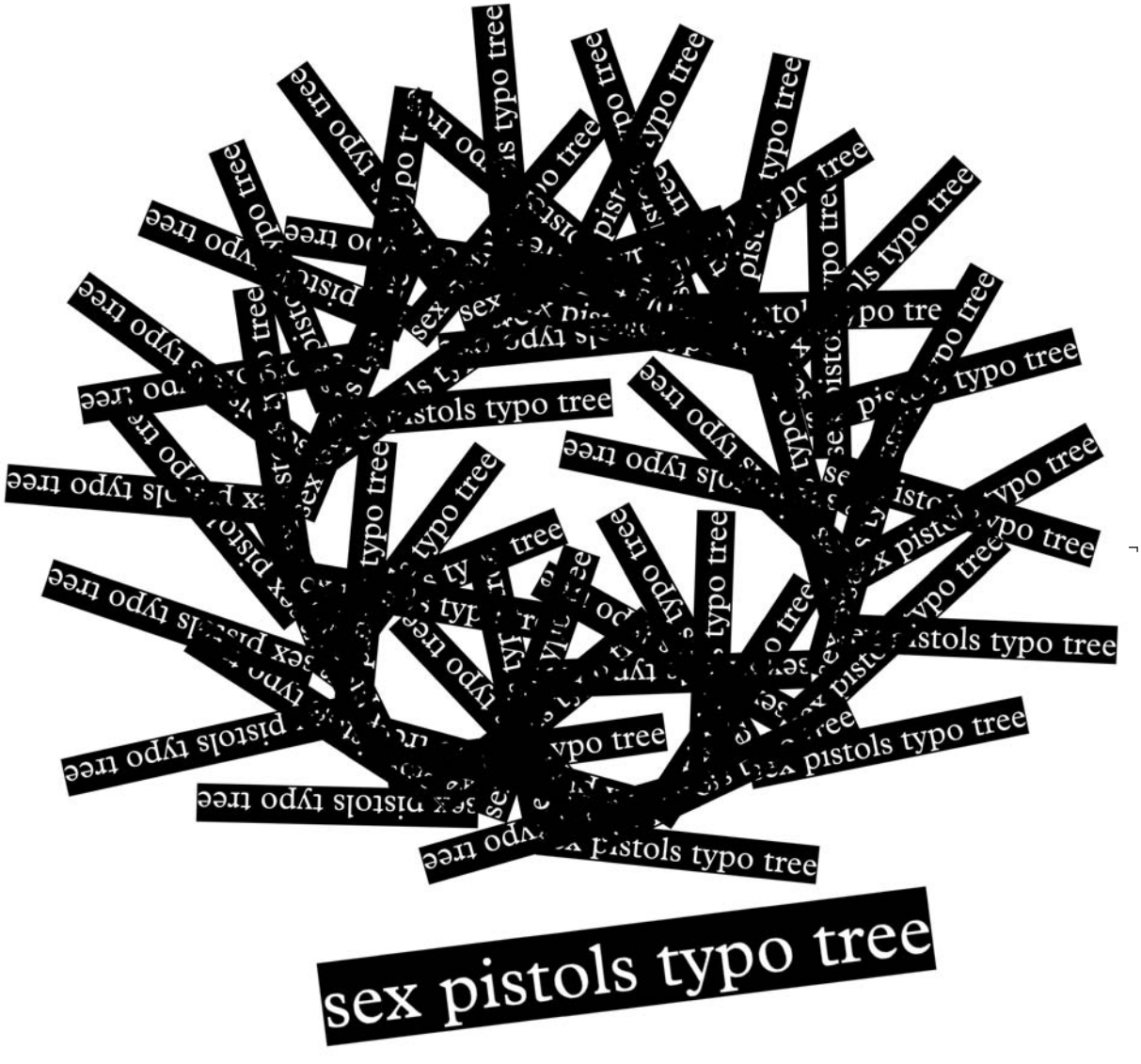


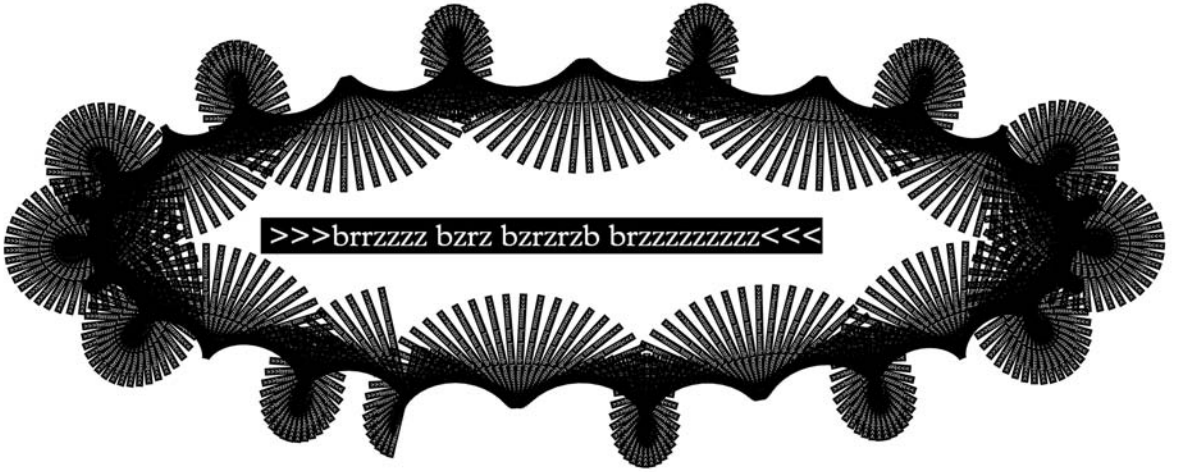


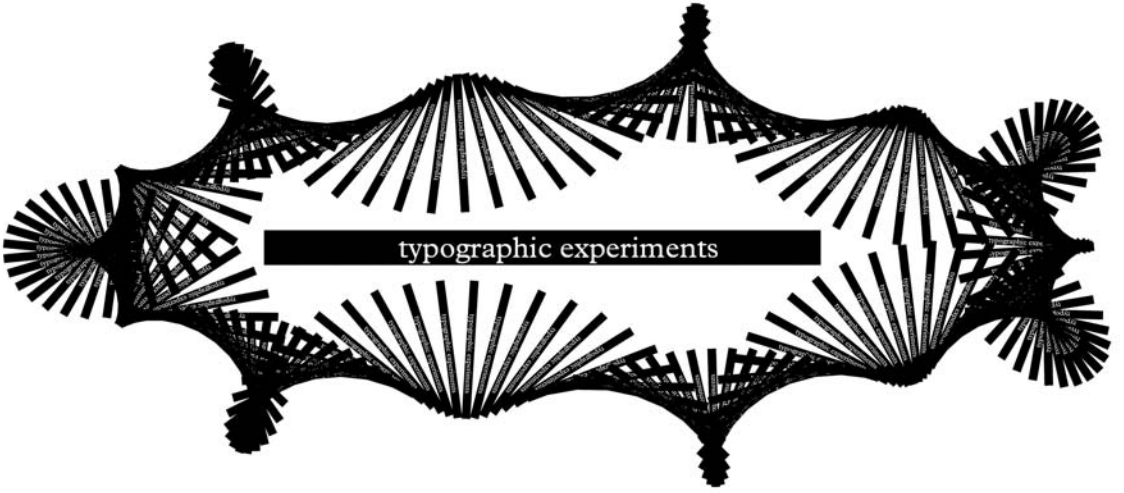


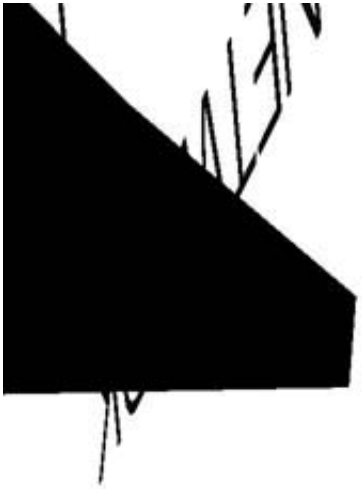





sex pistols typo tree


>>>brrzzzz bzrz bzrzrzb brzzzzzzzzz<<<


typographic experiments




TRUKTVISUALN