Journal of
Design Studio
v:7 n:1 July 2025

Algorithmic Art Praxis: A Framework for Contextualized Programming Education

Alp Tugan
Ozyegin University, Faculty of Architecture and Design, Istanbul, Türkiye (Corresponding author)
Ayse Hazar Koksal
Ozyegin University, Faculty of Architecture and Design, Istanbul, Türkiye.

Research Article / Received: January 4th 2025, Revised: Accepted: January 22nd 2025
Refer: Tugan, A., Koksal, A.H., (2025), Algorithmic Art Praxis: A Framework for Contextualized Programming Education, Journal of Design Studio, V.7 N.1, pp 5-29.
A. Tugan ORCID 0000-0003-3673-8675 (alp.tugan@ozyegin.edu.tr), A. H. Koksal ORCID: 0000-0001-6491-589X (ayse.koksal@ozyegin.edu.tr)
DOI: 10.46474/jds.1613349 https://doi.org/10.46474/jds.1613349

Abstract: The amalgamation of computational thinking (CT) with contextualized instruction provides a sturdy framework for enriching programming education, especially for novice learners in design-centric higher education programs, where visual and experiential learning modalities prevail. CT, recognized as a critical methodology for solving complex programming challenges, underpins the development of the Algorithmic Art Praxis (ALAP) Categories—a structured toolkit designed to bridge abstract computational concepts with tangible, art-based applications. This research utilizes a rigorously curated online database of algorithmic artworks as a primary source for content analysis and pedagogical investigation. Over 2,000 algorithmic artworks from secondary sources were subjected to a rigorous, iterative review process, narrowing the collection to 695 deeply analyzed samples that inform the database's foundational content. Through this analytical perspective, 18 distinct ALAP Categories were identified, each mirroring fundamental programming principles as exemplified in algorithmic art. These categories establish a structured taxonomy that harmonizes computational thinking activities with contextualized programming education, thereby providing a customized approach to addressing the distinct cognitive and creative requirements of design students. The ALAP toolkit, consisting of the 18 categories, a succinct reference guide, and the curated database, serves as a versatile resource for educators, researchers, and students. Through the integration of computational thinking with algorithmic art, it facilitates the cultivation of programming proficiency in visually oriented learners while promoting engagement through relevance and creativity. This framework underscores the potential of contextualized learning to transform abstract programming concepts into accessible, meaningful educational experiences.

Keywords: Computational thinking, Algorithmic art, Programming, Contextualized education, Problem-solving, Visual learning

1. Introduction

Computers have become an integral part of our daily lives across a range of disciplines, including engineering, architecture, design, visual arts and music, over the past two decades. In addition to proficiency in third-party software tools such as Word, Excel and Photoshop, the ability to read and write computer programs has emerged as a highly sought-after skill (Shein, 2014; Romero et al., 2017). While governments encourage individuals to gain computer literacy, large companies, managers, and employers have started to prefer employees with programming knowledge in their job applications, regardless of their actual requirements (Guzidal, 2009). All these developments have caused computing education to become a skill not only for computer science (CS) majors but also for design students outside of this field.

Regrettably, programming courses have a lengthy track record of experiencing high dropout rates, failures, and inducing stress (Bryant et al., 2011). Research indicates various reasons for disengagement at programming courses independent from student majors. One of them is the prejudice against coding among students causes declining attendance to computing classes (Allwood, 1986; Winslow, 1996; Robins et al., 2003; Ring et al., 2008; Yardi & Bruckman, 2007). According to Yardi and Bruckman, younger generations perceive computer programming as tedious. Even if most teenagers use technology daily, using textual commands does not attract their interest. Although many modern programming language semantics are in English, students need help comprehend programming tools' linguistic grammar. Also, the abstract nature of programming languages compels novices on several concepts like variable types, loops, and conditional statements (Yadav et al., 2017; Robins et al., 2003; Brown & Wilson, 2018; Guo, 2017). Brown and Wilson (2018) claim that not all students have difficulty engaging with computing classes. While some novice students can effortlessly grasp the programming language syntax, some need help to handle the semantic concepts of programming language. Liao and Pope (2008) indicate that non-computer-major students are unwilling to deal only with numbers and semantic words on a white text window. Old-fashioned course materials cannot motivate students. Out-of-date computing exercises cannot help students build essential knowledge. The majority of studies advocate for educators to employ modern, high-level programming languages that generate visual output, including Processing, TouchDesigner, MAX, VVVV, and numerous others. Conversely, Hansen (2019) posits that the specific programming language employed in computing classes is inconsequential, provided that students are motivated by the relevance of the course content. As long as students are motivated by the relevancy of the course content, they internalize programming as a helpful aptitude for their vocational life (Lohiniva et al., 2021). It is therefore evident that enhancing the motivation and engagement of students represents a pivotal element within the context of programming classes. One potential solution to this issue is the implementation of a contextualized teaching approach. Previous research has demonstrated that teaching in context can positively influence student motivation (Bryant et al., 2011; Guzdial, 2006; Hansen, 2019). For students who are not majoring in computer science, contextualized courses can serve as an introductory gateway, enhancing the accessibility and relatability of the subject matter (Guzdial, 2010).

Computational thinking (CT) as a pedagogical approach has its roots in Seymour Papert's constructionist learning theory. It is considered to be an effective method for teaching programming tasks (Papert, 1980). CT has been proposed as a practical problem-solving approach, mainly through its emphasis on decomposition (Mollu, 2020). As a pedagogical approach has its roots in Seymour Papert's Constructionist learning theory (Papert, 1980). Papert's approach to constructionism highlights learning through active involvement and the creation of knowledge. This approach laid a solid theoretical groundwork for the principles and methods that later became linked with CT (Wing, 2008). Generally, CT has four main principles as a problem-solving method (Hansen, 2019). Decomposition entails separating objects, Pattern Recognition identifies recurring patterns, Abstraction involves representing the translation of collected ideas into computer domain, and finally, Algorithm Design interests arranging the order of the syntactic commands in the most optimal way related to identified programming tasks. The initial step is to deconstruct the intricate issues into smaller, more manageable components. By breaking them down into more manageable parts, it becomes possible to analyze and address each aspect effectively. Novices can handle the decomposition and pattern recognition steps using their natural language (Medeiros et al., 2019). However, effectively applying this principle requires overcoming a significant challenge because of the tacit nature of computer programming. Tacit knowledge refers to the implicit understanding gained through experience and practice, which is often challenging to articulate and share explicitly (Polanyi & Sen, 2009). In programming practice, conveying the specific heuristics and strategies used to decompose complex problems makes it challenging to implement CT principles effectively, which poses a barrier to learners and practitioners. As a result, motivational issues arise, and students tend to drop classes or lose interest in the topic (Farah et al., 2020). Research indicates that beginner programmers demonstrate an intuitive grasp of step-by-step instructions in natural language, influencing their approach to programming tasks (Bonar & Soloway, 1983, 1985). While natural language aids in understanding computational concepts, it also presents challenges when used for coding (Good & Howland, 2017). Novices often require support translating their natural language understanding into formal programming languages, leading to misconceptions and bugs (Bonar & Soloway, 1985). Studies have shown that beginners can articulate required instructions in natural language narratives but need help converting these ideas into programming constructs (Souza et al., 2011). In light of these challenges, researchers have put forth design guidelines for novice programming environments that take into account the role of diverse notations, including natural language, in facilitating a range of programming activities (Good & Howland, 2017). Despite teaching in context increases student motivation and CT eases the process of analyzing and identifying the programming tasks, novices face off to a new layer of challenge. In the context of computer art, novices need to learn programming, and also complex algorithms used by artists. While CT lacks of customized tools, teaching in art context demands for complex programming paradigms (Medeiros et al., 2019). Repenning et al (2016) notes that there need to be tools for different contexts of Computational Thinking. Computational Thinking Tools are designed to educate users and promote computational thinking. It is imperative that they address not only the syntactic aspects of programming, but also the semantic and pragmatic elements, while providing support for the formulation of problems, the expression of solutions, and the execution and evaluation of solutions. This approach can facilitate computational thinking in various disciplines without introducing unnecessary complexity. These findings highlight the necessity of establishing a connection between novices' intuitive understanding and formal programming languages in the context of computational thinking and programming education. Following these implications, there needs to be additional materials in order to ease the process of knowledge translation. In that sense, we articulate the following research question: ***"How can we integrate programming fundamentals with Algorithmic Art to enhance computational thinking skills, such as pattern recognition, abstraction, and algorithm design that emphasize real-world applications for higher education?"*** To answer this question, the systematic review and content analysis of 2000 images representative Algorithmic works of art were performed. In light of our research findings, we have devised a framework and presented a case study that educators can employ in the context of contextualized programming classes. We created semiotically meaningful constructs by identifying and classifying common strategies used in algorithmic art. These constructs act as shared symbols and representations, facilitating communication and knowledge sharing about tacit knowledge related to CT application.

2. Methodology

The development of the Algorithmic Art Praxis study involves a thorough review of online secondary data sources, including online databases, art galleries, and research centers. We created an online database[1] using a third-party web application to organize all resources in one place and provide learning and practice material for other researchers, educators, and students. The data was systematically analyzed by collecting and categorizing the artworks based on their formal aspects, such as style, medium, and composition. The categorization system, referring to the programming practice, was developed using an iterative design approach, which involved continuous refinement based on recorded samples related to the creative coding practices.

2.1. Data Collection

A comprehensive review was conducted of 2,220 secondary source images, from which 695 were selected for detailed analysis and recorded as entries in the database. Data was collected from reputable sources, including museum websites, digital art archives, and scholarly publications (Table 1). In selecting images for the online repository, consideration is given to the formal aspects of the artwork's visual composition. Provided that the final image is not solely a three-dimensional rendered or realistic image, it may be added to the collection for analysis.

The utilization of secondary data sources as a sample collection method is a crucial aspect of this research. The artworks' images and scanned photographs are collected from various sources, including gallery web pages, funded research collectives, published books, and magazines. When choosing a data source, it is of utmost importance to prioritize websites that are affiliated with official institutions,

Table 1: *List of secondary sources.*

Name	Content Type	URL
Atari Archives	Computational	www.atariarchives.org
Computer Art	Computational	dada.compart-bremen.de
DAM.org	Computational	dam.org
Digital Art Museum	Computational	digitalartmuseum.org
Guggenheim	Mixed	www.guggenheim.org
Internet Archive: Computers and Automation (1940-1980)	Mixed	www.archive.org
MOMA	Mixed	www.moma.org
MOMA San Francisco	Mixed	www.sfmoma.org
The MET Museum	Mixed	www.metmuseum.org
Monoskop	Mixed	monoskop.org
Rhizome	Mixed	rhizome.org
Scanlines: Computers & Art (1970-1980)	Mixed	scanlines.xyz
Spalter Digital	Mixed	spalterdigital.com
TATE	Mixed	www.tate.org.uk
The Art Story	Mixed	www.theartstory.org
TOPLAP	Mixed	toplap.org
Victoria & Albert Museum	Mixed	collections.vam.ac.uk
Whitney Museum of American Art	Mixed	whitney.org
WikiArt	Mixed	www.wikiart.org
ZKM	Mixed	zkm.de/en

communities, or other trustworthy sources. In pursuit of this objective, we have drawn upon the websites of distinguished institutions such as the Museum of Modern Art (MoMA) and the Guggenheim Museum, as well as those of esteemed researchers in the field, including DAM (Digital Art Museum), Spalter Digital, and Monoskop. For a comprehensive list of the research sources consulted, please refer to Table 1. The table features three columns, each containing relevant information in an organized manner as following;

- **Name:** Official name of the institution/collective/research center/museum.
- **Content Type**: The website's content can be classified into two principal categories: computational and mixed. Computational content encompasses technology-based artworks, including computer art, algorithmic art, and (new) media art. Mixed content, in contrast, incorporates both traditional and technology-based works of art.
- **URL:** Includes the website address of the relevant item.

Mixed type content sources involve a large repository of artists from diverse fields and artworks with variable mediums. The ones in the Table 1 like Victoria & Albert Museum website is not dedicated to works of computer art. However these sources collected vast amount of algorithmic art works as well. Rather than searching by artist names, keyword or tag based search method allows access all available works of art under the relevant keyword. The keywords used for searching the databases including mixed type are “Computer Art,” “Algorithmic Art,” “Generative Art,” and “(New) Media.”

The sources with “Computational” type like Anne and Michael Spalter Digital Art Collection, also known as Spalter Digital, is a prominent private collection of early computer art, renowned for its extensive scope and significance. It is home to over 1000 artwork images, making it one of the world's largest collections in the context of computer art, as per our research. While its primary focus lies in plotter drawings, the collection encompasses various other 2D media, including sculpture and 16mm film. It boasts major works and iconic pieces created by key artists in the field.

We utilized various secondary data sources in our research, including the Internet Archive[2] (IA), which offers a wide array of document sources covering diverse topics (Internet Archive, 2023). Our exploration of the IA website led to the discovery of documents related to early computer art, including one of the earliest open calls for computer art in the "Computers and Automation" magazine, which was published from the 1940s to the 1980s before assuming the name "Computers and People" in the 1970s (Berkeley, 1963; Franke, 1971; Macdonald, 1981). IA was an invaluable resource for accessing early works of algorithmic art and provided insight into early programming practices for creating algorithmic compositions from the 1950s to the 1970s. Our thorough review encompassed a total of 315 issues from the IA website, uncovering some of the earliest examples of computationally generated art. The complete archive is accessible on the IA webpage, and we meticulously reviewed each volume by examining pages and contents. Our review of the monthly issues involved two methods: The first method entailed searching for keywords such as "computer art," "computer-art," "algorithmic art," or "algorithmic-art." However, locating specific text within the scanned magazines proved challenging. The second method involved generating thumbnails of each page using a third-party application or IA's built-in PDF viewer, enabling us to more accurately and efficiently identify computer-generated artworks. This approach facilitated gathering more comprehensive information about the algorithmic artworks and their production methods.

2.2. Online Database System and Interface

In order to develop an accessible web application, we employed Notion[3], a note-taking application with a well-structured database infrastructure that perfectly aligns with our requirements. By utilizing Notion's relational database framework, we are able to input and deploy data to the web swiftly and efficiently, thus saving valuable time and resources. Furthermore, its cloud-based system enables us to instantly upload new images, input data, and promptly publish the new content online.

The database consists of 695 entries we carefully selected from a pool of 2200 samples discussed in the preceding section. Each entry contains ten distinct properties that provide information about the entry. Table 1 illustrates the database infrastructure. The Medium and Classification parameters can accommodate multiple values based on information obtained from image sources. The Praxis parameter can also hold multiple values derived from contextual analysis of the artwork about programming principles. Except for the category names in the Praxis section, all the information has been utilized as indicated in the relevant sources. The naming convention for the category names in the Praxis section will be elucidated in the subsequent section.

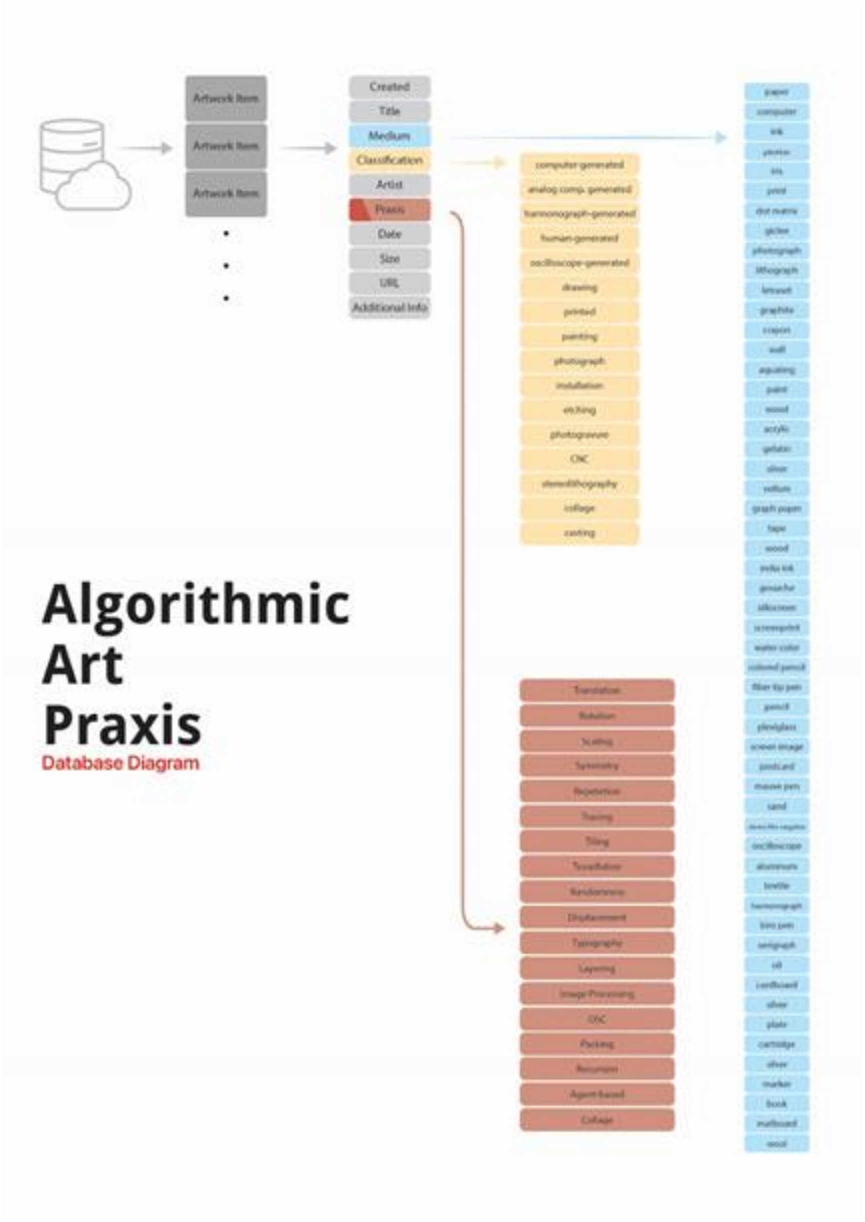


Figure 1: *Algorithmic Art Praxis Database Infrastructure*

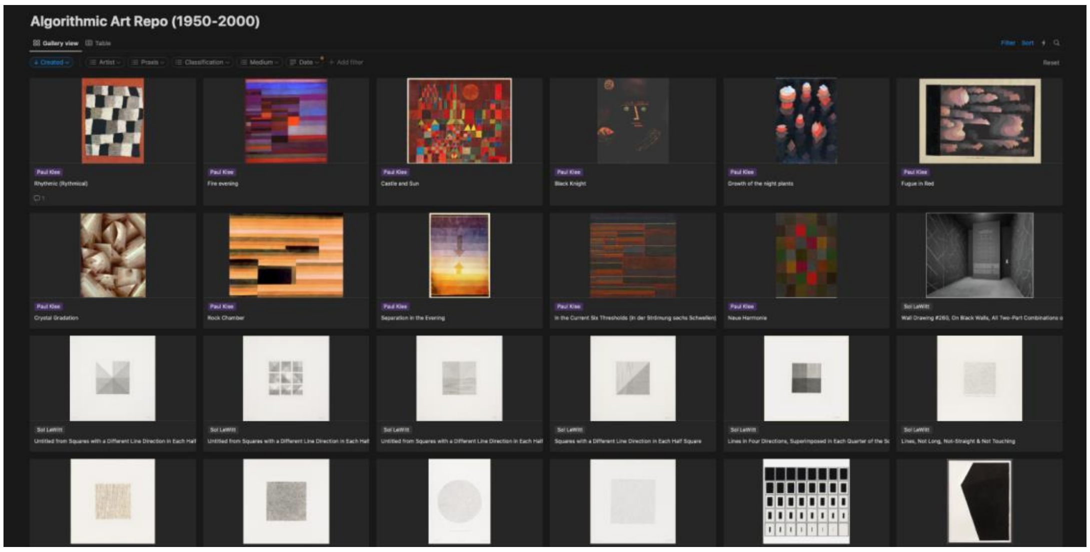

Algorithmic Art Repo (1950-2000)
Gallery view
Table
Filter
Sort
Created
Artist
Praxis
Classification
Medium
Date
+ Add filter
Reset
Paul Klee
Rhythmic (Rythmical)
Paul Klee
Fire evening
Paul Klee
Castle and Sun
Paul Klee
Black Knight
Paul Klee
Growth of the night plants
Paul Klee
Fugue in Red
Paul Klee
Crystal Gradation
Paul Klee
Rock Chamber
Paul Klee
Separation in the Evening
Paul Klee
In the Current Six Thresholds (In der Strömung sechs Schwellen)
Paul Klee
Neue Harmonie
Sol LeWitt
Wall Drawing #260, On Black Walls, All Two-Part Combinations o
Sol LeWitt
Untitled from Squares with a Different Line Direction in Each Half
Sol LeWitt
Untitled from Squares with a Different Line Direction in Each Half
Sol LeWitt
Untitled from Squares with a Different Line Direction in Each Half
Sol LeWitt
Squares with a Different Line Direction in Each Half Square
Sol LeWitt
Lines in Four Directions, Superimposed in Each Quarter of the Sc
Sol LeWitt
Lines, Not Long, Not-Straight & Not Touching

Figure 2: *ALAP Database Thumbnail View is online and accessible at* **<u>https://tinyurl.com/alap-database</u>**.

Each entry in the database consists of a singular image or multiple variations of the same artwork if the artist created them. The name of the artist and the artwork are displayed on each item in the gallery, as shown in Figure 3.

Users can access detailed information When they interact with an artwork displayed in the gallery (Figure 4). The gallery items share common parameters.

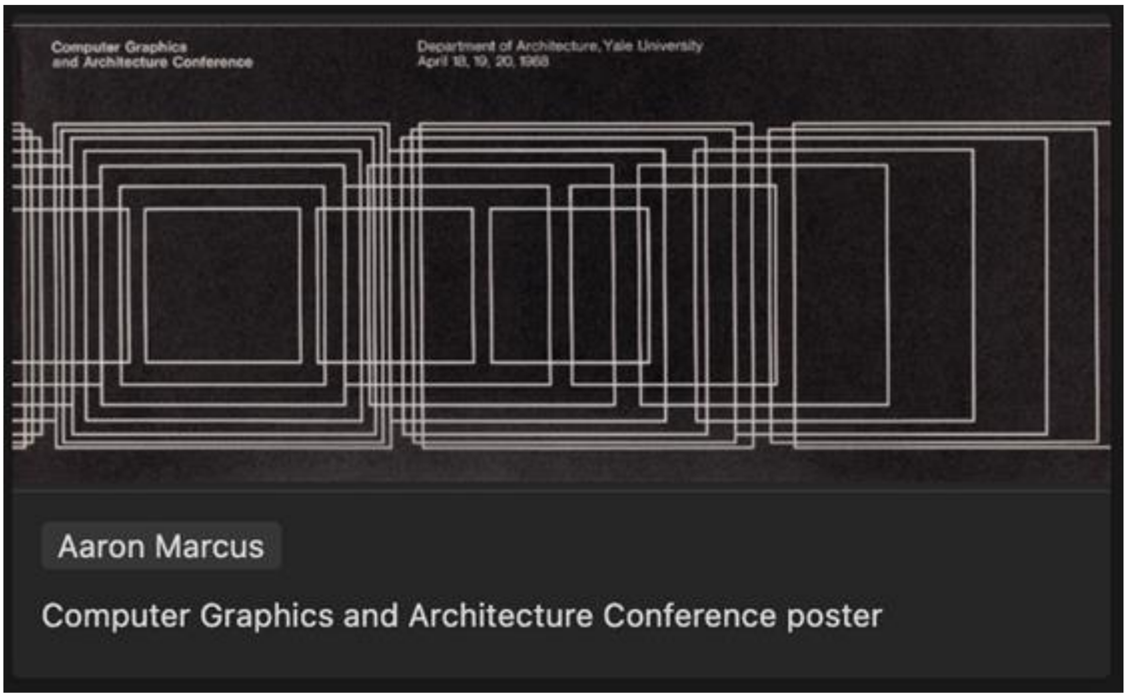
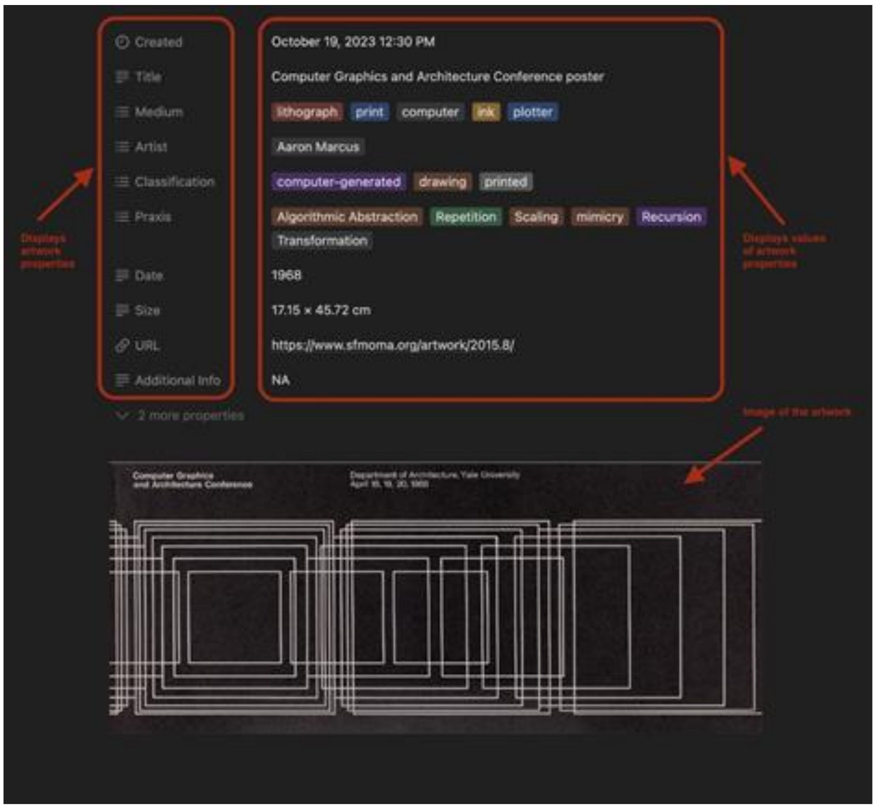

Computer Graphics
and Architecture Conference
Department of Architecture, Yale University
April 18, 19, 20, 1968
Aaron Marcus
Computer Graphics and Architecture Conference poster

Figure 3: *Gallery Item Close Up View*



Created	October 19, 2023 12:30 PM
Title	Computer Graphics and Architecture Conference poster
Medium	lithograph print computer ink plotter
Artist	Aaron Marcus
Classification	computer-generated drawing printed
Praxis	Algorithmic Abstraction Repetition Scaling mimicry Recursion Transformation
Date	1968
Size	17.15 × 45.72 cm
URL	https://www.sfmoma.org/artwork/2015.8/
Additional Info	NA

Displays artwork properties
Displays values of artwork properties
2 more properties
Image of the artwork
Computer Graphics and Architecture Conference
Department of Architecture, Yale University April 18, 19, 20, 1968

Figure 4: *Detail View of Clicked Artwork Item*

The following Table 2 lists the available sample (artwork) parameters, and their relevant descriptions related to Figure 4.

The database is designed to display a single image for each artwork item. However, the increase in the number of samples has led to an

Table 2: *Database Item Parameters and Descriptions*

Parameter	Description
Created	Database Item creation date.
Title	The artwork’s official title.
Medium	The tangible supplies that were utilized to make the artwork.
Artist	The artist(s)’s name and last name.
Classification	Processes used to produce the artwork.
Praxis	Outlines the potential algorithmic practice categories that could be applied to produce the artwork.
Date	Artwork creation date.
Size	The original dimensions of the artwork (metric or inch units).
URL	The official source of the artwork.
Additional Info	Information regarding the technical hardware and methods used in creating the artwork.

issue with similar artworks. This similarity is due to algorithmic art techniques, such as randomness, weighted distribution of numbers, and stochastic decision processes (Bailey, 2018; Phon-Amnuaisuk & Panjapornpon, 2012). Artists who use computers and algorithms to create variations of the same artwork by tweaking parameters based on computational paradigms produce results that are formally similar but aesthetically different (Boden & Edmonds, 2009; Galanter, 2016). At times, artists may choose to exhibit all iterations, while other times they may not. The images in the database have been thoughtfully organized based on their formal resemblances. If a set of works was produced within an iterative approach, an artwork item in the database may include multiple images, as explicitly declared by the source or if it is obvious. For example, Aaron Marcus' Cybernetic Landscapes series (Figure 5). The main goal was to group similar artworks together, prioritizing this over arranging them according to their series. Nonetheless, all relevant details have been provided to aid fellow researchers in tracing the origins of the artwork, including information on whether it belongs to a series or not.

2.3. Analysis and Results

The development of the ALAP categories followed an iterative design process (Figure 6). This approach allowed for continuous refinement of the categorization system as new data was incorporated into the ALAP database presented in the previous section. The Praxis categories have been carefully crafted to align with the formal and compositional aspects of the artworks. Through a thorough review of over 695 algorithmic works of art samples from 1920 to 2000, 18 distinct category names have been identified.

The main inspiration for the category names is the fundamental ingredient of creating a visual representation, the vertex. In computer graphics, geometric forms are defined through points per pixel. A vertex represents a point in space determined by both x and y coordinates. We can position the vertex in a computer

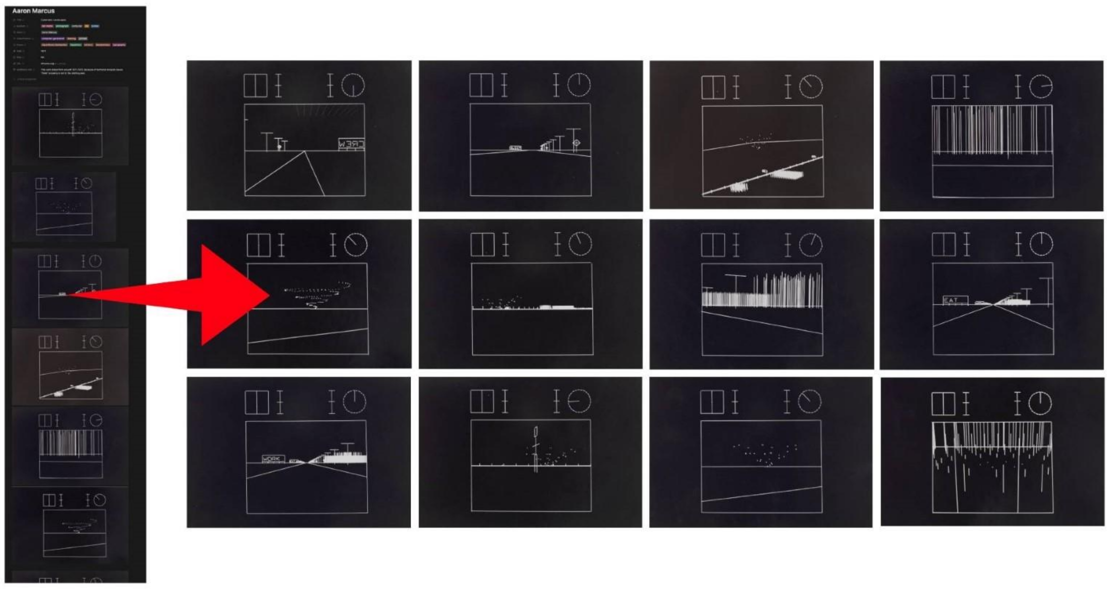


Figure 5: *Cybernetic Landscapes Series by Aaron Marcus*

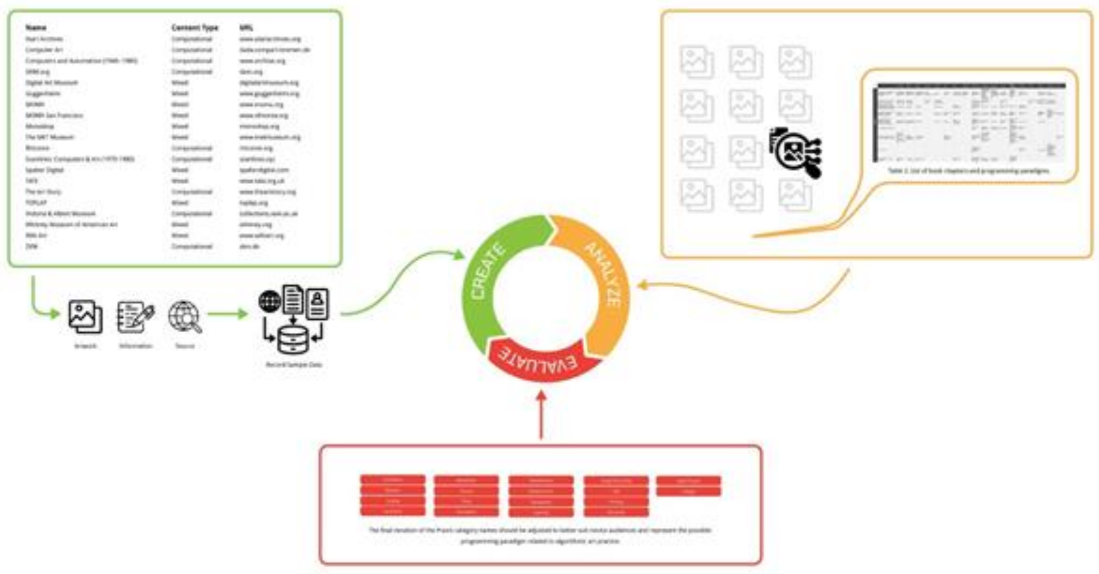


Figure 6: *Cybernetic Landscapes Series by Aaron Marcus*

window at any location. If we increase the number of vertices on the canvas, we can create more complex forms. A line comprises two vertices, while a triangle requires three, and a quadrilateral comprises four vertices, as shown in Figure 7. Whether a simple shape or a complex geometric form, the graphical element on the screen is the cause of vertices organization. By moving or rotating the vertices around a point, we can transform a geometric form into a leaf or increase the size of the leaf to fit our needs. In a basic sense, we can create any visual element using these three actions. We took these three properties for granted while determining the category names in the Praxis category. Thanks to programming language references and pre-coded examples, we compiled a list of function and algorithm names to serve as category names for representing a programming practice of the artworks.

We derived the category names using the list referenced from programming terms and algorithm names within creative coding frameworks depending on textual programming languages such as Processing (Reas & Fry, 2007; Shiffman, 2008; Terzidis, 2009; Pearson, 2011), P5JS (McCarthy et al., 2016), and openFrameworks (Noble, 2009; Perevalov & Tatarnikov, 2015). In our analysis, we meticulously document the formal aspects of elements in artwork images, focusing on the vertices of graphical elements. We thoroughly review the book chapters to identify relevant programming practices, and we organize each chapter with its respective page number under

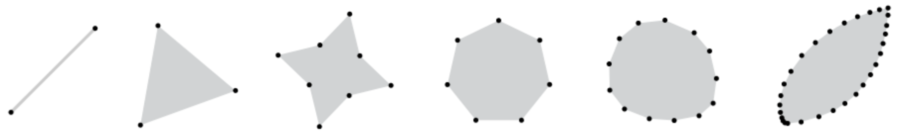
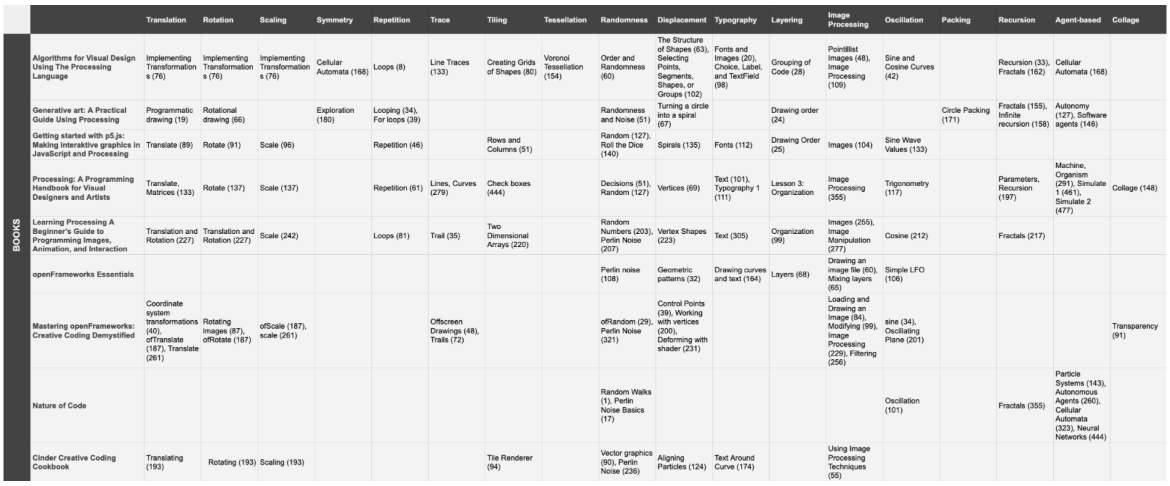


Figure 7: *Vertices shape simple and complex polygons in computer graphics*



		Translation	Rotation	Scaling	Symmetry	Repetition	Trace	Tiling	Tessellation	Randomness	Displacement	Typography	Layering	Image Processing	Oscillation	Packing	Recursion	Agent-based	Collage
BOOKS	Algorithms for Visual Design Using The Processing Language	Implementing Transformations (76)	Implementing Transformations (76)	Implementing Transformations (76)	Cellular Automata (168)	Loops (8)	Line Traces (133)	Creating Grids of Shapes (80)	Voronoi Tessellation (154)	Order and Randomness (60)	The Structure of Shapes (63), Selecting Points, Segments, Shapes, or Groups (102)	Fonts and Images (20), Choice, Label, and TextField (98)	Grouping of Code (28)	Pointillist Images (48), Image Processing (109)	Sine and Cosine Curves (42)		Recursion (33), Fractals (162)	Cellular Automata (168)	
	Generative art: A Practical Guide Using Processing	Programmatic drawing (19)	Rotational drawing (66)		Exploration (180)	Looping (34), For loops (39)				Randomness and Noise (51)	Turning a circle into a spiral (67)		Drawing order (24)			Circle Packing (171)	Fractals (155), Infinite recursion (158)	Autonomy (127), Software agents (146)	
	Getting started with p5.js: Making interaktive graphics in JavaScript and Processing	Translate (89)	Rotate (91)	Scale (96)		Repetition (46)		Rows and Columns (51)		Random (127), Roll the Dice (140)	Spirals (135)	Fonts (112)	Drawing Order (25)	Images (104)	Sine Wave Values (133)				
	Processing: A Programming Handbook for Visual Designers and Artists	Translate, Matrices (133)	Rotate (137)	Scale (137)		Repetition (61)	Lines, Curves (279)	Check boxes (444)		Decisions (51), Random (127)	Vertices (69)	Text (101), Typography 1 (111)	Lesson 3: Organization	Image Processing (355)	Trigonometry (117)		Parameters, Recursion (197)	Machine, Organism (291), Simulate 1 (461), Simulate 2 (477)	Collage (148)
	Learning Processing A Beginner's Guide to Programming Images, Animation, and Interaction	Translation and Rotation (227)	Translation and Rotation (227)	Scale (242)		Loops (81)	Trail (35)	Two Dimensional Arrays (220)		Random Numbers (203), Perlin Noise (207)	Vertex Shapes (223)	Text (305)	Organization (99)	Images (255), Image Manipulation (277)	Cosine (212)		Fractals (217)		
	openFrameworks Essentials									Perlin noise (108)	Geometric patterns (32)	Drawing curves and text (164)	Layers (68)	Drawing an image file (60), Mixing layers (65)	Simple LFO (106)				
	Mastering openFrameworks: Creative Coding Demystified	Coordinate system transformations (40), ofTranslate (187), Translate (261)	Rotating images (87), ofRotate (187)	ofScale (187), scale (261)			Offscreen Drawings (48), Trails (72)			ofRandom (29), Perlin Noise (321)	Control Points (39), Working with vertices (200), Deforming with shader (231)			Loading and Drawing an Image (84), Modifying (99), Image Processing (229), Filtering (256)	sine (34), Oscillating Plane (201)				Transparency (91)
	Nature of Code									Random Walks (1), Perlin Noise Basics (17)					Oscillation (101)		Fractals (355)	Particle Systems (143), Autonomous Agents (260), Cellular Automata (323), Neural Networks (444)	
	Cinder Creative Coding Cookbook	Translating (193)	Rotating (193)	Scaling (193)				Tile Renderer (94)		Vector graphics (90), Perlin Noise (236)	Aligning Particles (124)	Text Around Curve (174)		Using Image Processing Techniques (55)					

Figure 8: *Sources and book content relevant to the ALAP Categories*

the appropriate draft category name. Figure 8 displays the finalized category names, with book titles listed along the vertical axis and relevant topics associated with each category name listed along the top horizontal axis. Each cell within the table contains the corresponding book title and the page number in parentheses.

The iterative design process involves creating, testing, and refining a product/concept until it achieves the desired outcome. This process includes continuous comparisons, improvement, and adaptation to evolving needs. Creative coding frameworks like P5JS are primarily designed in high-level programming languages, resembling written human natural language in English. Consequently, for our research, the naming convention aims to mirror natural language as much as possible. We aim to keep Praxis category names comprehensible for novices without additional research into a programming practice. The process comprises three key phases: creation, analysis, and evaluation, illustrated in Figure 6.

1) **Creation**: Each image was added to the database as a sample, and its data was recorded based on the information obtained. Notably, no entry was made for the Praxis section during this stage.

2) **Analysis**: Once all the information collected from the source was entered into the database, the Praxis category was identified based on the formal and compositional features of the artwork, and its name was directly derived from the list of categories illustrated at Figure 8.

3) **Evaluation**: As the dataset grew, the Praxis categories underwent frequent review and refinement to accommodate the increasing number of new images. This led to further enhancements of the existing categories and, in some cases, the introduction of new ones to maintain precision and ensure an accurate representation of the data. For example, the initial version of "Transformation" was later changed to "Translate" to align with creative coding paradigms and programming languages. Eventually, "Translate" was refined to "Translation" to emphasize the practice rather than referencing a specific programming language syntax, as the Praxis category names need to be programming language agnostic. It was important to avoid confusion for beginners using different programming languages than Processing or P5JS, hence the adjustment from "Translate" to "Translation."

The iterative design process allowed the Praxis category system to evolve and adapt to the increasing data, fostering a robust and flexible framework for algorithmic art categorization to represent relevant programming practice. This evolution ensures that the system remains robust and up-to-date, ready to handle the

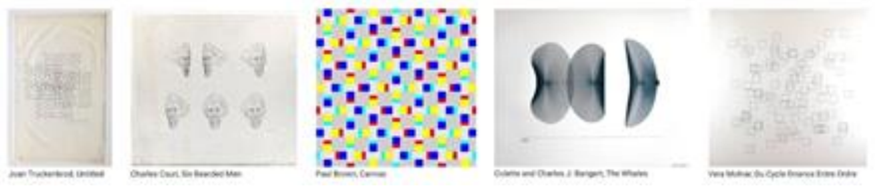

Joan Truckenbrod, Untitled
Charles Csuri, Six Bearded Men
Paul Brown, Canvas
Colette and Charles J. Bangert, The Whales
Vera Molnar, Du Cycle Errance Entre Ordre

Figure 9: *Artworks showcase transformational behaviours.*

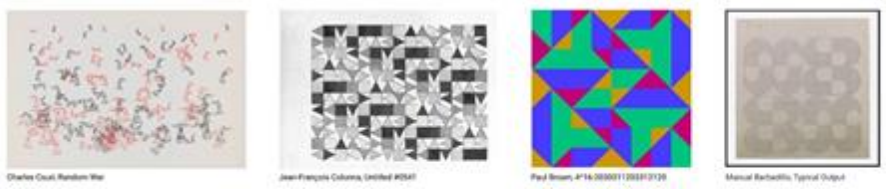

Charles Csuri, Random War
Jean-François Colonna, Untitled #0547
Manuel Barbadillo, Typical Output

Figure 10: *Artworks showcase rotational behaviours*

complexities of algorithmic art categorization. We provide comprehensive explanations and a collection of sample artwork images from our online ALAP database for every category listed below. It should be noted that all reference images of artworks per category can be accessed in larger resolution via the online ALAP Database as well (https://tinyurl.com/alap-database). Also, there are video tutorials on how to use the interface of the database[4].

Translation: It refers to changing the x and y coordinates of the individual vertices separately or all together. Transformation is close to Displacement. The difference is that the vertex points gradually form from a primitive shape to another complex shape. "Return to Square" is an excellent example of this behaviour. The main property to define transformation can be viewed in (Return to Square) work. The use of translate function can be used (Figure 9).

Rotation: It represents the change in the orientation of objects on the canvas (Figure 10).

Scaling: It represents the change in the dimensions of objects on the canvas (Figure 11).

Symmetry: It depicts mirrored forms of representations. Sometimes, it is combined with other Praxis, and results in minor differences might occur (Figure 12).

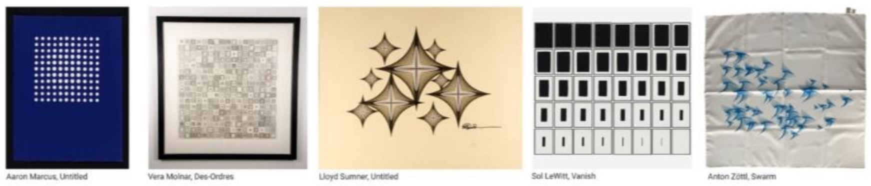


Figure 11: *Artworks showcase dimensional behaviours*

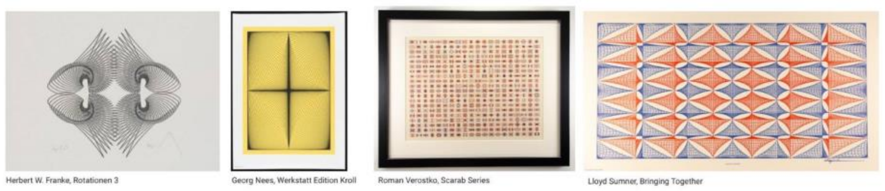

Herbert W. Franke, Rotationen 3
Georg Nees, Werkstatt Edition Kroll
Roman Verostko, Scarab Series
Lloyd Sumner, Bringing Together

Figure 12. *Artworks showcase symmetric behaviours*

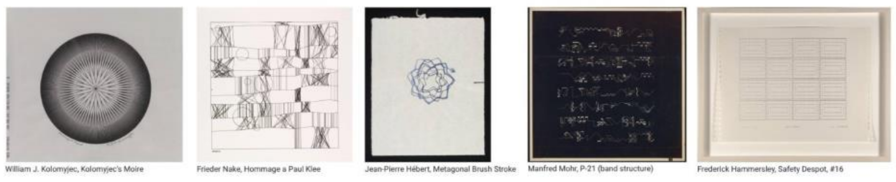

William J. Kolomyjec, Kolomyjec's Moire
Frieder Nake, Hommage a Paul Klee
Jean-Pierre Hébert, Metagonal Brush Stroke
Manfred Mohr, P-21 (band structure)
Frederick Hammersley, Safety Despot, #16

Figure 13: *Artworks showcase repetitive behaviours*

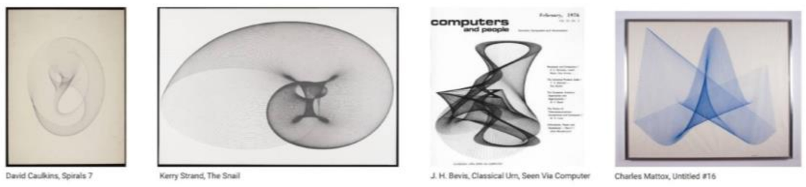

David Caulkins, Spirals 7
Kerry Strand, The Snail
J. H. Bevis, Classical Urn, Seen Via Computer
Charles Mattox, Untitled #16

Figure 14*: Artworks showcase leaves of traces behaviours*

Repetition: It means repeating an action manually or computationally repeatedly in a limited amount of cycles to produce the visual form. Repetition is often used to create patterns, textures, and complex visual structures. Loops are a common way to implement repetition in computer programs. They allow a specific set of instructions to be repeated a certain number of times (Figure 13).

Trace: It occurs when the object's opacity decreases or increases through the canvas. The main difference between Tracing and Layering is the way they represent how the following or upcoming forms are structured. Tracing is a continuous set of repetitions, whereas layering is more like the style of color printing techniques applied in traditional printing press (Figure 14).

Tiling: Tiling is a way of creating a grid-based distribution on the canvas. Individual patterns in the grids do not have to be continuous or mixed with each other. Multiple objects or object groups can be positioned in a regular grid manner. Individual patterns in the grid can be different from each other (Figure 15).

Tessellation: The art of Tessellation is a one-of-a-kind computer-generated technique that produces visually appealing and seamless patterns by utilizing a variety of shapes. This artistic form has a lengthy history and can be observed in numerous creative and mathematical ventures (Torrence, 2021). Even though Tessellation and tiling both involve covering a surface with a pattern of flat shapes, these terms are different. Tessellation pertains specifically to the creation of patterns by fitting shapes together without gaps or overlaps. In

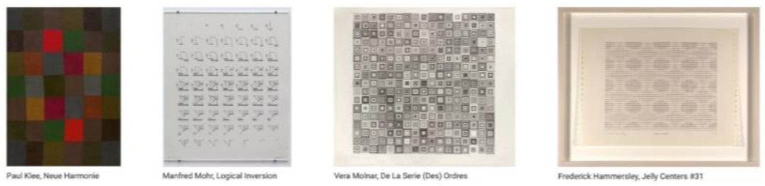

Paul Klee, Neue Harmonie
Manfred Mohr, Logical Inversion
Vera Molnar, De La Serie (Des) Ordres
Frederick Hammersley, Jelly Centers #31

Figure 15*: Artworks showcase tiled behaviours*

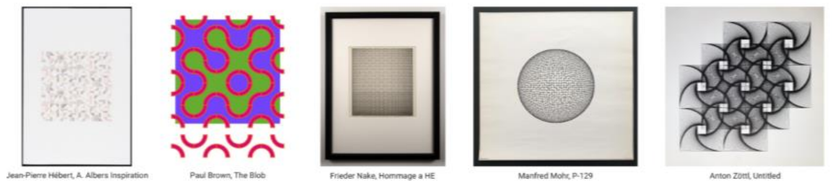

Jean-Pierre Hébert, A. Albers Inspiration
Paul Brown, The Blob
Frieder Nake, Hommage a HE
Manfred Mohr, P-129
Anton Zöttl, Untitled

***Figure 16**: Artworks showcase tessellated behaviours*

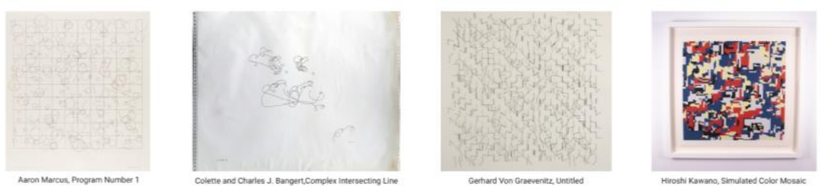

Aaron Marcus, Program Number 1
Colette and Charles J. Bangert,Complex Intersecting Line
Gerhard Von Graevenitz, Untitled
Hiroshi Kawano, Simulated Color Mosaic

***Figure 17:** Artworks showcase random behaviours*

Tessellation, each tile must have a relational and formal connection in order to create a grid-based distribution. Tiling, on the other hand, is a more general term that refers to covering a surface with a pattern of flat shapes that may or may not meet the specific requirements of Tessellation. In short, every Tessellation involves Tiling, but not every Tiling can be considered a Tessellation (Figure 16).

Randomness: It represents stochastic decisions executed through a series of numbers by pre-defined programming tools. Similar to the notion of throwing a dice or tossing a coin in real life. There is a 1/6 possibility of getting six from dice and a 1/2 possibility with a coin to get head or tails. Sometimes artists get benefit from the random decisions (Figure 17).

Displacement: There must be a form that can be generated at least 3 points. It is the act of repetition by modifying the existing form. Displacement tells us the change of a specific vertex or vertices position. The formal changes must be observable, such as in Figure 18.

Typography: It means a typographic element used in the artwork, like fonts or graphics, that generates textual forms (Figure 19).

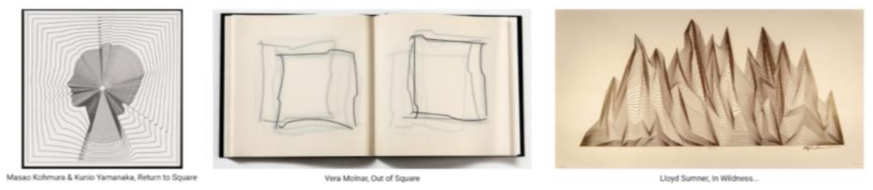

Masao Kohmura & Kunio Yamanaka, Return to Square
Vera Molnar, Out of Square
Lloyd Sumner, In Wildness...

***Figure 18:** Examples of artworks exhibiting displacement behaviours*

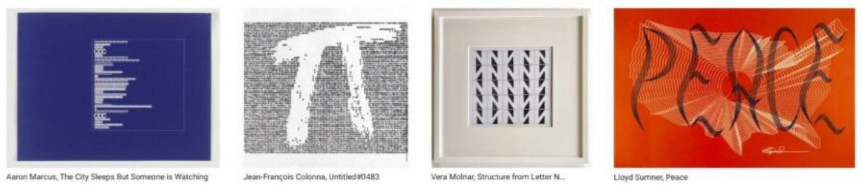

Aaron Marcus, The City Sleeps But Someone is Watching
Jean-François Colonna, Untitled#0483
Vera Molnar, Structure from Letter N...
Lloyd Sumner, Peace

***Figure 19:** Examples of artworks exhibiting typographic features*

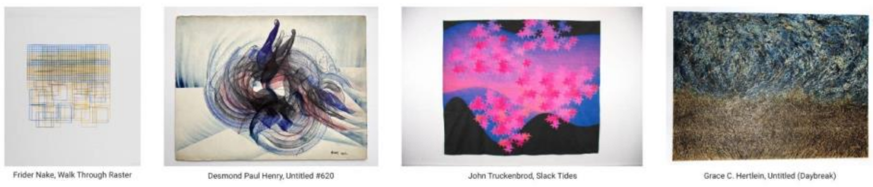

Frider Nake, Walk Through Raster
Desmond Paul Henry, Untitled #620
John Truckenbrod, Slack Tides
Grace C. Hertlein, Untitled (Daybreak)

Figure 20: *Examples of artworks that exhibit stratified behaviours*

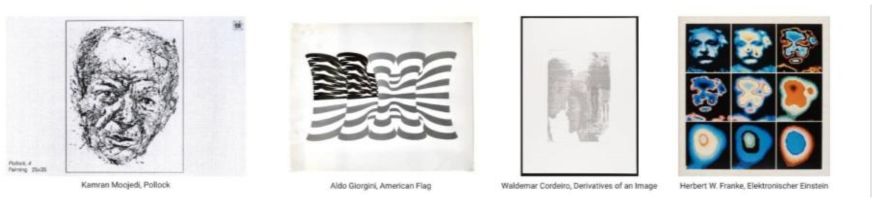

Kamran Moojedi, Pollock
Aldo Giorgini, American Flag
Waldemar Cordeiro, Derivatives of an Image
Herbert W. Franke, Elektronischer Einstein

Figure 21: *Examples of artworks that demonstrate processed image behaviours*

Layering: Layering refers to the procedural drawing order of the visual elements on the canvas. An explanatory example of layering can be Frieder Nake's work, which is a type of layering. During the periods of plotters, it was not possible to draw multi-color images. Another approach was to redraw iterations of the same idea on the same paper by switching the marker or pen (Figure 20).

Image Processing: In image processing category, usually, the image is preloaded into the computer buffer. The artist may use additional filters to this data and create something similar or a completely different image from the loaded data (Figure 21).

Oscillation (OSC): The concept of oscillation pertains to the recurring pattern of periodic phenomena, such as that of a sine wave. One can observe a repetition of neighbouring points in the visual representation; it could imply the utilization of trigonometric functions. The periodic pattern may consist of distinctive alterations with each cycle. It does not have to depict the reoccurrence of the same graphical object due to the nature of computer art programming practices and the artist's intuition (Figure 22).

Packing (Space-Filling): Packing involves fitting the objects into a limited space (a.k.a space-filling or packing algorithm). The rule is that objects must not interfere with each other (Figure 23).

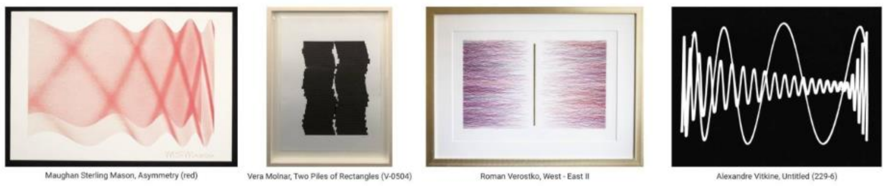

Maughan Sterling Mason, Asymmetry (red)
Vera Molnar, Two Piles of Rectangles (V-0504)
Roman Verostko, West - East II
Alexandre Vitkine, Untitled (229-6)

Figure 22: *Examples of artworks that demonstrate forms of wave like behaviours*

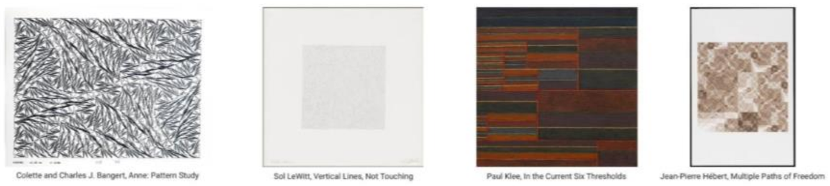

Colette and Charles J. Bangert, Anne: Pattern Study
Sol LeWitt, Vertical Lines, Not Touching
Paul Klee, In the Current Six Thresholds
Jean-Pierre Hébert, Multiple Paths of Freedom

Figure 23: *Examples of artworks that Packing behaviours*

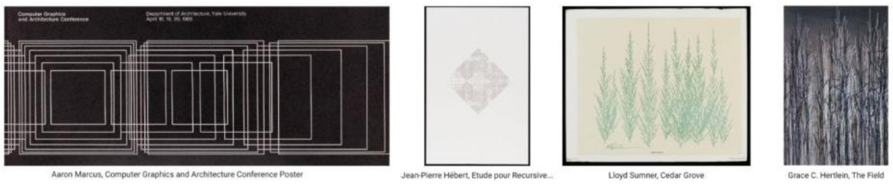

Aaron Marcus, Computer Graphics and Architecture Conference Poster
Jean-Pierre Hébert, Etude pour Recursive...
Lloyd Sumner, Cedar Grove
Grace C. Hertlein, The Field

Figure 24: *Examples of artworks that demonstrate recursive behaviours*

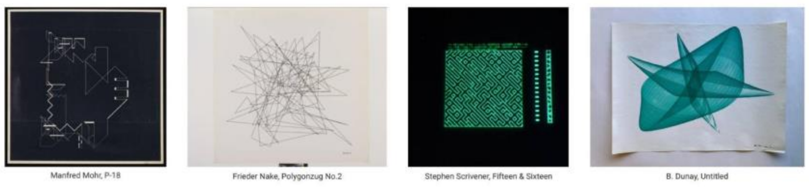

Manfred Mohr, P-18
Frieder Nake, Polygonzug No.2
Stephen Scrivener, Fifteen & Sixteen
B. Dunay, Untitled

Figure 25: *Examples of artworks that demonstrate autonomous behaviours*

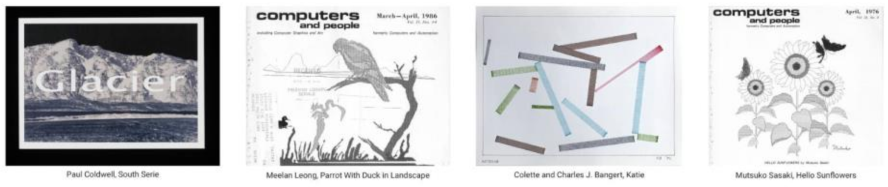

Glacier
computers and people
computers and people
Paul Coldwell, South Serie
Meelan Leong, Parrot With Duck In Landscape
Colette and Charles J. Bangert, Katie
Mutsuko Sasaki, Hello Sunflowers

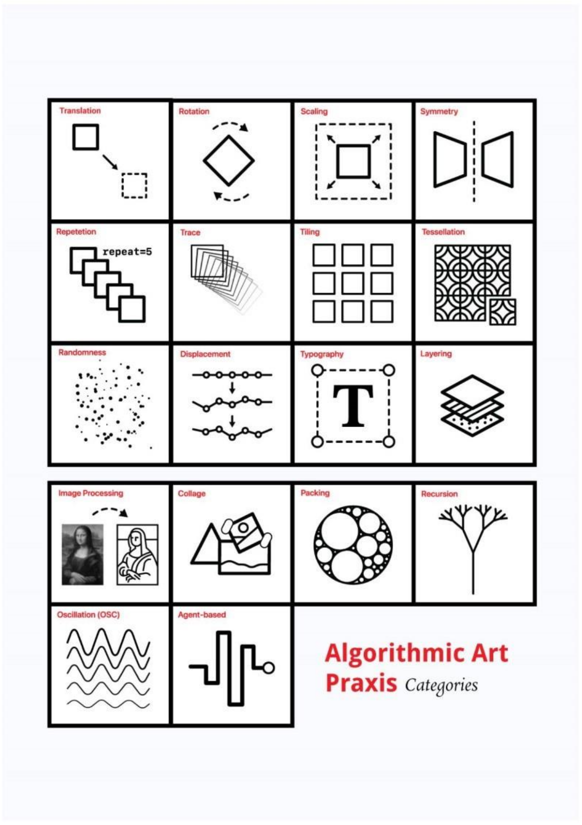
Figure 26: *Examples of artworks that demonstrate collage like behaviours*

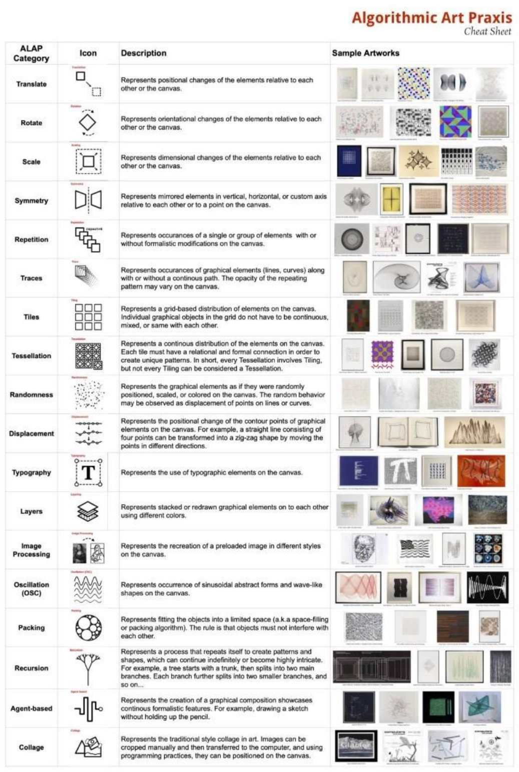
Recursion: Recursion is a programming technique where a function calls itself to solve a problem. It can be used to create complex and organic forms in computer art. Recursion is different than repetition in terms of paradigmatic aspects in programming. For example, a repetition using a for loop draws five circles in a row, each with a slightly different x-coordinate. A recursive function draws a fractal pattern by repeatedly drawing circles and calling itself with a smaller size—for example, Georg Nees. In summary, repetitions via for loops are well-suited for simplicity, while recursion can create more complex and organic patterns in computer art (Figure 24).

Agent-based: The artist creates an algorithm, a mechanical device, or instructs human agents to produce the artwork by instructing the agents partly or entirely (Figure 25).

Collage: Collage praxis category is like the traditional collage methods in art. Variable techniques can be applied and combined in algorithmic art. Images can be cropped manually and then transferred to the computer, and using programming practices, they can be positioned on the canvas (Figure 26).

The ALAP database can serve as customized learning material for contextualized programming education. Instructors can utilize examples from the database to demonstrate relevant programming practices. Novices can explore the database to understand the connection between programming practices and visual compositions within Algorithmic Art. Prior to advancing to the next section, it is necessary to introduce a pair of helpful tool for learners: the ALAP Categories and Cheat Sheet (Figure 28). The final stage of our study involves organizing all the information into a manageable framework and incorporating illustrations for each category item. The resource is designed for two separate A4 size paper and can benefit both learners and educators during the analysis of algorithmically created visuals.




Algorithmic Art Praxis
Cheat Sheet

ALAP Category	Icon	Description	Sample Artworks
Translate		Represents positional changes of the elements relative to each other or the canvas.	
Rotate		Represents orientational changes of the elements relative to each other or the canvas.	
Scale		Represents dimensional changes of the elements relative to each other or the canvas.	
Symmetry		Represents mirrored elements in vertical, horizontal, or custom axis relative to each other or to a point on the canvas.	
Repetition		Represents occurances of a single or group of elements with or without formalistic modifications on the canvas.	
Traces		Represents occurances of graphical elements (lines, curves) along with or without a continous path. The opacity of the repeating pattern may vary on the canvas.	
Tiles		Represents a grid-based distribution of elements on the canvas. Individual graphical objects in the grid do not have to be continuous, mixed, or same with each other.	
Tessellation		Represents a continous distribution of the elements on the canvas. Each tile must have a relational and formal connection in order to create unique patterns. In short, every Tessellation involves Tiling, but not every Tiling can be considered a Tessellation.	
Randomness		Represents the graphical elements as if they were randomly positioned, scaled, or colored on the canvas. The random behavior may be observed as displacement of points on lines or curves.	
Displacement		Represents the positional change of the contour points of graphical elements on the canvas. For example, a straight line consisting of four points can be transformed into a zig-zag shape by moving the points in different directions.	
Typography		Represents the use of typographic elements on the canvas.	
Layers		Represents stacked or redrawn graphical elements on to each other using different colors.	
Image Processing		Represents the recreation of a preloaded image in different styles on the canvas.	
Oscillation (OSC)		Represents occurrence of sinusoidal abstract forms and wave-like shapes on the canvas.	
Packing		Represents fitting the objects into a limited space (a.k.a space-filling or packing algorithm). The rule is that objects must not interfere with each other.	
Recursion		Represents a process that repeats itself to create patterns and shapes, which can continue indefinitely or become highly intricate. For example, a tree starts with a trunk, then splits into two main branches. Each branch further splits into two smaller branches, and so on...	
Agent-based		Represents the creation of a graphical composition showcases continous formalistic features. For example, drawing a sketch without holding up the pencil.	
Collage		Represents the traditional style collage in art. Images can be cropped manually and then transferred to the computer, and using programming practices, they can be positioned on the canvas.	

Figure 27: *ALAP Categories (left) and Cheat Sheet (right)*

As our primary audience consists of visually oriented learners, we have developed 18 distinct illustrations representing each category, along with a corresponding cheat sheet containing descriptions for each category. This resource aims to help beginners easily grasp the programming practice and patterns depicted in the artwork during their potential lectures. In addition to the online ALAP database, the printed versions of the categories and cheat sheet are intended to serve as tangible tools to facilitate the CT process during programming activities.

3. Case Study

In our case study, we will be utilizing the P5JS creative coding tool, which is based on the JavaScript programming language. However, individuals with intermediate or advanced programming skills can adapt and employ these concepts in other textual or visual programming languages. It is assumed that the students have a fundamental understanding of programming theory, including the concepts of variables, functions, and loops.

The case study will follow the following steps

1. Choosing an artwork from the online ALAP database.
2. Printing the ALAP Categories and Cheatsheet.
3. Applying Computational Thinking principles to analyze the artwork.
4. Using the ALAP Categories to address the programming tasks.

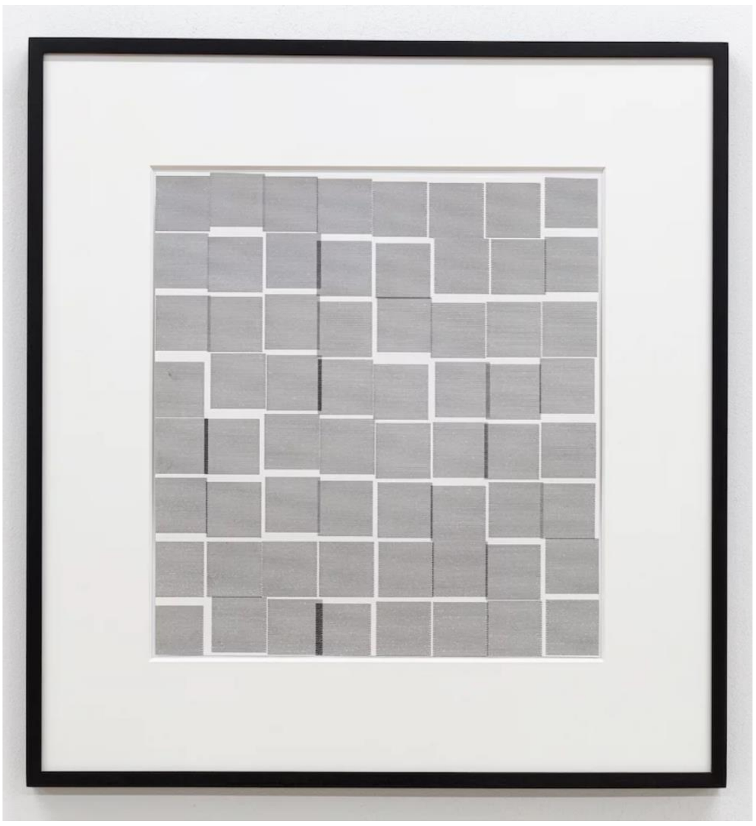


Figure 28. *Vera Molnar, Carrés (1991)*

For this case study, we have selected Vera Molnar's artwork "Carrés" (Figure 28).

By leveraging the principles of Computational Thinking (CT), we can systematically address the problem and devise a solution. Decomposition, Pattern Recognition, Abstraction, and Algorithmic Design principles of CT offer a structured approach to problem-solving and can provide guidance through the process of re-creating the selected algorithmic artwork using P5JS.

Decomposition is the process of breaking down a problem into smaller, more manageable parts. When creating Molnar's artwork, this step involves identifying the individual elements in the composition. During this phase, we concentrate on the formalistic features and observe the artwork's formal aspects without considering the programmatic part. For beginners, it's helpful to meticulously jot down every tiny physical feature they notice in the composition using their preferred medium, such as writing or illustrating on paper or a digital tablet. This approach allows us to focus on individual tasks one at a time, rather than attempting to figure out the entire computer program all at once. Figure 29 depicts our notes on the decomposition step.

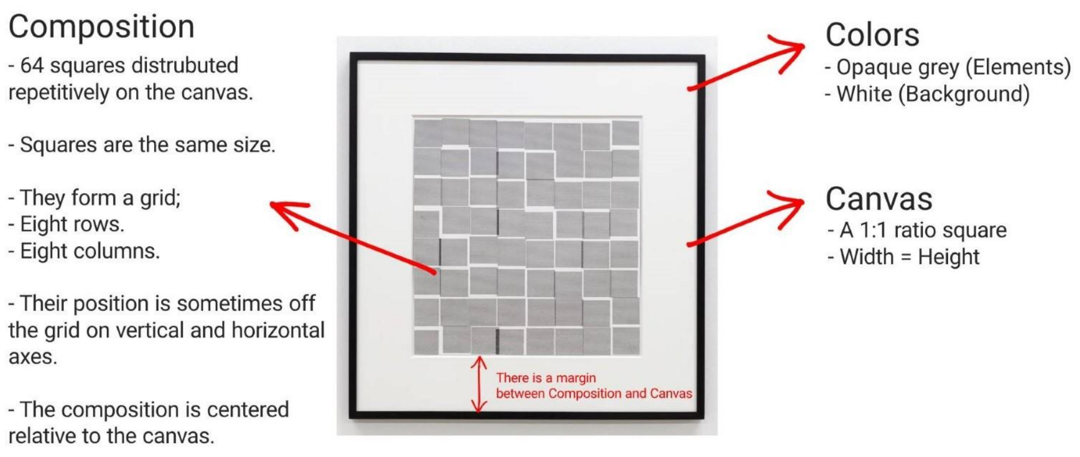

Composition
- 64 squares distrubuted repetitively on the canvas.
- Squares are the same size.
- They form a grid;
- Eight rows.
- Eight columns.
- Their position is sometimes off the grid on vertical and horizontal axes.
- The composition is centered relative to the canvas.
There is a margin between Composition and Canvas
Colors
- Opaque grey (Elements)
- White (Background)
Canvas
- A 1:1 ratio square
- Width = Height

Figure 29: *Notes for Decomposition step*

In this step, we identify the correlations and relationships between different composition parts. For example, all elements in the artwork are copies of the same square. Therefore, we do not need to declare the size for each square individually. Instead, we can define a variable to hold the size of a square and instruct the computer program to use that same value for all squares. Another formalistic feature we can identify is the noticeable vertical distance between each row compared to the horizontal distance between squares. Even if the positions of the squares appear random, the vertical distance varies more than the horizontal distance.

Similarly, we can define another variable to store the color value of each square. The ALAP Categories sheet also acts as a bridge, allowing us to identify programming paradigms observed in the artwork (Figure 30). For instance, the objects appear to be distributed in a grid format resembling an 8x8 matrix, even if they seem randomly positioned on the canvas. We can categorize this as Tiling and start researching relevant sources related to the P5JS programming language. Upon observing the position of each square, we notice they are slightly off their exact position, displaced by a few pixels to the left, right, up, or down. This leads us to identify randomness as a part of the composition but with constraints, such as just a few pixels of variation.

Additionally, the margin between the composition and the frame depends on the size of the squares. The margin from the sides is two times larger than the size of a square. In this step, the possibilities are endless, and the discovery of patterns may vary according to the viewer's experience and level of familiarity with the compositional aspects of an artwork. It does not require specific talent but depends on how one looks at and perceives their environment. We mark down the Translation and Repetition categories because the squares are distributed on the canvas repeatedly. Lastly, we mark layering because the drawing order matters.

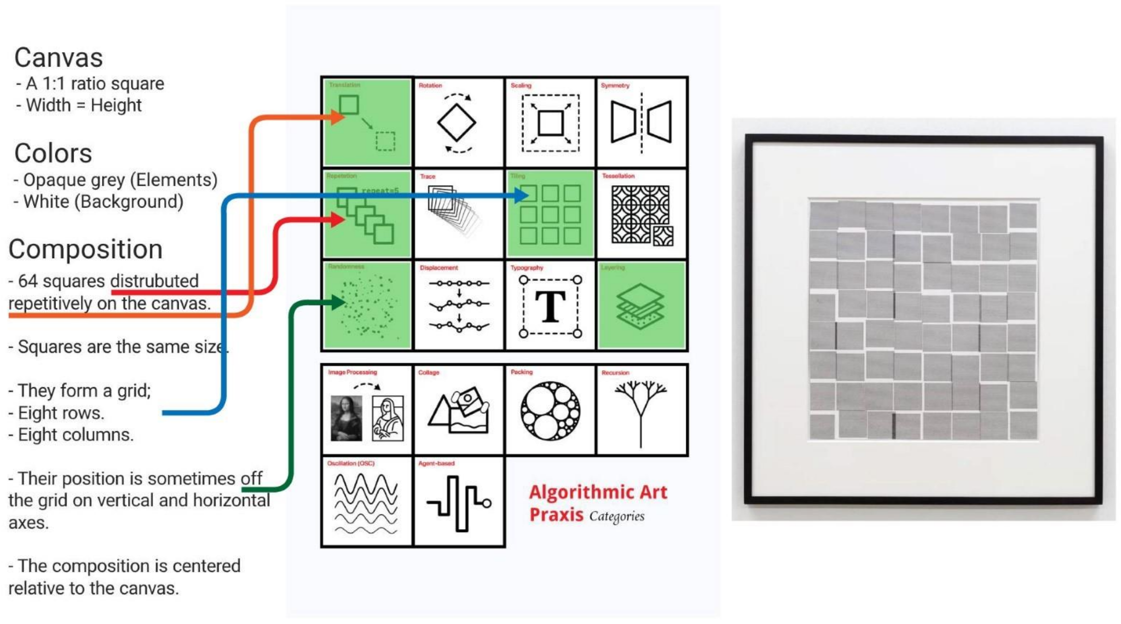
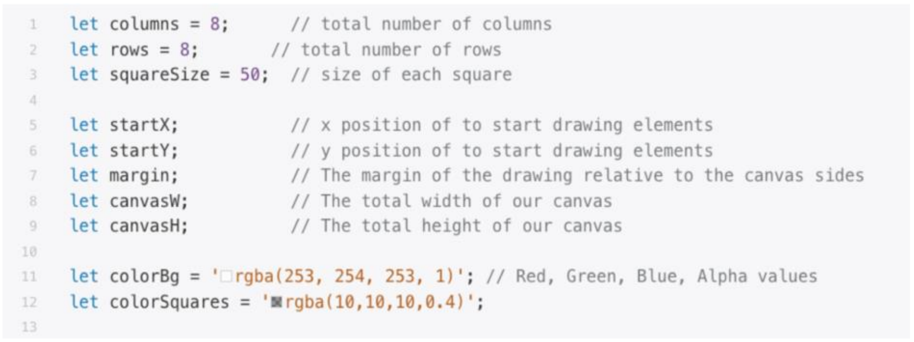


Figure 30: *Abstraction of Programming Practices Identified*

In the abstraction step, it's crucial to simplify complex concepts by concentrating on their essential features based on the capabilities of our computer and programming language, using the ALAP categories sheet as a guide. As shown in Figure 31, we can align the programming paradigms with our notes on the artwork. Instead of using natural language at this stage, we should express our findings in a declarative manner to ensure they are meaningful to the computer. During this phase, we can establish variable names such as "colorBg" for the background color and "colorSquares" for storing square color values. At this stage, we commence coding by declaring variables and assigning values. Figure 31 illustrates ten distinct variables extracted from our previous analysis. The abstraction stage resembles uncovering the meaning of a term in a particular language from a dictionary. The ALAP Categories sheet helps us search for relevant code snippets, online resources, and tutorials.

In the final step, we create a step-by-step procedure to develop the computer program. Figure 32 displays the code with comments indicating its function. Additionally, we organize the code snippets to present the relevant ALAP categories. To achieve Tilling, we employ nested for loops (lines 35,36). Within the inner loop, spanning lines 37 to 44, we initially compute x and y-axis values (lines



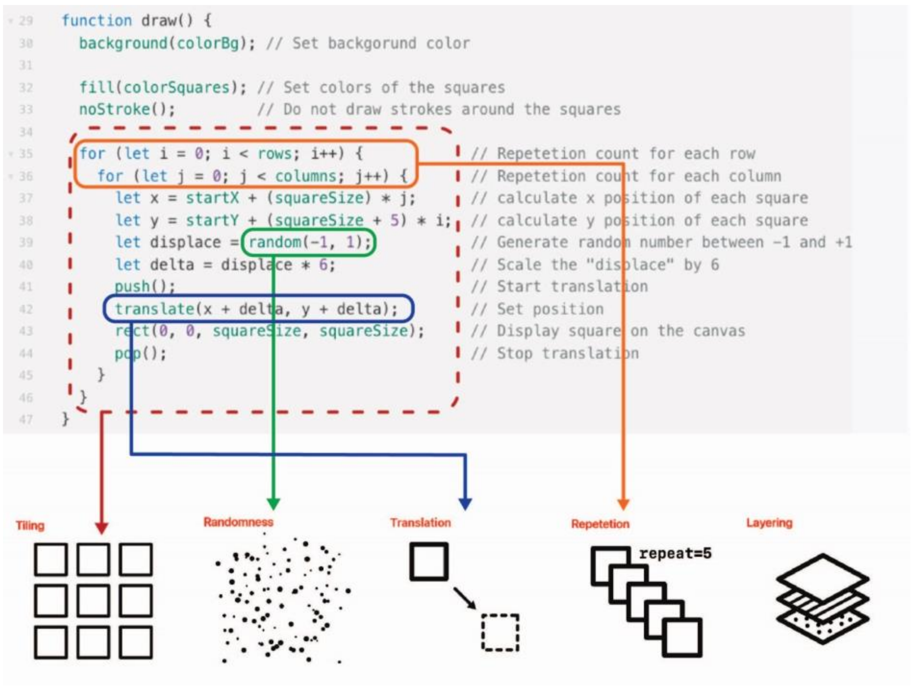
```
let columns = 8;      // total number of columns
let rows = 8;       // total number of rows
let squareSize = 50;  // size of each square

let startX;           // x position of to start drawing elements
let startY;           // y position of to start drawing elements
let margin;           // The margin of the drawing relative to the canvas sides
let canvasW;          // The total width of our canvas
let canvasH;          // The total height of our canvas

let colorBg = '□rgba(253, 254, 253, 1)'; // Red, Green, Blue, Alpha values
let colorSquares = '■rgba(10,10,10,0.4)';

```

Figure 31: *Variable declaration*

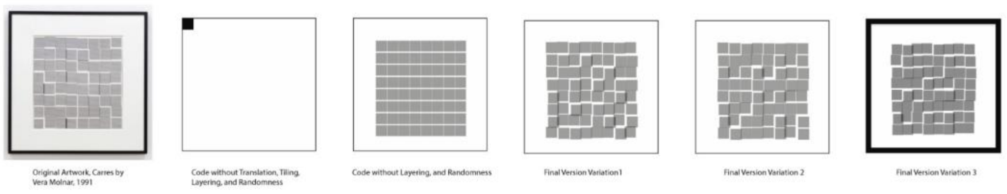


```
29  function draw() {
30    background(colorBg); // Set backgorund color
31
32    fill(colorSquares); // Set colors of the squares
33    noStroke();         // Do not draw strokes around the squares
34
35    for (let i = 0; i < rows; i++) {                 // Repetetion count for each row
36      for (let j = 0; j < columns; j++) {            // Repetetion count for each column
37        let x = startX + (squareSize) * j;           // calculate x position of each square
38        let y = startY + (squareSize + 5) * i;       // calculate y position of each square
39        let displace = random(-1, 1);                // Generate random number between -1 and +1
40        let delta = displace * 6;                    // Scale the "displace" by 6
41        push();                                      // Start translation
42        translate(x + delta, y + delta);             // Set position
43        rect(0, 0, squareSize, squareSize);          // Display square on the canvas
44        pop();                                       // Stop translation
45      }
46    }
47  }
```

Figure 32: *Algorithm Design*

37, 38) for each repetition to exhibit 64 squares on the canvas in a grid format. We introduce random displacement to each square by generating a random value within the range of -1 to +1 (line 39), which is then multiplied by six (line 40) to confine off-grid positions between -6 and +6. To position the squares, we utilize the translate function in each iteration (line 42) and incorporate the variable delta (line 40) to disperse the squares randomly.

The program utilizes random number generators to produce various iterations of the same concept without the need for manual editing. Figure 33 displays chosen outputs generated by the code. The progression of the artwork creation is visualized in a step-by-step manner, from left to right.



Original Artwork, Carres by Vera Molnar, 1991
Code without Translation, Tiling, Layering, and Randomness
Code without Layering, and Randomness
Final Version Variation1
Final Version Variation 2
Final Version Variation 3

Figure 33: *Result of the finalized program*

4. Conclusion

In this article, we propose the Algorithmic Art Praxis (ALAP) Framework as an output of our study, which was created through extensive research performed on secondary data resources, analyzing 2000 images and resulting in 695 entries to provide tools and learning material for computational thinking and contextualized programming education. While some entries have limited data, others provide detailed content. In conjunction with computer programming paradigms, the analysis has proposed 18 categories related to programming practices used in algorithmic art. The proposed framework consists of three main components that support contextualized programming education and serve as a tool for computational thinking and problem-solving methods. By proposing ALAP Categories, Cheat Sheet, and a public online database, we provide hands-on learning and teaching material that has the potential to grow and improve over time with the contribution of other researchers. The categories serve as semiotically defined constructs aimed at helping novices analyze artworks and bridge the gap between artistic expression and programming concepts.

The process of collating information from disparate sources to ascertain the most accurate data was challenging at times. In particular, gathering data from disparate and constrained sources proved to be a highly time-consuming endeavor. Although the present study is concerned with the formal aspects of created artworks, we have assumed the responsibility of providing the most accurate information about the artwork for the academic community. In addition to our framework, we aim to provide a unique resource material for teaching and learning programming, which we have previously argued is one of the missing tools for contextualized programming education. At the same time, our online database is intended to be a valuable resource for other researchers working on the topic in future studies who are looking for such information in one place rather than searching multiple web resources.

Following our research and analysis results, we demonstrated a case study on how others can integrate the ALAP framework as a helper tool with Computational Thinking principles in their creative coding classes. Our framework should be applied in introductory programming classes after the students are taught fundamental programming concepts. Then, using the ALAP categories, each Praxis category should be explained and demonstrated with various examples via our online ALAP database. During the lectures, students can also review the online database from their personal computers, and they can keep the ALAP categories and cheat sheet in printed format for in-class practices to benefit from the framework. Using these tools allows the learner to match similar formal behavior on the artwork with a relevant programming practice used in creative coding. So, the learner can research different sources or relevant book chapters using the keywords. For our case study, we suggest checking the P5JS forum[5], documentation[6], and the books (Table 2) related to P5JS for beginners and introductory programming classes.

By providing a structured approach that links visual elements to programming paradigms, ALAP offers valuable tools for researchers, educators, and students in digital art and computer science looking for concrete resource material and tools contextualized within Algorithmic works of art.

Finally, future research should focus on empirically testing the effectiveness of the ALAP framework in educational settings and exploring its potential applications in other domains where computational thinking intersects with creative practices.

Notes:

1. Online ALAP Database web site (https://tinyurl.com/AlgorithmicArtPraxis).
2. The Internet Archive is a non-profit organization that operates as a digital library for Internet sites and other cultural artifacts. Its aim is to provide universal access to all knowledge, offering free access to researchers, historians, scholars, individuals with print disabilities, and the general public. It functions like a traditional library, but in digital form, and is committed to preserving and providing access to the world's cultural heritage for future generations.
3. Notion web site (https://www.notion.so).
4. ALAP Online Database tutorial videos (https://www.youtube.com/watch?v=UWWDdKc2xko).

5. Processing Forum web site (https://discourse.processing.org/).
6. P5JS Documentation web site (https://p5js.org/reference/), examples (https://p5js.org/examples/) and tutorials (https://p5js.org/tutorials/).

Acknowledgment: N/A
Conflict of Interest: The authors stated that there are no conflicts of interest regarding the publication of this article.
Ethics Committee Approval: N/A
Author Contributions: The authors confirm sole responsibility for the following: study conception and design, data collection, analysis and interpretation of results, and manuscript preparation.
Financial Disclosure: The authors declared that this study has received no financial support.

References

Allwood, C. M. (1986). Novices on the computer: A review of the literature. *International Journal of Man-Machine Studies*, 25(6), 633–658. https://doi.org/10.1016/S0020-7373(86)80079-7

Bailey, J. (2018). *Why Love Generative Art*?. notre traduction, juillet.

Boden, M. A., & Edmonds, E. A. (2009). What is generative art? *Digital Creativity*, 20(1–2), 21–46. https://doi.org/10.1080/14626260902867915

Brown, N. C. C., & Wilson, G. (2018). Ten quick tips for teaching programming. *PLoS Computational Biology*, 14(4), e1006023. https://doi.org/10.1371/journal.pcbi.1006023

Bryant, R., Weiss, R., Orr, G., & Yerion, K. (2011). Using the context of algorithmic art to change attitudes in introductory programming. *Journal of Computing Sciences in Colleges*, 27(1), 112–119. http://dl.acm.org/citation.cfm?id=2037177

Berkeley, E. (1963). *Computer Art Contest. Computers and Automation*, XII (1), p. 21.

Dorin, A., McCabe, J., McCormack, J., Monro, G., & Whitelaw, M. (2012). A framework for understanding generative art. *Digital Creativity*, 23(3–4), 239–259. https://doi.org/10.1080/14626268.2012.709940

Farah, J. C., Moro, A., Bergram, K., Purohit, A. K., Gillet, D., & Holzer, A. (2020). Bringing computational thinking to non-STEM undergraduates through an integrated notebook application. *European Conference on Technology Enhanced Learning*, 2676. http://ceur-ws.org/Vol-2676/paper2.pdf

Galanter, P. (2016). Generative Art Theory. In *A Companion to Digital Art*, C. Paul (Ed.). https://doi.org/10.1002/9781118475249.ch5

Guo, P. J. (2017). Older Adults Learning Computer Programming: Motivations, Frustrations, and Design Opportunities. *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, 7070–7083. https://doi.org/10.1145/3025453.3025945

Guzdial, M. (2006). Teaching computing for everyone. *Journal of Computing Sciences in Colleges*, 21(4), 6. http://dl.acm.org/ft_gateway.cfm?id=1127390&type=pdf

Guzdial, M. (2010). Does contextualized computing education help? *ACM Inroads*, 1(4), 4–6. https://doi.org/10.1145/1869746.1869747

Hansen, S. M. (2019). Mapping Creative Coding Courses: Toward Bespoke Programming Curricula in Graphic Design Education. Eurographics 2019 - Education Papers, 4 pages. https://doi.org/10.2312/EGED.20191024

Kelleher, C., & Pausch, R. (2005). Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Computing Surveys*, 37(2), 83–137. https://doi.org/10.1145/1089733.1089734

Liao, L., & Pope, J. W. (2008). Computer literacy for everyone. *Journal of Computing Sciences in Colleges*, 23(6), 231–238. https://doi.org/10.5555/1352383.1352423

Lohiniva, M., & Isomöttönen, V. (2021). Novice Programming Students' Reflections on Study Motivation during COVID-19 Pandemic. 2021 *IEEE Frontiers in Education Conference* (FIE), 1-9.

Luxton-Reilly, A. (2016). Learning to Program is Easy. *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*, 284–289. https://doi.org/10.1145/2899415.2899432

Macdonald, N. (1981). *Computers and People*—Vol XXX - 7-8. 30(7–8).

McCarthy, L., Reas, C., & Fry, B. (2015). *Getting Started with p5.js: Making Interactive Graphics in JavaScript and Processing*. Maker Media, Inc.

Medeiros, R. P., Ramalho, G. L., & Falcao, T. P. (2019). A Systematic Literature Review on Teaching and Learning Introductory Programming in Higher Education. *IEEE Transactions on Education*, 62(2), 77–90. https://doi.org/10.1109/TE.2018.2864133

Mollu, M. (2020). Computational thinking as a problem-solving framework for the 21st century. *International Journal of Educational Technology and Society*, 23(2), 305–315.

Noble, J. J. (2009). *Programming interactivity: A designer's guide to processing, Arduino, and openFrameworks* (1st ed). O'Reilly.

Papert, S. (1980). Mindstorms: *Children, computers, and powerful ideas*. Basic Books.

Pearson, M. (2011). *Generative art: A practical guide using processing*. Manning; Pearson Education.

Perevalov, D., & Tatarnikov, I. (2013). *Mastering openFrameworks: Creative coding demystified: a practical guide to creating audiovisual interactive projects with low-level data processing using openFrameworks*. Packt Publishing.

Perevalov, D., & Tatarnikov, I. (2015). *openFrameworks Essentials: Create stunning, interactive openFrameworks-based applications with this fast-paced guide.* Packt Publishing.

Phon-Amnuaisuk, S., & Panjapornpon, J. (2012). Controlling Generative Processes of Generative Art Somnuk Phon-. *Procedia Computer Science*, 13, 43–52. https://doi.org/10.1016/j.procs.2012.09.112

Polanyi, M., & Sen, A. (2009). *The Tacit Dimension*. University of Chicago Press.

Reas, C., & Fry, B. (2007). *Processing: A programming handbook for visual designers and artists*. MIT Press.

Ring, B. A., Giordan, J., & Ransbottom, J. S. (2008). Problem Solving Through Programming: Motivating the Non-Programmer. *Consortium for Computing Sciences in Colleges*, 23(3), 7.

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and Teaching Programming: A Review and Discussion. *Computer Science Education*, 13(2), 137–172. https://doi.org/10.1076/csed.13.2.137.14200

Romero, M., Lepage, A., & Lille, B. (2017). Computational thinking development through creative programming in higher education. International *Journal of Educational Technology in Higher Education*, 14(1), 42. https://doi.org/10.1186/s41239-017-0080-z

Shein, E. (2014). Should Everybody Learn to Code? *Association for Computing Machinery*, 57(2), 16–18. https://doi.org/10.1145/2557447

Shiffman, D. (2008). *Learning Processing: A beginner's guide to programming images, animation, and interaction*. Morgan Kaufmann/Elsevier.

Shiffman, D. (2012). *The Nature of Code*. Nature of Code.

Wing, J. M. (2008). Computational thinking and thinking about computing. Philosophical Transactions of the Royal Society A: Mathematical, *Physical and Engineering Sciences*, 366(1881), 3717–3725. https://doi.org/10.1098/rsta.2008.0118

Winslow, L. E. (1996). Programming pedagogy—a psychological overview. *ACM SIGCSE Bulletin*, 28(3), 17–22. https://doi.org/10.1145/234867.234872

Terzidis, K. (2009). *Algorithms for visual design using the processing language*. Wiley Pub.

Torrence, B. (2021). Tessellations: Mathematics, Art, and Recreation. *American Mathematical Monthly*, 128(10), 955–959. https://doi.org/10.1080/00029890.2021.1971917

Yadav, A., Stephenson, C., & Hong, H. (2017). Computational thinking for teacher education. *Communications of the ACM*, 60(4), 55–62. https://doi.org/10.1145/2994591

Yardi, S., & Bruckman, A. (2007). What is computing? Bridging the gap between teenagers' perceptions and graduate students' experiences. *Proceedings of the Third International Workshop on Computing Education Research - ICER '07*, 39. https://doi.org/10.1145/1288580.1288586