MedienPädagogik
Zeitschrift für Theorie und Praxis der Medienbildung
www.medienpaed.com
ISSN 1424-3636

Themenheft Nr. 33: Medienpädagogik und Didaktik der Informatik.
Eine Momentaufnahme disziplinärer Bezüge und schulpraktischer Entwicklungen.
Herausgegeben von Torsten Brinda, Ira Diethelm, Sven Kommer und Klaus Rummler

Algorithmische Kunst als Bildungsgegenstand

Gedanken zu einer fachlichen Bildung über Fächer hinaus

Susan Grabowski und Frieder Nake

Zusammenfassung

Die algorithmische Revolution hat bewirkt, dass heute viele Menschen einen leistungsstarken Computer bei sich tragen und zu Hause weitere stehen haben. Diese Computer sind mit dem Internet zu einem populären Medium verwachsen. Mit der Umwälzung der technischen Grundlagen aller Kultur geht einher eine neue Art des Denkens: Das algorithmische Denken. Nur wenige sind sich bewusst, dass diese Art des Denkens ein Denken auf das Berechenbare, auf die Maschine hin ist – und somit ein Denken der Verengung. Der Fähigkeit des Menschen zur immer fortgesetzten Interpretation setzt die Maschinenwelt die Notwendigkeit der einzigen Determination entgegen. In diesem Widerspruch bewegt sich aktuelle Kultur. In der Kunst begegnet verengende Algorithmik erweiternder Ästhetik. In der über fünfzig Jahre alten algorithmischen Kunst treffen sich beide. Darin sehen wir Chancen für Bildungsprozesse. Was sollen junge Menschen über Wirkungen ihrer Art des Kommunizierens wissen? Wie können sie verstehen, dass das, was sie ständig tun, nur funktioniert, weil Algorithmen und Datenstrukturen es bewirken? Wie können sie zur semiotischen Schicht der Wirklichkeit gelangen, die der Simulation und Vorbereitung physischer Wirklichkeit dient? Immer schon gehören Simulation und Automatisierung zu den Aufgaben der Informatik, Algorithmen und Datenstrukturen sind ihre Mittel. Immer schon gehören inhaltliche Gestaltung und instrumentelle Anwendung zu den Aufgaben von Medienbildung. Was davon muss heute in allgemeine Bildung eingehen und wie? Diese Frage beschäftigt uns im Zuge der aktuellen Einführung von Informatik bereits in der Grundschule. Dieser Essay befasst sich mit dem «algorithmischen Denken», das für das Verständnis der digitalen Technik grundlegend ist. Wir diskutieren dabei die Eignung von Werken aus der algorithmischen Kunst. An Beispielen zeigen wir, wie algorithmisches Denken beim Betrachten von Kunst gelernt werden, und welche Rolle algorithmische Kunst dabei spielen kann. Wir bewegen uns auf der Ebene fachlicher Kompetenzen. Dabei wird deutlich, dass wir für Bildung das Fach hinter uns lassen müssen.

Grabowski, Susan, und Frieder Nake. 2019. «Algorithmische Kunst als Bildungsgegenstand. Gedanken zu einer fachlichen Bildung über Fächer hinaus». *MedienPädagogik* 33, (Februar), 76–101. https://doi.org/10.21240/mpaed/33/2019.03.19.X.

Algorithmic Art in the School Curriculum. Thinking beyond disciplines

Abstract

As a result of the algorithmic revolution, many of us are now carrying a high-performance computer in their pockets, and have others waiting at home. With the Internet, these computers have been integrated into an attractive medium. A new kind of thinking is emerging with this unique turn-over of all the technological infrastructure of culture: algorithmic thinking. Only few realise that this kind of thinking is a thinking towards computability, and to the machine. As such it is narrowing thought. Humans are capable of continued interpretation without limits. The world of machines, on the other hand, must necessarily limit interpretation to unique determination. Current culture is caught in this contradiction. In art, closing-down algorithmics encounters opening-up aesthetics. In algorithmic art, now more than fifty years old, both meet. In this encounter we expect chances for educational processes. What should young people know about impacts of their way of communicating? How can they come to understand that their way of communication depends on the permanent activity of algorithms and data-structures? How can they see that a semiotic layer of reality is simulating and preparing for the changes of the physical world? Simulation and automation have always been subject matter of computing, algorithms and data-structures are its means. Design of contents and instrumental use have always belonged to media education. What of this must today become part of a general education, and how? – We are concerned about this question in the context of introducing the discipline of computing in primary school already. This essay is about «algorithmic thinking» which is basic for an understanding of digital computing. We show examples of how to learn algorithmic thinking by studying works of art, and what role there is for algorithmic art, in particular. Our discussion is on the level of disciplinary skills. But we see how important it becomes to leave behind the confines of discipline if we want to achieve good general education.

Was sollen Kinder vom Computer wissen?

Schulische Diskussionen um Medien- oder Informatische Bildung fragen häufig nach einem umfassenden Verständnis dessen, was Computer bewirken. Angesichts epochaler Umwälzungen aller Kultur durch Informationstechnologie scheint es nicht zu hoch gegriffen, eine neue Art von Aufklärung zu verlangen, wobei wir Aufklärung als «Ausgang des Menschen aus seiner selbstverschuldeten Unmündigkeit» (Kant 1966: 53) verstehen. Unmündigkeit ist für Kant «das Unvermögen, sich seines Verstandes ohne Leitung eines anderen zu bedienen» (ebd.). Im Umgang mit der technisch geprägten Welt mündig zu sein und selbständig (das heisst: kritisch): das wäre das Ziel.

Was aber tun, um dem Computer gegenüber mündig zu werden, selbstbestimmt unabhängig von Experten und Ideologen? Oberflächen der Benutzung reichen dazu nicht aus. Es gilt, den Dingen auf den Grund zu gehen, die «Menschen stärken, die Sachen klären», wie von Hentig dies 1985 postuliert hat.

Soll das der Ruf nach dem Programmieren sein? Um den Computer als Maschine besonderer Art zu begreifen, tut ein gewisses Mass an Programmier-Fertigkeit gut. Brächte das die Sache aber schon auf den Punkt? Wäre die Konsequenz, dass alle Schüler und Schülerinnen heute das Programmieren lernen müssen, und das schon in der Grundschule? Oder ist eine Beschäftigung mit dem Phänomen des Berechenbaren auf tieferer Ebene notwendig, als das Programmieren sie bietet?

Wie abstrakt, wie konkret?

Aus langer Erfahrung mit Einführungskursen (allerdings gewonnen mit Erwachsenen unterschiedlichen Vorwissens) erscheint es uns ratsam, nicht sogleich mit konkreten Formen des Programmierens zu beginnen. Denn diese Formen prägen leicht alles weitere Denken und behindern ein allgemeineres Begreifen fundamentaler Ideen des Computing.

So berichten zum Beispiel Studierende der PH Zürich bezüglich ihrer Erfahrungen zum Thema iterierte Strukturen («Schleifen»), dass sie selbst und ihre Schüler statt der abstrakten Darstellung der Iteration ein Schema der Programmiersprache gelernt hätten. Ähnliches konnten wir bei Lernenden der Universität Bremen beobachten: Sie lernten Programm-Schemata auswendig, anstatt die Bedeutung der Iteration für jede Berechnung zu begreifen. Abstrahierend zu denken, verlangt das Programmieren; die konkrete Formulierung im Programm sollte Nebensache bleiben.

Wir schliessen keinesfalls die praktischen Aspekte der Programmierung aus, wollen aber den Menschen grundlegende Prinzipien näher bringen: Denken im System und nicht als Schema.

Computer sind *semiotische Maschinen* (Nake 2009): Sie bearbeiten Zeichen und Zeichenprozesse, jedoch in der reduzierten Form von Signalen. Klassische Maschinen hingegen bearbeiten stoffliche Prozesse. Computer benötigen berechenbare *Beschreibungen* der Operationen, die sie ausführen. Dadurch sind Menschen, die Computer einrichten und verwenden, prinzipiell fern von ihren Gegenständen.

Mit Computern etwas anstellen zu wollen, bedeutet immer, aus der Ferne mit den Dingen umzugehen, um die es geht, also etwa mit einer Zeichnung. Alles ist abstrakter als in der gewohnten Welt: z.B. ist eine gerade Linie die Verbindung zweier Punkte in einem Koordinatensystem. Das macht die schwierigen wie die schönen Seiten des *Computing* aus.

Es schadet nicht, ein Programm zu entwickeln. Dazu ist ein hohes Mass an Abstraktion verlangt. Ein gutes Gespür dafür muss sein. Zu erreichen ist es auf vielfache Weise. Das Wie kennen wir gut noch nicht. Wir wissen: Das verlangt Übung, um die verborgenen Geheimnisse zu entdecken. In der Musik spielt der eine Noten, der andere aber ein Stück. Ihr Verständnis von Musik wird weit auseinander liegen. Rein äusserlich tun beide das Gleiche oder Ähnliches. Ihre Auffassungen aber sind verschiedene. Viel ist seit einigen Jahren von sog. Programmierumgebungen zu erwarten. Sie helfen sehr. Die *eine beste* Umgebung aber gibt es nicht.

Eine Programmiersprache und -umgebung festzulegen, die für alle Schülerinnen und Schüler einer Altersgruppe obligatorisch wäre, halten wir eher für falsch. Logo war einmal die Wahl – falls es um Geometrie ging. Pascal war favorisiert mit guten Gründen, Basic manchmal. Die Einführung des *drag & drop Programming* durch «AgentSheets» (Repenning 1991; 1993) hat eine Ebene des Programmierens ohne symbolische Befehlsfolgen eröffnet. – Trotzdem ist die Diskussion um eine «Informationstechnische Grundbildung» in der BRD in den 1990er Jahren fast folgenlos geblieben (s. Wilkens 2000). Aktuell gibt es eine neue Diskussion um die Einführung von Informatik an den Grundschulen. Dazu fragen wir, welche Kompetenzen und Inhalte es sind, die unabhängig von einer Programmsyntax ermöglichen, Computerdinge zu begreifen? Welches sind die bildenden Inhalte, mit denen Kinder in die Zukunft gehen sollen?

Von K–12 lernen?

Ein Blick in die USA mag anregend sein. Dort wird in der *primary and secondary school* («K–12») ein diese Jahre umfassendes «Computer Science Framework» propagiert. Es soll auf verschiedenen Ebenen und für verschiedene Berufsfelder Kinder und Jugendliche auf das vorbereiten, was die schnelle Welt des Computing uns allen abverlangt. Das Konzept eines *computational thinking* (Wing 2006) als Basiskompetenz für alle Schülerinnen und Schüler scheint den Kern auszumachen. Die Industrie unterstützt das sehr. «Computational thinking is a fundamental skill for everyone, not just for computer scientists. To reading, writing, and arithmetic, we should add computational thinking to every child's analytical ability» (Wing 2006, 33). Unter «computational thinking» wird verstanden, Probleme zu formulieren und deren Lösungen in berechenbaren Schritten (Algorithmen) zu repräsentieren, ob diese nun von Maschinen oder Menschen ausgeführt werden (K–12 2016, 68f).

Computational thinking ist ein mentales Problemlösen. Es beginnt *vor* dem Programmieren, denkt aber die operativen Bedingungen des Computers mit (ebd.). Es geht um Konzepte des Programmierens im Denken auf die Maschine hin, die das Mass des Erfolges ist: sie muss so funktionieren, wie das Programm es vorschreibt.

Das erfordert implizit ein Denken über das eigene Denken. Das ist grundlegend für alles, was mit Computern geschieht. *Computational thinking* verlangt, Wege zur Lösung von Problemen zu formulieren, die auch wir selbst gehen könnten, *ohne* Computer, aber doch ganz nah am Computer – ob Aufgaben aus der Mathematik stammen oder der Kunst.

Alexander Repenning betont, es ginge beim *computational thinking* nicht darum, *wie* ein Computer, sondern gemeinsam *mit* dem Computer zu denken (Repenning 2015, 8). Gemeinsam mit dem Computer zu denken, kann naturgemäss nur metaphorisch gemeint sein. Denn *denken* kann der Computer nicht. Dagegen braucht der Mensch ein gewisses Mass an Denken, das ein Computer täte, wenn er es könnte. Dieses kann nur ein kontextfreies Denken sein, ein zur berechenbaren Operation herabgesunkenes Denken.

Auch wenn Repennings *computational-thinking-tool* «*AgentCubes*» vorrangig dazu dient, «computational thinking patterns» (vgl. ebd., 17f.) spielerisch zu entdecken, bleibt das Miteinander-Denken mit dem *Tool* ein schöner Wunsch auf niedriger Ebene denkender Tätigkeit, da das *Tool* i.d.R. zur Lösung zunächst vorgegebener Aufgaben in einer doch begrenzten Umgebung[1] eingesetzt wird, wie in Spielen wie *Frogger* oder *Pacman*. Die Frage ist, wie es vom spielerischen Nachahmen zum unabhängigen Schaffen geht, wenn das Entwicklungsfeld begrenzt ist? Wie entwickeln Studierende und Kinder über Pacman hinaus eigene Projekte? Immerhin wissen wir, dass jeder Umgang mit einem Werkzeug eigenes Denken und möglicherweise kreatives Handeln fördert.

Abb. 1.: Kernpraktiken des Computing (K–12 2016, 68).

1 Z.B. können mit *AgentCubes* keine Liniengrafiken erzeugt werden.

Abb. 1 zeigt in den weissen Kreisen 3 bis 6 Kernpraktiken von «computational thinking» wie das K–12 Konzept sie sieht. Eine Kernpraktik ist *Erkennen und Definieren berechenbarer Probleme* (Nr. 3). Die Menschen sollen lernen, berechenbare von nicht-berechenbaren Dingen zu unterscheiden. Können Lernerfolg oder Kreativität eines Kindes berechnet werden? Kann berechnet werden, wie ein Läufer seine Leistung steigern kann? Welche berechenbaren Modelle führen zur Wettervorhersage?

Der Computer ist als *Maschine der Berechenbarkeit* konstruiert worden. Berechenbarkeit im mathematischen Sinn wird zu einem zentralen Thema von Bildung unserer Zeit. Denn nur das, was berechenbar formuliert werden kann, kann Gegenstand der Bearbeitung durch Computer werden.

Das heisst, dass kein Ding oder Prozess unmittelbar Gegenstand der Bearbeitung durch Computer werden kann. Immer muss das Ding oder der Prozess erst durch eine Beschreibung (ein Modell) vertreten werden. Dinge und Prozesse müssen durch Zeichen erfasst werden, wenn sie Gegenstand algorithmischer Behandlung werden sollen.

Der Begriff der Berechenbarkeit als mathematisch exakter Begriff geht auf Alan Turing, Emil Post und Alonzo Church[2] zurück. Was der Begriff im Einzelnen bedeutet, wollen wir hier nicht ausführen. Hingewiesen sei nur auf Turings anschaulichen Zugang zum Begriff (Turing 1936).

Turing beschrieb eine *Papier-Maschine*, die bald nach ihm *Turing-Maschine* genannt wurde. Sein Anliegen war es, das Handeln eines Menschen zu beschreiben, der rechnet. Unabhängig von allen wirklich rechnenden Menschen musste die Beschreibung sein, aber so, dass ihre Ergebnisse mit denen aller wirklich rechnenden Menschen übereinstimmten. Das Rechnen *als solches* wollte Turing nachbilden[3].

Sein Ergebnis zeigt, dass «berechenbar» ist, was sich auf sehr wenige und sehr einfache *Grundfunktionen* und wenige *Regeln* zurückführen lässt. Ein Ding oder einen Prozess berechenbar zu formulieren, verlangt stets, ihn drei Reduktionen zu unterwerfen (Nake und Grabowski 2001):

1. Die *semiotische* Reduktion macht aus Dingen und Prozessen Zeichen. Dinge und Prozesse verlieren all ihre Stofflichkeit;
2. Die *syntaktische* Reduktion reduziert Zeichen auf ihre syntaktische Dimension. Sie bleiben Signale – das, was bezeichnet. Zeichen verlieren mit ihrer semantischen und pragmatischen Dimension alles, was wir unter «Bedeutung» fassen;
3. Die *algorithmische* Reduktion schliesslich lässt nur das noch übrig, was berechenbar ist an Zeichen und Prozess.

2 Church formulierte die (unbeweisbare) These, dass alles, was überhaupt berechenbar ist, auch von einer Turing-Maschine berechnet werden kann.

3 Anschauliche Beispiele zur Turing-Maschine finden sich z.B. bei Weizenbaum (1978) oder aktueller bei Zitzler (2017)

Viele Studierende und auch Kinder wissen, dass Dinge aus der Welt für den Computer abstrakt und berechenbar gemacht werden müssen. Wenige aber haben eine konkrete Vorstellung davon, wie das praktisch umzusetzen sei. An einfachen Beispielen von Bildern kann man diese Reduktionsschritte sehr konkret verfolgen. Ein Bild wird etwa nach Formen und Farben analysiert. Aus dieser Zertrümmerung mag es möglich sein, durch Anwendung von Regeln das Bild zu rekonstruieren.

Kernpraktik Nr. 4 (Abb. 1) ist die *Nutzung und Entwicklung von Abstraktion*. Abstraktion ist ein «Absehen-Von». Von besonderen konkreten Merkmalen eines Phänomens können wir absehen, wenn es auf Aspekte, Einordnungen, Klassifizierungen ankommt. Der Gewinn von Abstraktion liegt in der Verallgemeinerung, der Zugehörigkeit des Phänomens zu einer umfassenderen Klasse von Phänomenen. Im informatischen Handeln begegnen wir dem auf Schritt und Tritt.

Die *Entwicklung berechenbarer Artefakte* ist Kernpraktik Nr. 5. Die Entwicklung von Algorithmen ist die Hauptaufgabe jeder Form des Computing. In der Welt der Bilder sind das Bild-Algorithmen.

Algorithmen sind die wichtigsten Gegenstände der Informatik. Sie sind deswegen immer wichtiger Bestandteil informatischer Bildung. Im Begriff vom «algorithmischen Denken» widmen wir ihnen unsere besondere Aufmerksamkeit. Wie beim computational thinking geht es dabei um Analyse, Abstraktion und Automation, mit Fokus auf den Übergang von der Analyse zur Abstraktion und deren Formulierung als Algorithmus[4]. Mit der Wandlung des Computers von der Rechenmaschine zum «instrumentalen Medium» (Schelhowe 1997) wird die Beschäftigung mit Algorithmen zum Thema hoher gesellschaftlicher Relevanz. Was im Rahmenkonzept zur amerikanischen K–12 Schulbildung als wichtigste Praktiken angeführt wird, können wir in den Themen *Berechenbarkeit, Abstraktion, Algorithmen und Programmieren* zusammenfassen. Drei dieser Themen tauchen auch unter den «Seven Big Ideas of Computer Science» auf, mit denen wir uns kurz befassen wollen (https://csprinciples.cs.washington.edu/sevenbigideas.html).

Seven Big Ideas of Comuter Science

Im Jahre 2012 erschienen in dem Magazin *Inroads* der *Association for Computing Machinery* (ACM) sieben «großartige» Ideen, die die Informatik bestimmten (Research Committee 2011). *Inroads* ist Bindeglied zwischen der professionellen Welt des Computing und der allgemeinen, politischen und kulturellen Öffentlichkeit. Es wurde zu einer Zeit gegründet, als die algorithmische Revolution soweit die Infrastrukturen aller Gesellschaft umgewälzt hatte, dass das Stichwort von der *Digitalisierung* sich in alle öffentlichen Diskurse einnistete, wie unverstanden auch immer das geschah.

4 Auf eine Abgrenzung von *computational* und *algorithmic* thinking gehen wir an dieser Stelle nicht ein.

Was *Inroads* veröffentlichte, waren die Ergebnisse einer kleinen Forschungsgruppe (*task force*). Ihre Aufgabe war, Computer Science in die breite Öffentlichkeit zu bringen, weit über die Schranken der Disziplin hinaus. Personen von fünf US-Universitäten waren beteiligt, unterstützt von einer grossen Beratergruppe. Explizites Ziel war es, «computational thinking» breit zu verankern.

In Kurzform (für Manager) hiessen jene sieben grossen Ideen: Abstraktion, Algorithmus, Kreativität, Daten, Internet, Wirksamkeit, Programmierung. Ihre Langfassungen lesen sich im Original so:

1. Computing is a creative human activity that engenders innovation and promotes exploration.
2. Abstraction reduces information and detail to focus on concepts relevant to understanding and solving problems.
3. Data and information facilitate the creation of knowledge.
4. Algorithms are tools for developing and expressing solutions to computational problems.
5. Programming is a creative process that produces computational artifacts.
6. Digital devices, systems, and the networks that interconnect them enable and foster computational approaches to solving problems.
7. Computing enables innovation in other fields including science, social science, humanities, arts, medicine, engineering, business.

Die Behauptungen wären eine kritische Analyse wert. Wir zitieren diese US-amerikanische Position, um auf das aufmerksam zu machen, was in vielen Abwandlungen auch in gesellschaftlichen und politischen Diskursen unseres Landes um Bildung in der digital durchfurchten Welt wirksam ist. Algorithmen und Programme, Netzwerke, Daten, Information und Wissen bleiben hängen, wenn wir danach fragen, was für die Organisation von Bildungsprozessen wichtig sein mag. Was nicht explizit auftaucht, ist: Berechenbarkeit. Und: Kritik. Was erst recht nicht auftaucht, sind: Energie, Klima, Frieden, Demokratie. Oder auch reiche Kommunikation, Beziehungsfähigkeit, Empathie, Liebe, Mitmenschlichkeit. Für John Dewey, Bürger jener USA, war «Demokratie» letzten Endes einziges Ziel aller Bildung (vgl. Dewey 1916, 2000).

Interessant ist die grosse Idee, die das seit Jahrzehnten erfreulicherweise meist gemeinsam genannte Tripel von *Daten, Information und Wissen* aufgreift. Indem Daten in Kontexte eingebettet werden, kann aus ihnen Information entstehen. Indem Daten und Information in Lebenszusammenhänge eingebracht werden, wird daraus Wissen. Im weiten Feld des Computing ist das durchaus bekannt. Gehandelt aber wird selten danach. In der Schule wäre es von grösster Bedeutung. Computer haben es nur mit Daten zu tun, *nie* mit Information. Denn Daten bilden die syntaktische, Information aber bereits die semantische Ebene der Zeichen.

Gleich zu Anfang aber erscheint die *Kreativität*. In der Liste für Manager wird sie als Begriff ausdrücklich hervorgehoben, als sei sie eine Idee der Computer Science. In den Langfassungen taucht zweimal das Adjektive *creative* auf, einmal das Substantiv *creation*.

Kreatives Verhalten und kreative Fähigkeiten beginnen im Nachahmen. Es heisst, schon bei Leonardo da Vinci hätten die Studierenden im zweiten Jahr Bilder grosser Meister kopieren müssen, um den Umgang mit Farbe zu erlernen, bevor sie sich an Eigenes herantrauen durften. Kinder lernen im Nachahmen der Eltern (vgl. Bandura 1994). Später orientieren sie sich an Idolen, die im wahrsten Sinne des Wortes vorbilden. Auch in der Welt des Berechenbaren stützt sich vieles auf das Nachahmen und das Nachahnen, wenn wir aus Programm-Beispielen lernen.

Bald nachdem der Künstler Manfred Mohr sich Ende der 1960er Jahre dem Computer zugewandt hatte, machte er sich auf in den vierdimensionalen Raum und später in höhere mathematische Dimensionen. Er bewegt sich elegant in jenen Sphären, aus denen er sich Inspiration für Geschehnisse auf der zweidimensionalen Bild-Ebene holt. Er hat verstanden: Die höheren Dimensionen sind *nicht* zu visualisieren. Nur *Anlass* können sie sein, nicht Vorzeigen für Bildprozesse. Manfred Mohr ist so in seiner künstlerischen Praxis vielen voraus. Den mathematischen und berechenbaren Raum gemäss seiner Eigenheiten für neue Bildformen zu nutzen, halten wir nicht nur für kreativ, sondern für eine aufklärerische Aufgabe heute.

In der Algorithmischen Kunst tauchen zwanglos Themen wie «Berechenbarkeit», «Daten», «Abstraktion», «Algorithmen», «Programmieren», auch «Kreativität» auf. Haben wir in der Algorithmischen Kunst eine Thematik, die im Rahmen von Bildung prominent sein kann? Das Grossthema des digitalen Bildes also? Dürfen die Professionellen der Bildungsprozesse sich Bild und Kunst entgehen lassen, wenn es um Bildung aus dem Geiste der grossen aktuellen Themen geht?

Was zeichnet Algorithmische Kunst aus?

Algorithmische Kunst ist Kunst, die Algorithmen verwendet. Wir beschränken uns hier auf Bilder. Algorithmisch Künstlerisches verlangt die Entwicklung von Algorithmen, die Bilder erzeugen. Ob diese dann als «Kunst» gelten, soll uns nicht interessieren. Wie ein Algorithmus ein Bild erzeugen kann, wissen wir zunächst überhaupt nicht. Wir müssen es lernen.

Wir wussten einst auch nicht, wie man zwei Zahlen mit je 8 Dezimalstellen addiert oder multipliziert. Wir haben auswendig gelernt, wie zwei Ziffern zu addieren sind. Ziffern sind einstellige Zahlen. Für die mehrstelligen brauchten wir ein *allgemein gültiges Schema*. Für die Addition schreiben wir die zwei Zahlen auf bestimmte Weise untereinander. Wir gehen nun von rechts nach links und addieren je die zwei Ziffern, die übereinander stehen. Wir achten dabei auf den «Übertrag». Das ist etwas schwieriger zu begreifen. Aus Fehlern und mit viel Übung lernt man das Schema.

Für das Funktionieren dieses Schemas ist die Stellen-Schreibweise der Dezimalzahlen von entscheidender Bedeutung. Die Codierung der Zahlen, die Form also, in der wir sie kennen, ermöglicht es, das durchaus komplexe Verfahren auf Elementares zurückzuführen, das wir auswendig lernen. Das ist die Einbettung in die Kultur, die uns umgibt. Im Schema handeln wir ähnlich wie Maschinen.

Prinzipiell Ähnliches finden wir in algorithmischen Bildern. Wir betrachten eine kleine Sammlung von Bildern algorithmischer Herkunft. Wir machen Grundelemente und elementare Operationen in den Bildern ausfindig. Wir erwarten dabei, dass durch Anwendung der Operationen auf die Elemente Bilder ähnlicher Art entstehen. Solche Operationen sind z.B. Rasterung, Permutation, Variation, Iteration, geometrische Transformation, Rekursion, Interpolation. Werte von Parametern behandeln wir als Zufallszahlen.

Durch die Parametrisierung von Elementen wird das einzelne Bild zum Exemplar einer «Klasse». Das einzelne Bild gewinnt Sinn und Besonderheit erst in der Zugehörigkeit zu einer Klasse.

Kennzeichen algorithmischer Bilder sind (i) die präzise, regelgeleitete Anordnung von (ii) Grundelementen, die eine (iii) Klasse von Bildern beschreibt. Im *Klassencharakter* algorithmischer Bilder unterscheiden sie sich von traditionellen Bildern, die als einzelne Bestand haben (Nake und Grabowski 2005, 136; Grabowski 2007). Wir fassen zusammen:

> Das algorithmische Bild *denken* wir. Die Kunst eines Werkes algorithmischer Kunst liegt in seiner *Klassenzugehörigkeit*, nicht im Werk allein.

Generell betrachten wir artistische Werke als Zeichen. Algorithmische Werke sind *algorithmische Zeichen* (Nake 2001). Algorithmische Zeichen werden stets von zwei Agenten interpretiert: vom menschlichen Betrachter und vom maschinellen Berechner. Wir sehen das algorithmische Zeichen als verdoppeltes Bild: Es weist *Oberfläche und Unterfläche* auf.

Auf der *Oberfläche* des Bildes erscheint sichtbar ein Exemplar der Klasse, zu der es gehört. Auf der *Unterfläche* befindet sich, vom Computer berechenbar, die generative Software als Beschreibung der Klasse.

Das Bild als Oberfläche ist das uns vertraute. Das Bild als Unterfläche ist das radikal Neue, das die algorithmische Revolution in die Welt gesetzt hat. In der Unterfläche enthält das einzelne Bild seine eigene Beschreibung. Die oberflächige Erscheinung nimmt verschiedene Ausprägungen an. Sie entspringen stets der einen, gleichen Klasse.

Ein Reigen von Beispielen

Wir geben eine Auswahl von Beispielen für algorithmisches Herangehen. Sie kommen aus Lehrveranstaltungen, die Susan Grabowski durchgeführt hat. Studierende haben dabei Bilder auf deren Repertoire von Elementen und Operationen hin analysiert. Aus den Ergebnissen haben sie algorithmische Verfahren synthetisiert. In den Anwendungen treffen Ästhetik und Algorithmik aufeinander.

Repertoire und Konstruktionsregeln

Einstieg waren Übungen algorithmischen Denkens *ohne* Computer («CS-Unplugged»). Studierende nahmen zunächst vorgelegte Bilder mit Rasterstruktur auseinander: Was ist das Repertoire? Wie entsteht aus ihm das Bild? Was sind zugelassene Konstruktionen? Nicht immer lässt sich die Frage nach dem Repertoire einfach beantworten. Je mehr «gemalt» ein Bild ist, umso geringer weist es ein *diskretes* Repertoire auf. Noch problematischer kann es sein, konstruktive Operationen zu isolieren. Rasterbilder und konstruktivistische Maler machen solch eine Übung leichter.

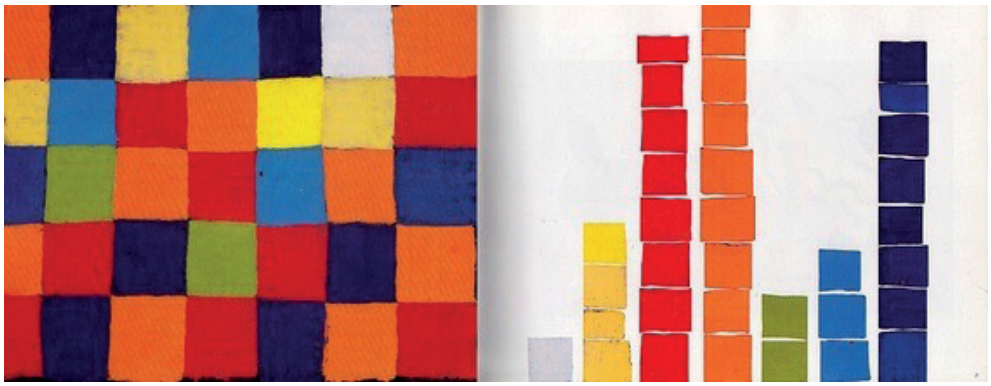


Abb. 2.: Paul Klee, Farbtafel, ca. 1930 (links). Zerlegt nach Urs Wehrli (rechts; 2002).

Urs Wehrli führt mit seinen ironisch-lehrreichen Projekten «Kunst aufräumen» (2002) auf eine wunderbare Fährte mit Nebeneffekt. Links in Abb. 2 sehen wir 5 mal 7 Farbfelder von Paul Klee, die in 7 verschiedenen Farbtönen wie zufällig verteilt sind. Pro Zeile und Spalte dürfen sie nicht öfter als dreimal vorkommen. Zwei gleichfarbige Kästchen liegen in einer Spalte nie übereinander. In einer Zeile ist es aber erlaubt, dass zwei fast gleichfarbige Felder nebeneinander stehen. Rechts sehen wir das Ergebnis von Wehrlis Aufräumen: Die Felder nach Farben in Türmchen aufgesammelt. Das komplette Repertoire.

Nebenbei tritt das Thema des *Sortierens* hier auf, wichtig für die Informatik. Ein bekanntes Sortierverfahren ist «Bubble-Sort»[5]. Wir spielen es: Sechs Personen stellen sich nebeneinander auf. Jede denkt sich eine Zahl zwischen 1 und 20, wie im folgenden Schema. Die Personen sollen sich aufsteigend nach ihren Zahlen neu aufstellen.

10	3	8	1	5	15

Von links nach rechts schauen je zwei ihre Zahlen an und entscheiden, ob sie die Plätze tauschen oder nicht. Ist die rechte Zahl kleiner, tauschen sie; ist sie grösser, tun sie nichts. Es beginnt damit, dass 10 und 3 tauschen. Auf dem Platz der 3 steht nun die 10, die mit 8 tauscht. Die 10 und 1 tauschen, dann 10 und 5, aber 10 und 15 bleiben.

3	8	1	5	10	15

Im zweiten Durchgang bleiben 3 und 8 stehen, aber 8 und 1 tauschen, 8 und 5 tauschen, der Rest bleibt. Nun sieht es so aus:

3	1	5	8	10	15

Im dritten Durchgang sind wir fertig: 3 und 1 tauschen, der Rest bleibt.

1	3	5	8	10	15

Statt Zahlen in einer Sequenz können wir in einem quadratischen Raster zufällig gewählte Rot- und Blautöne anbringen. Nun ist es aber in der Ebene mit dem Ordnen etwas heikel. Wir sortieren in einem Versuch die Zeilen nach der Rotkomponente von links nach rechts abnehmend, danach die Spalten nach der Blaukomponente von oben nach unten. Abb. 3 zeigt vier Zustände einer solchen «Sortierung».

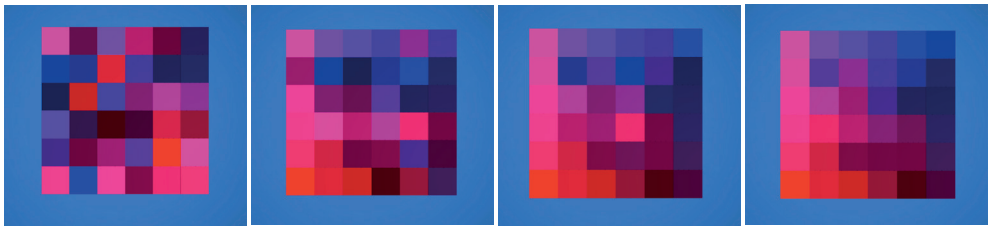


Abb. 3.: Anfangs-, Zwischen- und Endbilder einer «Sortierung» in der Ebene (Bubblesort generiert von Susan Grabowski 2018).

Die typische Eigenschaft einer Sortierung wird bei diesem Vorgehen nicht sichtbar. Der Schritt von der linearen Folge zur ebenen Fläche macht Ordnung zum Problem. Das gibt Anlass für eine neue Fragestellung. Sie zu behandeln, lädt zum Experimentieren ein in einem Feld, wo Mathematik, Algorithmik und Ästhetik sich treffen. Abb. 4 ist Ergebnis eines anderen algorithmischen Experiments, das vielen als stärker geordnet erscheinen mag.

5 Vgl. Bubble sort in Wikipedia: https://en.wikipedia.org/wiki/Bubble_sort

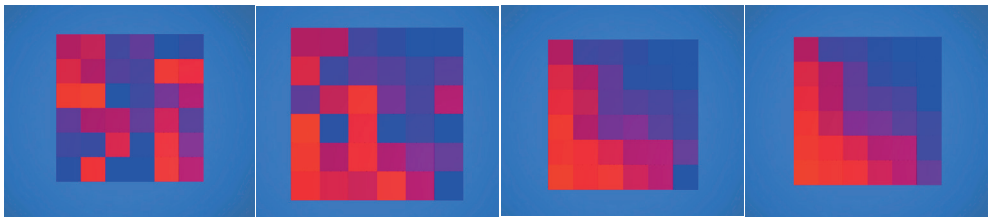


Abb. 4.: Anfangs-, Zwischen- und Endbilder einer anderen ebenen «Sortierung» (Bubblesort generiert von Susan Grabowski 2018).

Konstruktions-Regeln erfinden

Galt die erste Übung dem Aufspüren von Regeln in Werken von Künstlern, so stellen wir jetzt Regeln neu auf. Abb. 5 zeigt ein Seminarbeispiel. Studierende geben sich eigene Regeln, die Ihre Mitstudierenden entdecken sollen.

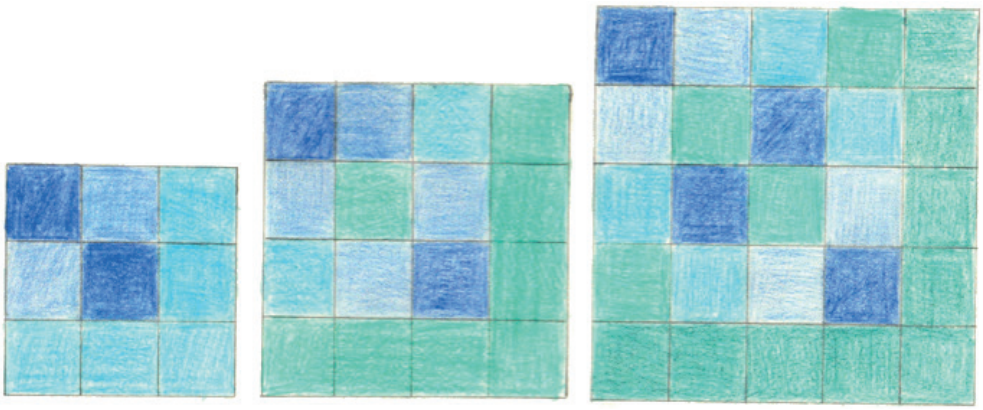


Abb. 5.: Bilder nach Regeln aufbauen. Studentinnenzeichnung 2015.

Die oberste Zeile jeder der drei Anordnungen dient als Ausgangssituation. In der zweiten Zeile wird fortlaufend springend das je zweite Kästchen der ersten Zeile gezeichnet: Beim 3 x 3-Arrangement also die Kästchen Nr. 2, 1, 3 (von links im Ring gezählt). In der dritten Zeile wird jedes dritte Kästchen der Zeile darüber gezeichnet. Nicht ganz leicht ist diese Regel zu durchschauen.

Max Bills «Ausdehnung in Gelb» (1972/73) ist Anlass für zwei Gestaltungen in Abb. 6. Links ist Bills Werk abgebildet. Die Studierenden greifen folgende Merkmale auf:

- Das Bild besteht aus einem dunkelblauen Quadrat.
- Darüber liegt mittig ein um 45 Grad gedrehtes Quadrat in dunklem Gelb. Seine Ecken berühren die Kanten des blauen Quadrates.
- Ein drittes, auch um 45 Grad gedrehtes Quadrat liegt über der blauen, aber unter der gelben Fläche. Es füllt die Hälfte der Fläche vom gelben zum blauen Quadrat aus. Seine vier sichtbaren Balken sind in Pastellgelb und Pastellblau, sich je gegenüber liegend, gefärbt.

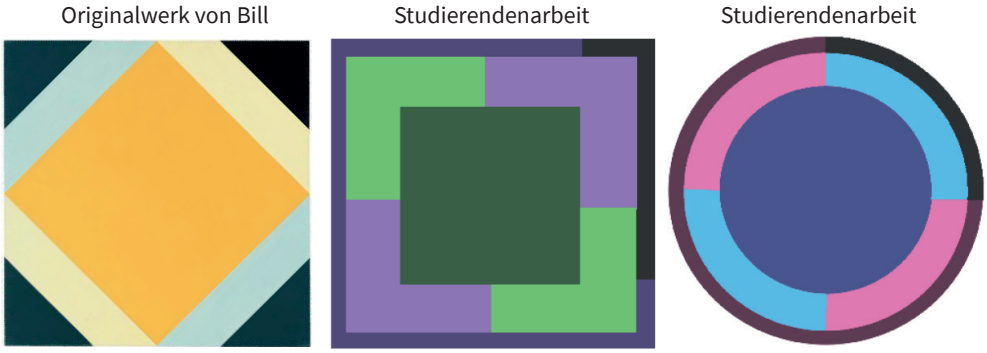


Abb. 6.: Regeln entdecken und anwenden: Max Bill «Ausdehnung in Gelb» (2009), Studierendenarbeit 2015.

Die Farbkomposition ist typisch für Bill: das Gelb in der Mitte strahlt zum dunklen Blau in den Ecken aus und erzeugt warme gelb-blaue Zwischentöne. (Bill 2009: 72) Studierende sahen das Werk als Überlagerung von drei Schichten an, die in einem 2 x 2-Raster durch vier Dreiecke angeordnet und beschnitten sind, wie Abb. 7 zeigt. Kunsthistoriker dagegen beschreiben das unmittelbar Sichtbare, wie z.B. vier rechtwinklige dunkelblaue Dreiecke in den Ecken der Komposition (ebd.). Die Aufgabe lautete. die (oder eine) Regel im Werk von Bill zu finden und diese für ein neues Bild zu nutzen, wie die Beispiele in Abb. 6 zeigen.

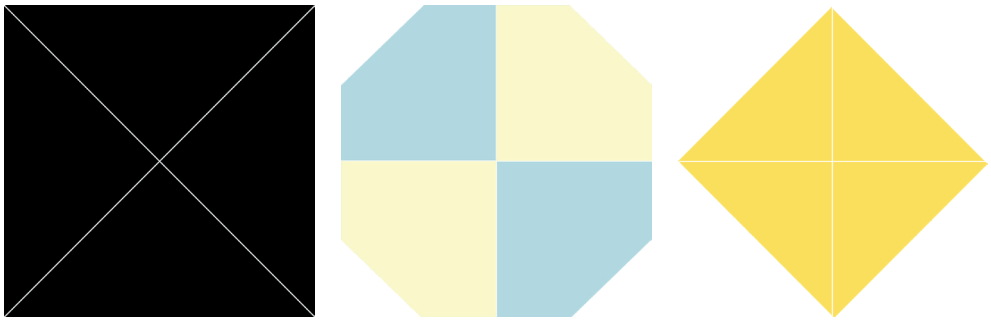


Abb. 7.: Mögliche Schichten des Originalwerkes von Max Bill, generiert von Susan Grabowski 2015.

Repertoire und Konstruktionsregeln, noch einmal

Wir hatten damit begonnen, Repertoire und Konstruktionsregeln eines Bildes zu identifizieren. Das Repertoire eines Bildes mag offen zu Tage liegen. Es mag aber auch versteckt oder gar nicht explizit im Bild vorhanden sein. Ist das Bild selbst algorithmischen Ursprungs, so kann das Bestimmen des Repertoires einfach sein. – Wir wählten das Werk von Georg Nees, «Andreaskreuz» (Abb. 8 links oben). Im Seminar stellten wir diese Aufgabe:

- Welche Regeln stecken im Werk?
- Wie können diese Regeln in einem analogen Spiel vermittelt werden?
- Wie könnte ein Programm aussehen, das solch ein Bild produzieren würde?

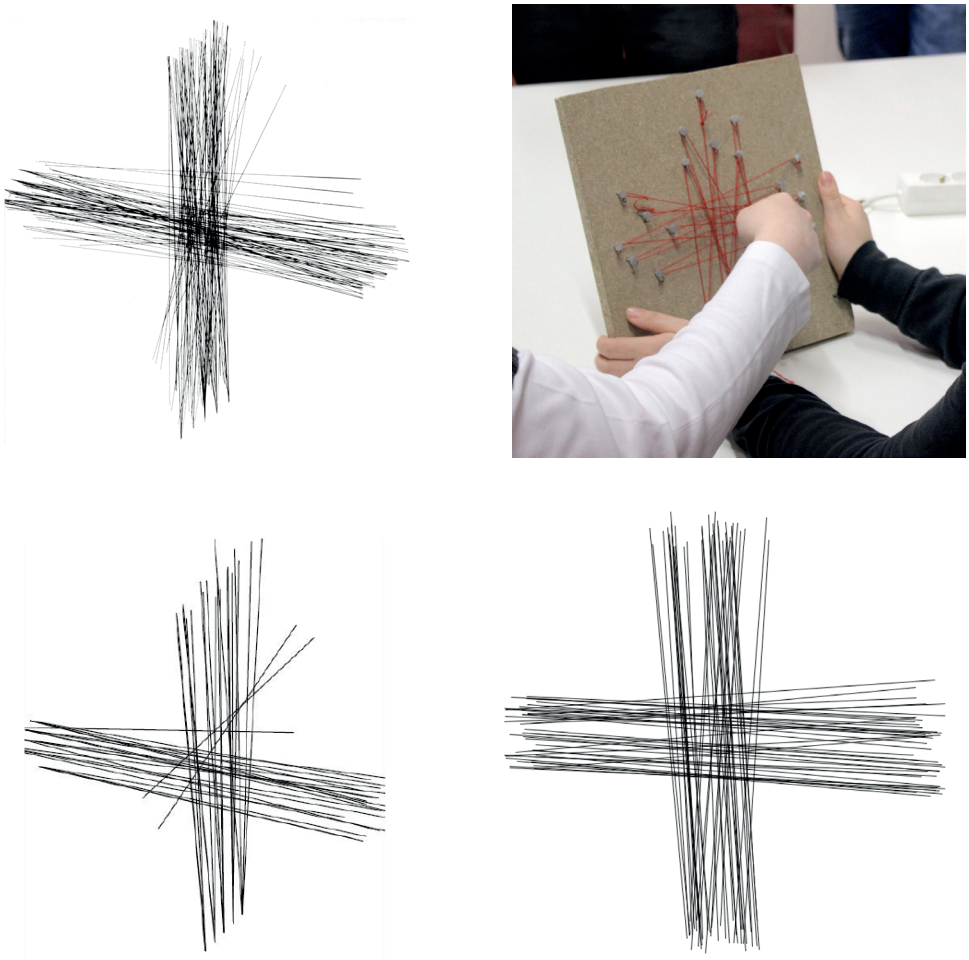


Abb. 8.: Georg Nees: Andreaskreuz (oben li.) 1965. Inspiration für analoge und digitale Paraphrasen. Mit freundlicher Genehmigung des Künstlers und der Studierenden 2015.

Studierende sind wie folgt vorgegangen:

– Einige versuchten, das Werk mit Filzstiften und Geodreieck von Hand oder mit Hilfe des Programms *Illustrator* nachzuzeichnen. Sie zeichneten horizontale und vertikale Linien. Dann zählten sie die Linien (je 50 horizontal und vertikal). Dabei bemerkten sie, dass es sich um fortlaufende Linienzüge handelt, die ihre Richtung scharf ändern (nahezu «zurück»). – Welches Repertoire entdeckten sie? *Zwei Polygonzüge mit je 50 Kanten*! Was sind die Regeln? Die Kantenlängen schwanken gering um einen Wert. Die Richtungen gehen mit geringer Abweichung nach oben & unten bzw. links & rechts.

– Ihren Programmier-Versuch (unten links) beschreibt eine Gruppe so: «Das Werk besteht aus horizontalen und senkrechten Geraden. Bei den senkrechten Geraden haben wir 5 Ausgangspunkte definiert, von denen die Geraden innerhalb eines Winkels nach oben verlaufen. Bei der Horizontalen verlaufen die Geraden ebenfalls von fünf Ausgangspunkten, wovon ein Ausgangspunkt ausserhalb des sichtbaren Bereiches liegt und die Geraden nach unten geneigt sind. Die scheinbar aus dem Raster fallenden Geraden haben wir separat definiert.» – Algorithmisches Denken muss präziser sein.
– Die Beschreibung zum Versuch unten rechts lautet: «Das Werk besteht aus Senk- und Waagrechten schwarzen Linien vor einem weißen Hintergrund. Ausgangspunkte der Linienbündel befinden sich in der Mitte des Bildes. Auffällig ist, dass alle Linien individuelle Längen besitzen, die sich nur minimal unterscheiden. Algorithmisch betrachtet, dominieren in dem Werk die Wiederholung und der Zufall. Pro Ausrichtung wiederholt sich der Vorgang ca. 50mal. Start-, Endpunkt und Länge der Linien verändern sich innerhalb zufällig bestimmter Intervalle.» – Die Annäherung an algorithmisches Denken ist besser gelungen. Ein Spaziergang wird es nicht werden.

Ein Programm für ein Bild

Liegen Repertoire-Elemente und elementare Operationen vor, so können wir sie zu einem Programm für neue Bilder verknüpfen. Diese Bilder sind *gedacht*. Sie existieren als Instanzen einer Klasse. Wir sehen sie nach wie vor als einzelne Stücke. Doch ästhetisch interessant werden sie durch die Klasse, die sie repräsentieren. Sie entspringen dem Gedanken mehr als der Hand.

Wir begegnen in den algorithmischen Bildern einem radikalen Bruch, der diese Art von Kunst auszeichnet. Es ist der Schritt von der materiellen zur zunächst immer semiotischen Existenz des Bildes.

Wir legen den Studierenden ein Bild von Vera Molnar vor (Abb. 9). Dem Werk liegt ein 14 x 14-Raster zugrunde. In den Rasterfeldern befinden sich schwarze Quadrate gleicher Grösse an leicht variierenden Orten. Den Reiz des Bildes macht jedoch am stärksten die Veränderung des Schwarz zu Silbertönen aus: als ob von oben Licht ins Bild fiele. Die Frage an die Studierenden lautete erneut: Welche Regeln stecken im Werk? Welcher Algorithmus könnte eine Klasse solcher Bilder erzeugen? Das Bild sollte jetzt in Processing re-codiert werden.

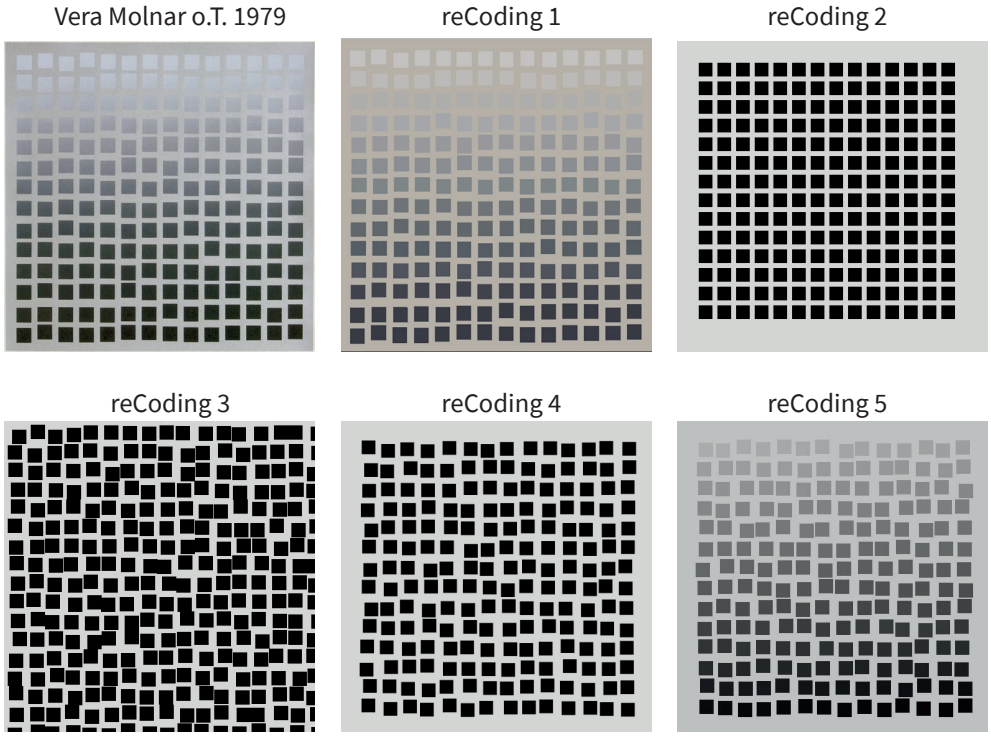


Abb. 9.: Vera Molnar: 196 carrès 1979 sowie 5 Versuche einer reCodierung. Mit freundlicher Genehmigung der Künstlerin und der Studierenden 2017.

Das Ergebnis reCoding 1 kommt aus einer Programmierung Kästchen für Kästchen in Hunderten von Codezeilen. Der Schritt zur Parametrisierung und damit zur Klasse wird vermieden. Nr. 2 dagegen ist als das andere Extrem radikal vereinfacht auf das regelmässige Schema des Rasters, also die zweidimensional einfach zu beschreibende Folge der Quadrate ohne jede Variation. Vier Zeilen Code reichten dafür. Erst bei den nächsten Schritten (reCoding 3 bis 5) wird das Werk spielerisch algorithmisch konstruiert. Im regelmässigen Raster wird nun an zufällig gewählter Stelle, innerhalb des Rasterfeldes bleibend, ein schwarzes Quadrat platziert. Diese Quadrate könnten auch in der Grösse noch variieren.[6]

Wir haben die Studierenden erst ihren eigenen Weg gehen lassen. Eine detailgetreue Rekonstruktion ist zunächst ein möglicher Weg einer Rekonstruktion. Ein freierer algorithmischer Ansatz ist ein anderer. Er erscheint den meisten als einfacher und kürzer, indem er das Bild als Raster von Objekten einer Klasse behandelt. Dieser Schritt vom regelmässig algorithmisch erzeugten Raster zum pro Zelle einzeln erzeugten Zufall scheint bei den Studierenden für die Programmierung viel bewirkt zu haben.

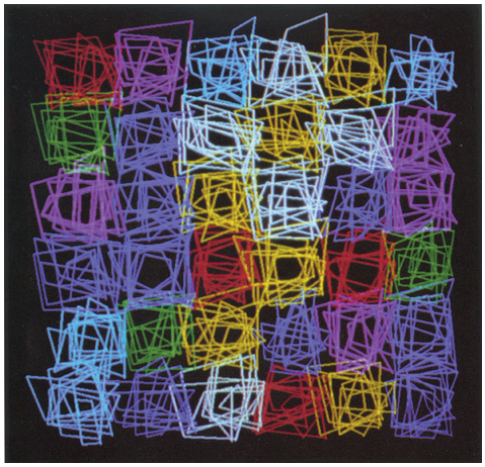
6 Die Studierenden sollten alle ihre Schritte beim Programmieren aufzeichnen, schriftlich und als Programmvarianten.

Der Schritt in die Interaktion

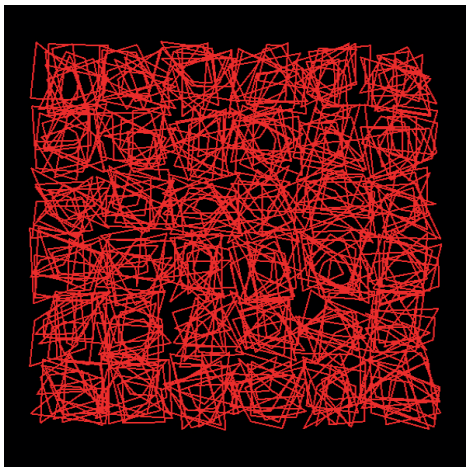
Mit der interaktiven Verwendung von Algorithmen (in Form von Programmen) lassen wir die pure Berechenbarkeit hinter uns. Betrachten wir einen Menschen zusammen mit einer Software als *ein* System, so ist die Funktion, die dieses System als Ganzes realisiert, nicht unbedingt berechenbar. Das wird nur dadurch möglich, dass der Mensch Teil des Geschehens wird. Dies zu begreifen, also nach dem Verdikt der Berechenbarkeit deren Überwindung selbst handzuhaben, muss Bestandteil des ästhetisch-künstlerischen Umgangs mit Algorithmen werden.

Wir nehmen ein weiteres Werk von Vera Molnar zum Anlass für eine Interaktion, die keine ist: Sie ist Eingriff in das Programm selbst (Abb. 10).

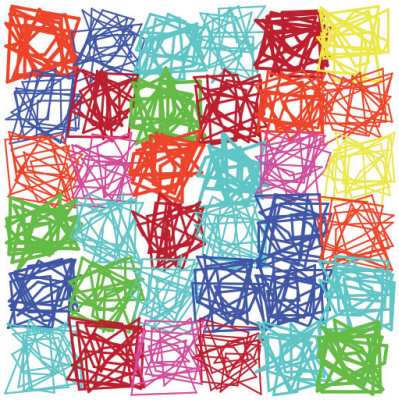
Vera Molnar 1986



Variante 1



Variante 2



Variante 3

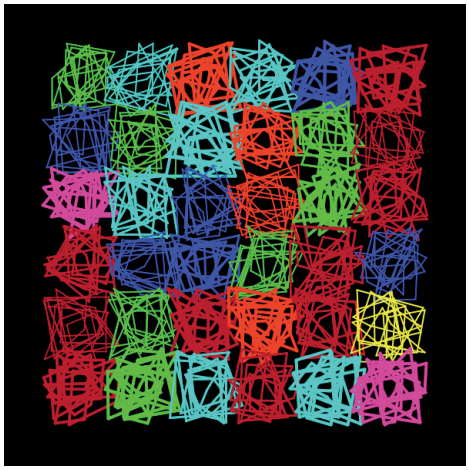


Abb. 10.: Vera Molnar: structures de quadrilatères 1986, mit drei Abwandlungen. Mit freundlicher Genehmigung der Künstlerin und der Studierenden 2015.

Das Werk «structures de quadrilatères» besteht aus einem 6 x 6 Raster. Jedes Raster-Quadrat ist mit geraden Linien gefüllt. Jedes Liniengebilde besteht aus einer Kombination von verschieden grossen Vierecken. Pro Quadrat gibt es bis zu 10 Linienvierecke gleicher Farbe und gleicher Strichstärke. Farben und Strichstärken variieren von Quadrat zu Quadrat. Insgesamt werden 7 verschiedene Farben und 2 verschiedene Strichstärken verwendet. Linienquadrate einzelner Rasterfelder überschneiden sich teilweise mit Linienquadraten benachbarter Rasterfelder. Das Programm sorgt für das Zeichnen solcher Konfigurationen.

Das Bild hat eine Hintergrundfarbe. Vor ihr 36 Quadratzellen. Jede erhält eine Anzahl von Linienquadraten einer Farbe, evtl. variierender Strichstärke, um zufällige Winkel verdreht, ein wenig aus dem Zentrum verschoben.

Der interaktive Eingriff geschieht hier nun nicht, wie üblich, an der Oberfläche des Bildes über Sensoren, sondern durch vorangehenden Eingriff in die Unterfläche, durch Manipulation des Programms selbst, wenn auch nur durch Setzen der Parameter für die genannten Zufallswahlen.

Beispiele für drei Themen

Wir geben in diesem Abschnitt einfache Beispiele aus Lehrveranstaltungen zur Behandlung der Themen Berechenbarkeit, Abstraktion und Algorithmen.

Berechenbarkeit

Schon früh in der Entwicklung algorithmischer Kunst gab es Versuche, Bilder von Künstlern nachzuahmen. A. Michael Noll hat 1966 zwei solche Versuche mit Bildern von Piet Mondrian und Bridget Riley angestellt. Abb. 11 zeigt das oft abgebildete Ergebnis im Fall von Piet Mondrian.

Abb. 11.: «Berechenbarkeit». A. M. Noll, Simulation 1965 (li.) eines Gemäldes von Piet Mondrian, Pier and Ocean, 1917 (re.). Mit freundlicher Genehmigung des Künstlers A. M. Noll.

Um Berechenbarkeit zu erzielen, musste Noll zunächst die rundliche Form des Mondrianschen Gemäldes fassen. Er wählte den Kreis. Der Grund Mondrians für die mehr oder minder gerade Linie am oberen Rand ist der Horizont, zu dem hinaus Mondrian, am Meer stehend, blickt.

Mondrians Vorlage weist in einem diffus bestimmbaren Bereich, der sich in das Bild hinein nach unten drängt, eine Verdünnung der kurzen horizontalen und vertikalen Balken auf, die er hier verwendet. Noll fällt das auf, er nimmt es in die Simulation auf in Form einer Parabel (berechenbar!).

Nolls Programm zeigt zu einem frühen Zeitpunkt, welche Gewalt einer Vorlage angetan werden muss, um sie zu simulieren.

Abstraktion

Zu Beginn des 20. Jahrhunderts war der Schritt zur nicht-figürlichen (abstrakten) Malerei das grosse kunsthistorische Thema. Abb. 12 zeigt ein einfaches Beispiel.

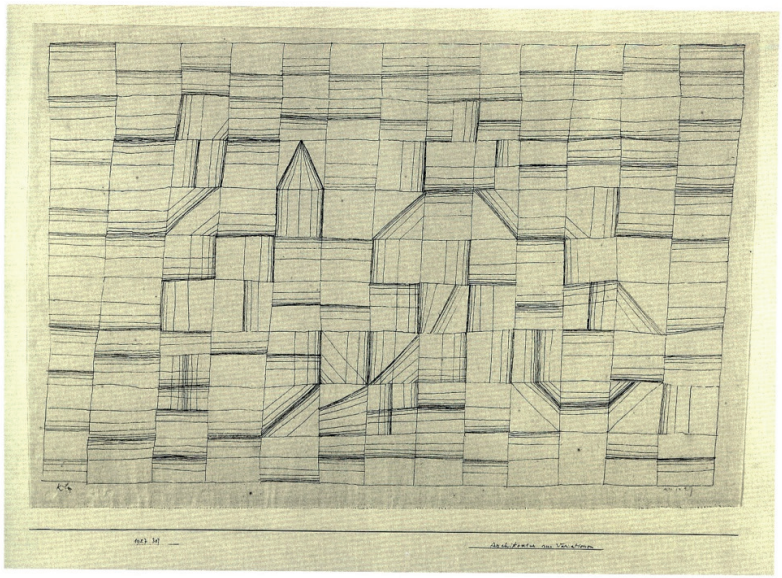


Abb. 12.: «Abstraktion». Paul Klee: Architektur aus Variationen (Klee 1927).

Wie mag Klee bei dieser Zeichnung vorgegangen sein? Wir stellen uns vor, wir stehen auf einem Berg und schauen auf eine Stadt hinunter. Wir abstrahieren von der realen Welt auf eine 2-dimensionale Skizze. Aus einem Haus werden vielleicht 6 Linien, die als Quadrat mit Spitzdach angeordnet sind. Strassen bestehen aus 2 parallel angeordneten Linien. Jeder Gegenstand der Szene wird im Abstrakten auf ein Liniengebilde reduziert, welches zudem in einem Raster angeordnet ist. Eine gute Gelegenheit für ein Computerprogramm.

Algorithmen

Abstraktion als generelles Verhältnis zur Welt macht den Kern all dessen aus, was die Algorithmische Revolution bedeutet. Das ist jedoch sehr abstrakt gesagt. Denn die Art von Abstraktion, die unsere Zeit prägt und die alles umstürzt, was uns lieb und fest war, zielt auf eine gewaltige Konkretion: Die Realisierung in Maschinerie all dessen, was durch Abstraktion als Algorithmus und Modell berechenbar herausgezogen wird aus der dampfenden, brodelnden Welt. An Abstraktion als solcher ist die Welt der Berechenbarkeit nur marginal interessiert. Als Mittel zum Zweck jedoch ist sie das sehr wohl. Die Abstraktion muss im Algorithmischen enden. Das ist die neue Konkretion! Und die berechenbaren Modelle gilt es letzten Endes in der flüssig-flüchtigen Maschinerie semiotischer Natur neu zu erbauen: in Software.

Zentraler Gegenstand für Bildungsprozesse sind daher Algorithmen. Es kann nicht ausbleiben, dass auf sie hin (Abstraktion, Mathematik) und von ihnen weg (Programmierung, Ethik) alles sich konzentriert.

Wir geben ein maximal reduziertes Beispiel, vielleicht das, was jene versucht haben mögen, die in den 1960er Jahren mit algorithmischer Kunst begonnen haben. Was sollten sie auch sonst als erstes tun, als die Maschinen einfache gerade Linien zeichnen zu lassen? So soll die Aufgabe lauten:

> 100 gerade Linienstücke sollen auf der Bildfläche von 400 x 400 Pixeln[7] in zufälliger Richtung, an zufälligem Ort und von zufälliger Länge gezeichnet werden. Auch die Farbe eines Linienstücks soll zufällig sein. Die Zeichnung soll im Bild einen Rand von 40 Pixel Breite frei lassen.

Abb. 13 zeigt ein Beispiel, das vom hierzu entworfenen Programm der Abb. 14 berechnet wurde.

Der Reiz jeder Arbeit mit Software liegt darin, dass die Maschine selbst, die ein Programm ausführt, recht gut Auskunft darüber gibt, was hier wie funktioniert. Der gute Rat: Entwerft als erstes, skizziert, verschafft Euch Klarheit über das, was Ihr erreichen wollt, schreibt es auf, macht alles explizit, noch vor dem Programm-Code. Dieser Rat jedoch verhallt fast ungehört. Niemand kümmert sich um ihn. *The taste of the algorithm is in running the program*.

7 Von Pixeln konnte damals noch nicht gesprochen werden, das ist erst heute möglich.

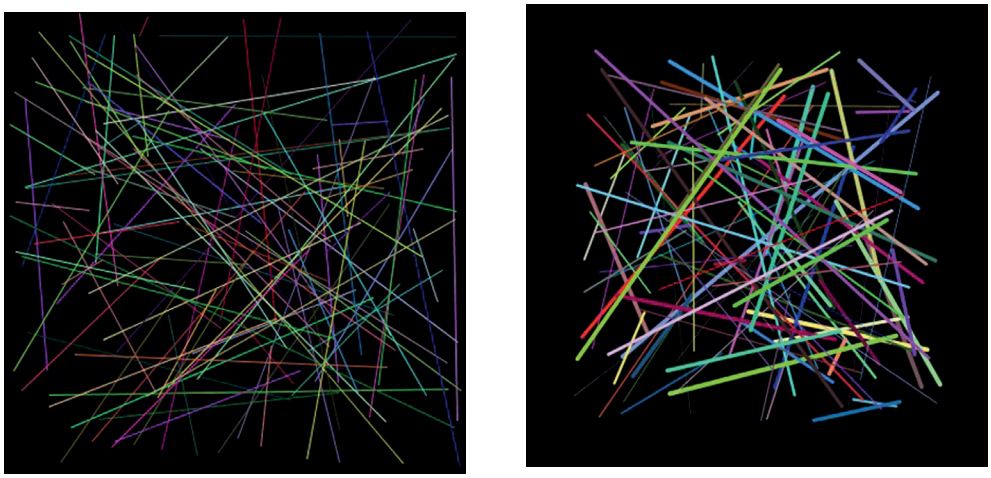


Abb. 13.: Berechenbare Artefakte entwerfen, entwickeln, testen, verfeinern. Zufällige gerade Linienstücke, generiert von Susan Grabowski 2018.

Die pädagogischen Ratschläge sind ja richtig. Sie werden ihre Bedeutung und Wirkung dann zeigen, wenn das Programm wächst, wenn der Algorithmus mehr und mehr an Abstraktionen verlangt. Sie schlagen sich nieder in der angesprochenen Klassenbildung! Und sie wiederum wird handfest in Parametern. Ein gut entworfenes, mit ästhetischer Kompetenz entwickeltes Programm beginnt mit einer Liste all dessen, was hier auch anders gewählt werden könnte, und das sind die Werte der Parameter. Auch das soll hier nicht weiter ausgeführt werden. Geringe Hinweise zu unserem simplen Fall seien genug (vgl. hierzu den Code von Abb. 14).

line_c

```
2
3  int anzahl=100;       // zeichne 100 linien
4
5  void setup() {        // grundeinstellungen
6
7     size(400,400);     // zeichenfläche von 400 x 400 px
8     background(0);     // hintergrundfarbe schwarz
9     smooth();         // kantenglättung der linien einschalten
10    noLoop();         // keine Wiederholung - nur 1 mal durchlaufen
11
12 }
13
14 void draw () {
15
16   for (int i=0; i<anzahl; i++)// schleife für 100 linien
17   {
18       // zufällige strichfarbe
19       stroke(random (0,255), random (0,255), random (0,255));
20       // zufällige strichstärke
21       strokeWeight(random (0, 4));
22       //zufälliges Geradenstück innerhalb des Rand von marg Pixel Breite
23     line(random (marg, width-marg), random (marg, height-marg),
24          random (marg, width-marg), random (marg, height-marg));
25     }
26 }
```

Abb. 14.: Programm-Code für eine simple Grafik. Notiert in Processing, generiert von Susan Grabowski 2018.

Die Bildgrösse ist konstant – ein Mangel! Das RGB-System für die Farbwahl – das ist ästhetisch unklug. Die zufälligen Wahlen völlig unkontrolliert im Programm zu platzieren: das wird den Umgang mit dem Programm erschweren. Alles muss einfache Kontrollmöglichkeiten erlauben.

Einen Hinweis auf die Struktur (im Programmtext) erlauben wir uns: die mit der Zeile

```
for (int i=0; i<anzahl; i++)
```

beginnende Struktur. Diese Struktur ist mit die wichtigste in einer algorithmischen Beschreibung. Sie bezeichnet eine Iteration. Das ist das Mittel, um auf endliche Weise eine potenziell unendliche Menge von Prozessen zu beschreiben.

Das Beispiel ist in der Programmiersprache *Processing* abgefasst, die sich erfolgreich an Künstler und Designer wendet. (Reas und Fry 2016) Algorithmen müssen, wie Programme, explizit notiert werden. Sie existieren nicht im Kopf allein. Sie müssen mitgeteilt werden können. Was aber den Algorithmus vor dem Programm auszeichnet, ist, dass er an keine bestimmte Notation gebunden ist. Das Insistieren auf dem Algorithmus vor dem Programm bedeutet, sich beim Entwurf nicht den Zwängen der Programmiersprache zu unterwerfen. (Hierin liegt vielleicht der Unterschied zwischen *Algorithmic* und *Computational Thinking*.) Wir wollen *ent*werfen, nicht unterworfen werden! Doch das ist vielleicht altmodisch gedacht angesichts der mächtigen Programmierhilfen heute.

Fazit

Um algorithmische Kunst ging es uns in diesem Essay. Nicht aus der Sicht von Künstlern, sondern von Lehrenden und Lernenden. Was ist davon zu halten, dass die notwendig erscheinenden Anstrengungen von Schulbildung im Dunstkreis des Digitalen als Gegenstand die Welt der Kunst und nicht die der Arbeit, nicht die des Krieges oder anderer Verwerfungen der Gesellschaft wählen? Gewiss *auch* diese anderen Bereiche. Denn wie soll eine Bildung in Zeiten der Digitalisierung ohne *Big Data*, ohne *Virtual Reality* oder *Artificial Intelligence* gelingen können? Ist aber Bildung nicht gerade dadurch zu gewinnen, dass nicht das pragmatisch Nächstliegende aufgegriffen wird, sondern das Fernliegende, das Überraschende, das vielleicht auch Fremde?

Deswegen unser Griff zur Algorithmischen, zur Künstlichen Kunst (ein Wort von Max Bense). Dem algorithmischen Denken, dem grossen Bruder des *computational thinking*, begegnen wir hier allemal.

Worin mögen sie sich unterscheiden? Letztlich in der Feinheit nur, dass das Algorithmische in seiner expliziten Form nicht nach dem Code verlangt, die jeder Programmierung als Fluch anhängt. Das Programmierte ist im Formalen gezwungen. Das Algorithmische atmet noch frei. Es ist dem Gedanken näher als der Form, in die es gepresst werden muss, wenn der Computer rechnen soll.

Die Form des Algorithmischen kann andeuten, dass Algorithmen und Ästhetik nahezu das Gleiche seien, wie es in so vielen spektakulären Fällen digitaler Transformationen zu sein scheint. Unsere Zeit hat keinen Überschuss an Kritik der Technologie, insbesondere semiotischer Maschinerie. Im Gegenteil, eher beobachten wir eine Akzeptanz der digitalen Medien, die erschrecken kann. Erst bei Skandalen der Raubritter von Facebook und Freunden kratzt sich der eine oder die andere am Kopf. Das Ungewöhnliche aufzugreifen und in seiner Ungewöhnlichkeit kritisch zu durchleuchten, wenn der algorithmische Furor zuschlägt, das lohnt Bildung als Anstrengung.

Literatur

Bandura, Albert. 1994. *Lernen am Modell*. Ansätze zu einer sozial-kognitiven Lerntheorie. Stuttgart: Klett-Verlag.

Bill, Max. 2009. «Ausdehnung von Gelb». In *Konkret: die Sammlung Heinz und Anette Teufel im Kunstmuseum Stuttgart*, herausgegeben von Simone Schimpf und Kunstmuseum Stuttgart. Bd. 1. Bestandskatalog / Kunstmuseum Stuttgart, S. 75. Ostfildern: Hatje Cantz-Verlag.

Dewey, John. 2000. *Demokratie und Erziehung. Eine Einleitung in die philosophische Pädagogik.* Hrsg. von Jürgen Oelkers. Weinheim, Basel: Beltz (Amerikanische Originalausgabe 1916).

Grabowski, Susanne. 2006. «ZeichenRaum: digitale Medien in Studienumgebungen am Beispiel der Computerkunst». Dissertation, Bremen: Universität Bremen. https://nbn-resolving.org/urn:nbn:de:gbv:46-diss000108784.

K–12 Computer Science Framework. 2016. https://k12cs.org/wp-content/uploads/2016/09/K%E2%80%9312-Computer-Science-Framework.pdf.

Kant, Immanuel. 1966. «Beantwortung der Frage: Was ist Aufklärung?» *Werke*, Bd. VI, 51-61. Darmstadt: Wiss. Buchgesellschaft.

Klee, Paul. 1992. «Architektur aus Variationen (1927)». In *Computer und Kunst. Programmierte Gestaltung: Wurzeln und Tendenzen neuer Ästhetiken*, herausgegeben von Erwin Steller. S. 328. Mannheim, Leipzig, Wien, Zürich: BI Wissenschaftsverlag.

Molnar, Vera. *196 carrés*. 1979. Fotografie mit Genehmigung der Kunsthalle Bremen 2016.

Molnar, Vera. 2006. «Structures de quadrilatères (Viereckstrukturen), 1986». In *monotonie, symetrie, surprise*, herausgegeben von Wulf Herzogenrath, und Barbara Nierhoff. Ausstellungskatalog Kunsthalle Bremen.

Nake, Frieder, und Susanne Grabowski. 2001. «Human–Computer Interaction Viewed as Pseudo-Communication». *Knowledge-Based Systems* 14 (8): 441–47. https://doi.org/10.1016/S0950-7051(01)00140-X.

Nake, Frieder. 2001. «Das algorithmische Zeichen». In *Informatik 2001: Wirtschaft und Wissenschaft in der Network Economy – Visionen und Wirklichkeit, Tagungsband der GI/OCG-Jahrestagung*, herausgegeben von Kurt Bauknecht, Wilfried Brauer, und Thomas A. Mück, 2:736–42. Schriftenreihe der Österreichischen Computer-Gesellschaft 157. Konstanz: UVK.

Nake, Frieder, und Susanne Grabowski. 2005. «Zwei Weisen, das Computerbild zu betrachten». In *HyperKult II. Zur Ortsbestimmung analoger und digitaler Medien*, herausgegeben von Martin Warnke, Wolfgang Coy, und Georg Christoph Tholen. Bielefeld: transcript Verlag. https://doi.org/10.14361/9783839402740-005.

Nake, Frieder. 2009. «The Semiotic Engine: Notes on the History of Algorithmic Images in Europe». *Art Journal* 68 (1): 76–89. https://doi.org/10.1080/00043249.2009.10791337.

Nees, Georg. 1965. «Andreaskreuz». In *1965. rot 19. Computer Grafik*, herausgegeben von Max Bense und Elisabeth Walter, S. 9. Stuttgart. Druck Hansjörg Mayer.

Noll, A. Michael. 1966. «Human or machine: A subjective comparison of Piet Mondrian's ‹Composition with Lines› (1917) and a computer-generated picture». *The Psychological Record*, vol. 16 (1): 1-10. https://doi.org/10.1007/BF03393635.

Mondrian, Piet. 1992. «Pier and Ocean. Komposition mit Linien (1917)». *Computer und Kunst. Programmierte Gestaltung: Wurzeln und Tendenzen neuer Ästhetiken*, herausgegeben von Erwin Steller. S. 325 f. Mannheim, Leipzig, Wien, Zürich: BI Wissenschaftsverlag.

Reas, Casey, und Ben Fry. 2016. *Processing. A programming handbook for visual designers and artists*. Cambridge, MA: MIT Press. 2nd ed.

Repenning, Alexander. 1991. «Creating user interfaces with Agentsheets». In *[Proceedings] 1991 Symposium on Applied Computing*, 190–96. Kansas City, MO, USA: IEEE Comput. Soc. Press. https://doi.org/10.1109/SOAC.1991.143873.

Repenning, Alexander. 1993. «Agentsheets: A Tool for Building Domain-Oriented Dynamic, Visual Environments». Dissertation, Boulder, CO: University of Colorado. https://www.cs.colorado.edu/~ralex/papers/PDF/Repenning-PhD.pdf.

Repenning, Alexander. 2015. *Computational Thinking in der Lehrerbildung. Zürich: Schriftenreihe der Hasler Stiftung*. http://www.fit-in-it.ch/sites/default/files/downloads/schrift_repenning-1411-gzd_deutsch_0.pdf.

Research Committee. 2011. *Seven Big Ideas of Computer Science.* Seattle: University of Washington https://csprinciples.cs.washington.edu/sevenbigideas.html

Schelhowe, Heidi. 1997. *Das Medium aus der Maschine*. Frankfurt a.M.: Campus.

Steller, Erwin. 1992. *Computer und Kunst. Programmierte Gestaltung: Wurzeln und Tendenzen neuer Ästhetiken*. S. 328. Mannheim, Leipzig, Wien, Zürich: BI Wissenschaftsverlag.

Turing, Alan Mathison. 1937. «On Computable Numbers, with an Application to the Entscheidungsproblem». Proceedings of the London Mathematical Society s2-42 (1): 230–65. https://doi.org/10.1112/plms/s2-42.1.230.

von Hentig, Hartmut. 1985. *Die Menschen stärken, die Sachen klären. Ein Plädoyer für die Wiederherstellung der Aufklärung*. Stuttgart: Reclam.

Wehrli, Ursus. 2002. *Kunst aufräumen*. S. 4–5. Zürich: Kein & Aber.

Weizenbaum, Joseph. 1978: *Die Macht der Computer und die Ohnmacht der Vernunft*. Frankfurt a.M.: Suhrkamp (Originalausgabe 1976).

Wilkens, Ulrike. 2000. *Das allmählichen Verschwinden der informationstechnischen Grundbildung. Zum Verhältnis von Informatik und Allgemeinbildung*. Herzogenrath: Shaker Verlag.

Wing, Jeannette M. 2006. «Computational Thinking». *Communications of the ACM* 49 (3): 33–35. https://doi.org/10.1145/1118178.1118215.

Zitzler, Eckart. 2017. *Dem Computer ins Hirn geschaut. Informatik entdecken, verstehen und querdenken*. Berlin, Heidelberg: Springer. https://doi.org/10.1007/978-3-662-53666-7.

Abbildungen

Abb. 1.: Kernpraktiken des Computing (K–12 2016, 68).

Abb. 2.: Paul Klee, Farbtafel, ca. 1930 (links). Zerlegt nach Urs Wehrli (rechts; 2002).

Abb. 3.: Anfangs-, Zwischen- und Endbilder einer «Sortierung» in der Ebene (Bubblesort generiert von Susan Grabowski 2018).

Abb. 4.: Anfangs-, Zwischen- und Endbilder einer anderen ebenen «Sortierung» (Bubblesort generiert von Susan Grabowski 2018).

Abb. 5.: Bilder nach Regeln aufbauen. Studentinnenzeichnung 2015.

Abb. 6.: Regeln entdecken und anwenden: Max Bill «Ausdehnung in Gelb» (2009), Studierendenarbeit 2015.

Abb. 7.: Mögliche Schichten des Originalwerkes von Max Bill, generiert von Susan Grabowski 2015.

Abb. 8.: Georg Nees: Andreaskreuz (oben li.) 1965. Inspiration für analoge und digitale Paraphrasen. Mit freundlicher Genehmigung des Künstlers und der Studierenden 2015.

Abb. 9.: Vera Molnar: 196 carrès 1979 sowie 5 Versuche einer reCodierung. Mit freundlicher Genehmigung der Künstlerin und der Studierenden 2017.

Abb. 10.: Vera Molnar: structures de quadrilatères 1986, mit drei Abwandlungen. Mit freundlicher Genehmigung der Künstlerin und der Studierenden 2015.

Abb. 11.: «Berechenbarkeit». A. M. Noll, Simulation 1965 (li.) eines Gemäldes von Piet Mondrian, Pier and Ocean, 1917 (re.). Mit freundlicher Genehmigung des Künstlers A. M. Noll.

Abb. 12.: «Abstraktion». Paul Klee: Architektur aus Variationen (Klee 1927).

Abb. 13.: Berechenbare Artefakte entwerfen, entwickeln, testen, verfeinern. Zufällige gerade Linienstücke, generiert von Susan Grabowski 2018.

Abb. 14.: Programm-Code für eine simple Grafik. Notiert in Processing, generiert von Susan Grabowski 2018.