

Arte e código: Experimentos com Arte Generativa Computacional

São Paulo, 2023
Universidade de São Paulo
Beatriz Costa do Nascimento
Orientadora: Prof. Dra. Sílvia Laurentiz

Arte e código: Experimentos com Arte Generativa Computacional

Beatriz Costa do Nascimento

Orientadora: Prof. Dra. Silvia Laurentiz

Trabalho de Conclusão de Curso
apresentado ao Departamento de
Artes Plásticas da Escola de
Comunicações e
Artes da Universidade de São Paulo
para obtenção do título de Bacharel
em Artes Visuais. São Paulo, 2023.



Agradecimentos

Aos meus pais Analu e Tony, pelo apoio, encorajamento e sacrifícios feitos ao longo de muitos anos para que meu sucesso fosse possível; por aceitar minha individualidade e me incentivar com o amor mais puro que já vi. Aos meus irmãos, Guilherme e Gustavo por sempre estarem disponíveis para qualquer tipo de conversa e desabafo.

Às minhas amigas de infância, Laura e Gabi, que estiveram presentes nos momentos mais importantes do meu desenvolvimento como artista e pessoa. Pelo incentivo diante minhas dúvidas e vibração nos meus sucessos.

Elio, Niq e outros colegas da graduação pela companhia nos perrengues, apoio emocional e experiências que só quem passou, sabe.

À minha orientadora, Prof. Dra. Silvia Laurentiz, pela inspiração como artista e conhecimento passado como educadora. Pelas broncas, puxões de orelha, paciência e sensibilidade no processo de criação deste trabalho. Aos outros professores da graduação que participaram direta ou indiretamente da minha formação.

Resumo

A arte generativa computacional é um tipo de arte que ficou muito popular com o avanço das tecnologias. Seu método de criação trouxe questões sobre originalidade, criatividade e autoria do resultado. Neste trabalho, a principal questão a ser explorada é a influência do artista no processo criativo usando ferramentas computacionais.

Serão feitos experimentos para tentar entender qual é a mudança no resultado estético e conceitual quando existe menos ou mais controle do criador sobre o programa.





Sumário

Introdução	05
1. Precedentes da Arte Generativa	07
1.1 Afinal, o que é generativo?	13
1.2 Arte Generativa hoje e os NFTs	17
2. Estudos de Caso	19
2.1 Vera Bighetti	21
2.2 Monica Rizzolli	23
2.3 Suzete Venturelli	25
3. Processo de criação	27
3.1 Desenvolvimento	29
3.2 O uso do ChatGPT	49
4. Resultados	54
4.1 Lúcido	56
4.2 Cerrado	67
Considerações Finais	77
Referências	80
Apêndices	81

INTRODUÇÃO

A arte generativa está muito presente no ambiente criativo tecnológico atual e possui uma definição ampla e aberta, incluindo uma visão de que esse tipo de arte existia mesmo antes do uso dos computadores. A criação generativa envolve um sistema com certa independência, podendo essa autonomia sistêmica ser controlada pelo artista ou não. Consequentemente, pode-se perguntar: se a arte generativa requer autonomia, de que forma o artista abre mão dessa decisão? Qual é o nível de autonomia necessário para que o trabalho seja considerado generativo? Como a redução ou aumento do controle do artista interfere no resultado estético e conceitual do trabalho? A partir dessas questões, surgiu a proposta de desenvolver um projeto que envolvesse duas partes de tamanho e relevância semelhantes: a criação do artista e a autonomia do código.

Em nível pessoal, o tema do trabalho não foi explorado de forma extensa na graduação, destacando assim sua importância e motivo de curiosidade. Entre estudantes de artes plásticas, colegas de curso em artes visuais, há certa resistência em relação à programação - um grupo no qual a própria autora se incluía -, o que torna o trabalho relevante para a 'desmistificação' desse modo criativo.

No âmbito universitário, o desenvolvimento desta pesquisa demonstra a característica interdisciplinar da arte generativa computacional e a possibilidade de colaboração entre artistas, designers, programadores, cientistas da computação, engenheiros, entre outros.

O primeiro capítulo será responsável pela contextualização da arte generativa, detalhando o início do uso do termo para se referir à criação com computador. Em seguida, serão apresentadas duas definições com exemplos de obras e artistas. Por fim, será discutido o uso da arte generativa computacional nos NFTs, uma abordagem adotada por diversos artistas quando se trata de divulgação e comercialização de suas criações.

No segundo capítulo, serão discutidas referências estéticas com o objetivo de orientar os experimentos que serão realizados para a criação de trabalho experimental. Aqui serão mencionadas as artistas Vera Bighetti, Monica Rizzolli e Suzete Venturelli.

O terceiro capítulo abordará a descrição do processo criativo, desde o aprendizado básico sobre programação até a construção do código que gerará as imagens das duas séries desenvolvidas: Lúcido e Cerrado. A descrição também incluirá detalhes sobre as tomadas de decisão, seus motivos e as ferramentas utilizadas para a criação, destacando-se os tutoriais encontrados em diversas plataformas e a ferramenta de inteligência artificial ChatGPT.

O quarto capítulo apresentará as 20 imagens finais, sendo 10 de cada série, e discutirá o percurso seguido a partir da conclusão do código. Por fim, serão apresentadas as considerações finais sobre o trabalho, com o intuito de responder às questões iniciais colocadas nesta introdução.

An abstract graphic on the left side of the page. It features a series of wavy, concentric black lines that create a sense of depth and movement, resembling a stylized wave or a topographical map. In the upper left corner, there is a bright yellow-green starburst or cross-like shape with a soft, blurred glow.

1. PRECEDENTES DA ARTE GENERATIVA

A arte computacional teve seu início nos anos 50 (BODEN e EDMONDS, 2009) e a generativa pode ser considerada parte dos sistemas nascidos nessa época. Com essa forma de criação, foi possível estabelecer uma colaboração entre o artista e a tecnologia do computador. O avanço da computação permitiu que essa forma de criar trouxesse mais complexidade para os resultados, ao passo que a relação entre o artista e o processo de criação também foi se transformando.

Existem questionamentos sobre originalidade, criatividade e a autoria do resultado, já que nesse tipo de criação se permite que o artista delegue parte do controle do processo generativo a uma máquina, sistema, ou outra invenção processual. Neste trabalho, a principal questão a ser desenvolvida é a de controle sobre o processo e o resultado da arte generativa. A partir de experimentações, serão exploradas as possibilidades de resultados a partir de tomada de decisões iniciais e o que acontece quando as decisões finais são cedidas ao código generativo.

Para investigar essa dinâmica, primeiramente, há de se definir o que é a arte generativa, uma vez que não existe um consenso, como é discutido pelo autor Philip Galanter na entrevista “A arte generativa é tão antiga quanto a própria arte”¹ (2004). A definição da arte generativa é ampla e muito discutida atualmente devido ao fácil acesso e popularização de ferramentas de criação, sua relação direta com a arte computacional e mesmo com os NFTs, que cresceu nas últimas décadas.

Apesar disso, conseguimos encontrar obras feitas há mais de 60 anos que são consideradas generativas e não dependem necessariamente do uso do computador.

Margareth Boden e Ernests Edmonds consideram que os primeiros trabalhos generativos surgem no final dos anos 50. Entretanto, um marco deste termo e em toda arte que utiliza tecnologia para seu funcionamento foi a exposição “Generative Computergraphik” feita em 1965 com curadoria de Max Bense e Siegfried Maser. George Nees publicou uma tese de doutorado onde definiu seus trabalhos, que estavam naquela exposição, como “generativos”, por exemplo, o da Figura 1, que foi feito através de uma plotadora guiada por códigos. No mesmo ano, em uma exposição chamada “Computer Graphik” (Druid, 2021) Nees expôs trabalhos com o artista Frieder Nake que também considerava suas obras generativas. Nesse momento, o termo passa a ser utilizado como uma forma de indicar que foi produzido por meio de uma máquina e em parte, de forma automática (BODEN e EDMONDS, 2009).

¹Tradução livre do original em inglês: “Generative art is as old as art”

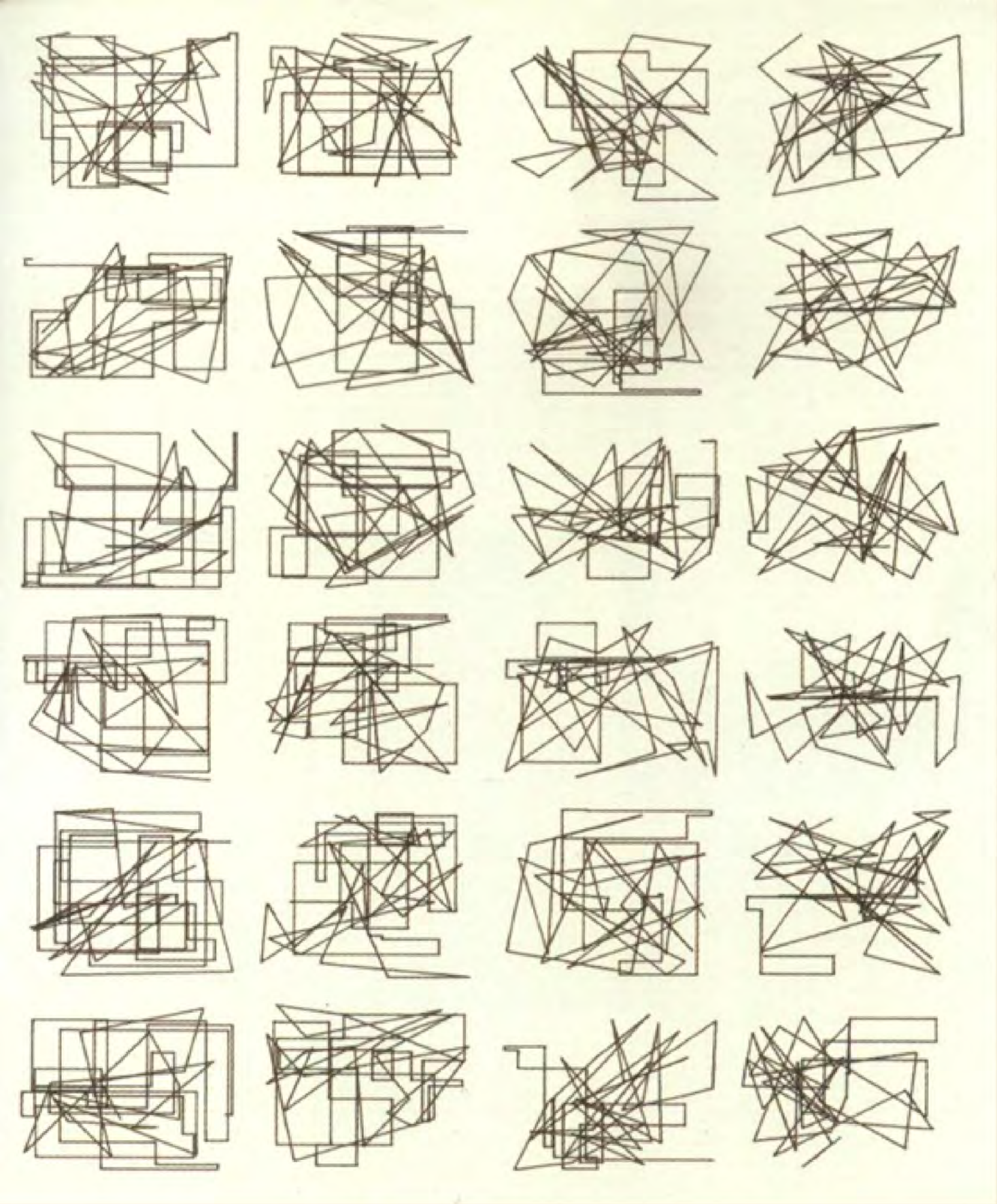


Figura 1 - Sem título,
George Nees, 1965.
Imagem criada por
plotadora guiada por
códigos.
<<https://proyectoidis.org/george-nees/>>
Acesso em julho de
2023.

Um grande destaque na arte computacional é a exposição feita por Jasia Reichardt em 1968, *Cybernetic Serendipity*. Apesar de não conter trabalhos necessariamente chamados de “generativos”, os métodos futuramente utilizados para criar arte generativa com auxílio da tecnologia já estavam presentes, como o trabalho *Colloquy of Mobiles* de Gordon Pask (Figura 2).

Esta instalação foi programada para reagir aos movimentos no ambiente mudando a posição dos móveis. A exposição também contou com uma variedade de obras divididas em três categorias: animações, imagens e músicas geradas por computador; objetos e ambientes cibernéticos, robôs e máquinas que faziam pinturas; e máquinas demonstrando o uso do computador (Reichardt, 1968).



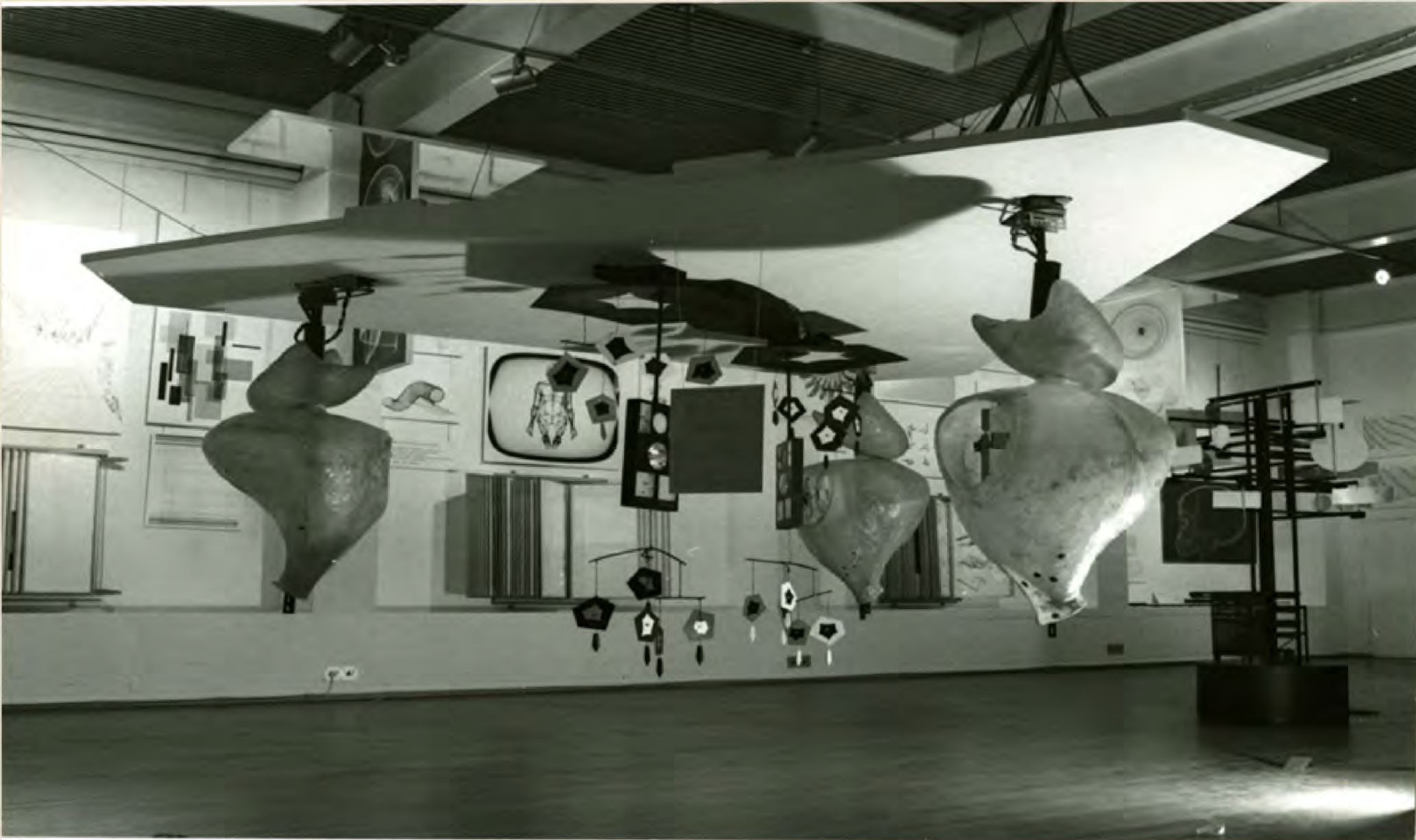


Figura 2 - Colloquy of mobiles, Gordon Pask, 1968. Instalação. <<https://frontiers-of-solitude.org/blog/434>> Acesso em julho de 2023.

O termo *cybernetic* se refere a uma definição feita por Norbert Wiener em seu livro “Cibernética: ou controle e comunicação no animal e na máquina” (1970). Conforme Catherine Mason, Wiener define cybernetics como o estudo do comportamento da máquina e de sistemas sociais e biológicos que serve como base para a produção artística na exposição e seu questionamento em relação às novas tecnologias e seus impactos na vida humana

.No catálogo que acompanha a exposição, “*Cybernetic Serendipity: The Computer and the Arts*” (1968), Reichardt diz que a revelação mais importante da exposição é que os computadores deixaram de ser utilizados apenas como ferramenta prática - matemática - e passaram a permitir soluções artísticas. Assim, nas palavras de Reichardt, mesmo pessoas que nunca desenharam e pintaram conseguiam criar imagens que se aproximavam aos trabalhos artísticos mostrados nas galerias tradicionais. Esse pensamento é utilizado até hoje na criação de arte generativa. A Figura 3 está presente em um vídeo introdutório sobre a exposição que conta com obras de diversos artistas, e é um exemplo do uso da tecnologia para criação na época.

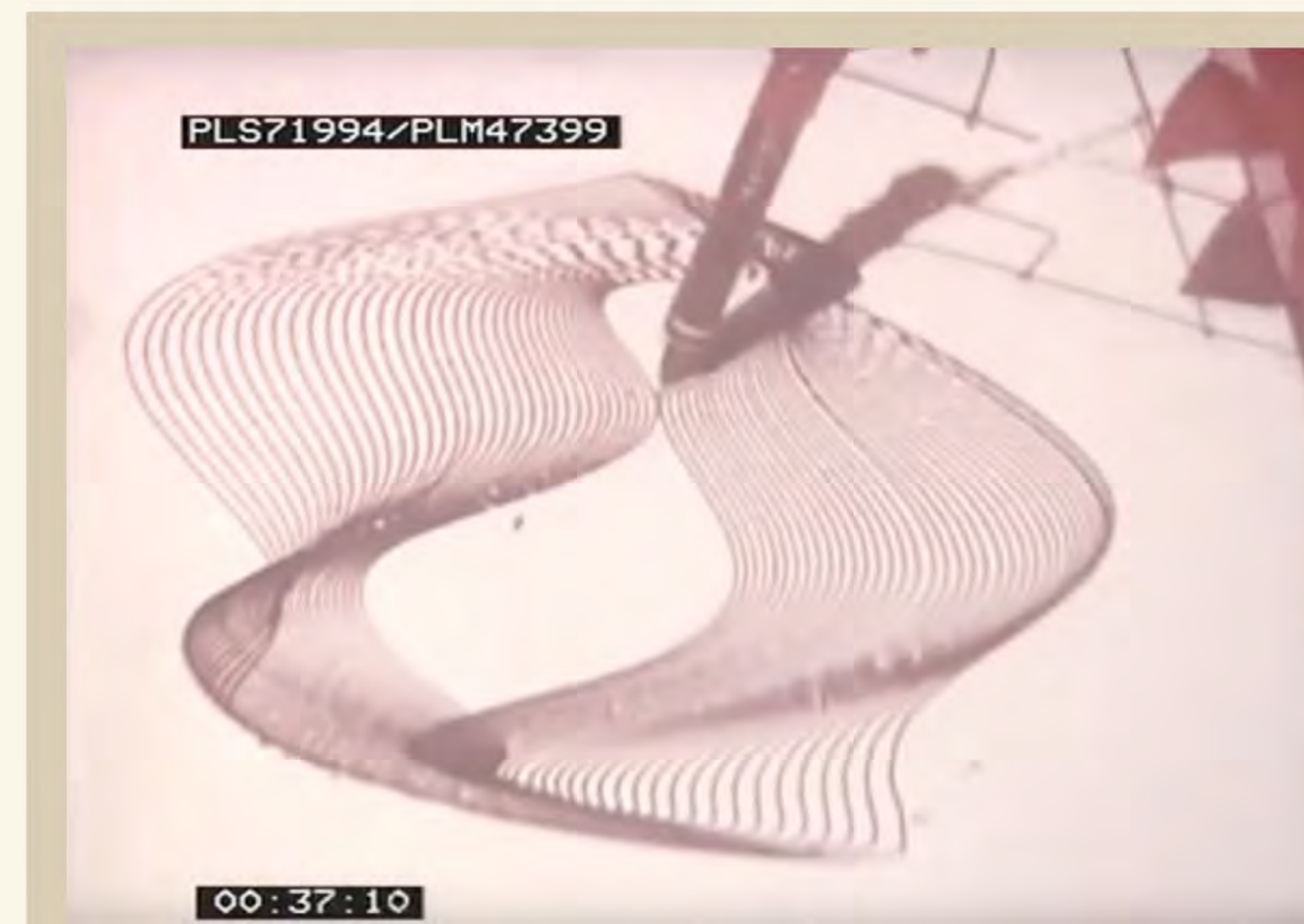


Figura 3 - Drawing Machine, John Ravillious. Presente na exposição Cybernetic Serendipity (ICA) - Late Night Lineup (1968). Captura de tela do vídeo (00:37) <<https://www.youtube.com/watch?v=n8TJx8n9UsA>> Acesso em julho de 2023.

A artista Vera Molnar é conhecida por seu pioneirismo na arte computacional e generativa desde os anos 60. Ela cofundou vários grupos de pesquisa de artistas pioneiros, como GRAV (*Groupe de Recherche d'Art Visuel*), que investiga abordagens colaborativas para arte mecânica e cinética, e o grupo de pesquisa para arte e ciência da computação no Instituto de Arte e Ciência de Paris.

Antes do computador, Molnar utilizava algoritmos ou *machine imaginaire* (máquina imaginária/imaginativa) que permitiam a criação de séries de imagens seguindo um conjunto de regras pré-ordenadas. A partir de 1968, o computador tornou-se um dispositivo central na criação de suas pinturas e desenhos, nos quais eram abordadas variações de formas e linhas.

Os trabalhos de Vera envolvem, muitas vezes, formas simples como linhas ou quadrados. Em Hipertransformações (Figura 4) a artista trocou os parâmetros no código da plotadora, mudando direções no eixo x e y, causando a distorção dos quadrados cada vez que eles eram desenhados. O uso da geometria simples demonstra como diversos resultados podem ser alcançados com mudanças pequenas na programação.

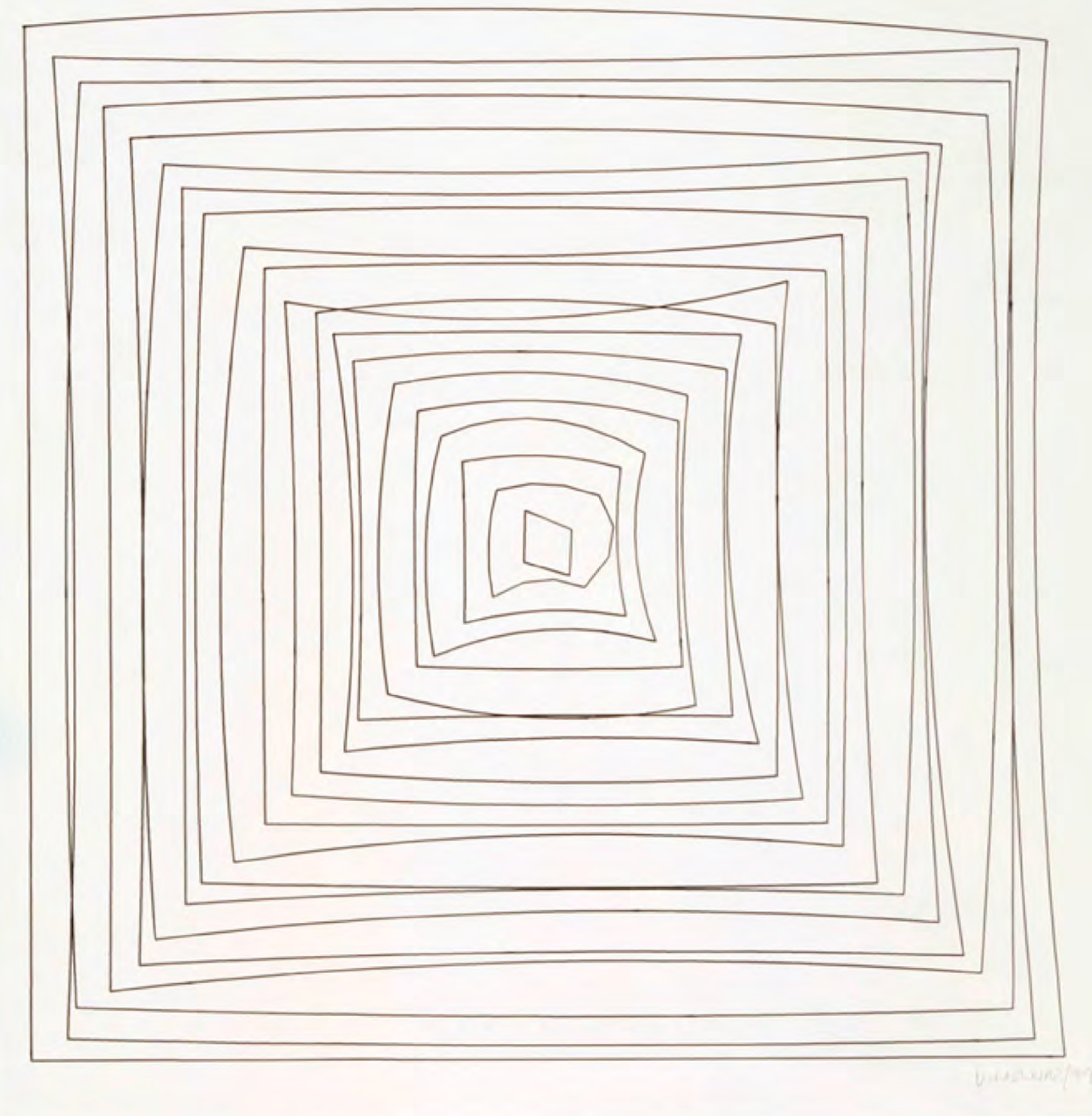


Figura 4 - Hipertransformações, Vera Molnar, 1975-76. Desenhos com plotter. 20x20cm <https://dam.org/museum/artists_ui/artists/molnar-vera/> Acesso em julho de 2023.

Também na arte computacional podemos citar Harold Cohen, que criou a máquina-artista Aaron, um projeto iniciado em 1973. Aaron é uma máquina guiada por códigos que no início apenas desenhava formas no papel. Muitos desenhos eram completados por Cohen através da pintura, até que o programa Aaron avançou o suficiente para fazer isso sozinho através de parâmetros previamente colocados. O trabalho da Figura 5 é um exemplo de uma criação mais avançada de Aaron, em 2003. É importante ressaltar que Cohen nunca utilizou o termo “generativo” para caracterizar seu trabalho. Apesar disso, Aaron é citado como um marco importante para esse pensamento (GARCIA, 2016).



Figura 5 - Harold Cohen, 030504, 2003. Imagem gerada pelo programa AARON através de plotadora, 75 x 50 cm. <<https://www.studiointernational.com/delving-into-coding-the-art-of-harold-cohen-aaron-computer-generated>> Acesso em julho de 2023.

1.1 AFINAL, O QUE É GENERATIVO?

Margaret Boden e Ernest Edmonds no artigo “*WHAT IS GENERATIVE ART?*” (2009) definem que o trabalho generativo “é feito, ao menos parcialmente, através de um processo que não está no controle direto do artista”². Os autores reconhecem que a arte generativa não é apenas a arte computadorizada ao mesmo tempo que apresentam diversas definições (como, por exemplo, de “*computer art*”, “*process art*”, “*computer generated art*”, etc.) que podem ser utilizadas como sinônimos. Ao comentar sobre a definição dada à arte generativa, citam:

Esta é uma definição muito ampla. Não especifica o tamanho mínimo da parte que ficará fora do controle. Não estabelece exatamente que tipo de processo generativo está em questão. Não diz o que conta como fora do controle direto do artista. E é omissa sobre a extensão (se houver) de quais os processos em questão podem ter sido deliberadamente moldados pelo artista antes de 'perder' o controle de fato. Em suma, nossa definição de G-art é bastante intuitiva. Em geral, ela seleciona casos de arte em que o controle pessoal é deliberadamente diminuído, ou mesmo totalmente abandonado, e processos relativamente impessoais assumem o controle.³ (BODEN E EDMONDS, 2009, pg. 10)

A obra Interrupções (Figura 6) de Vera Molnar é um exemplo dessa definição dada por Boden e Edmonds. A artista estabeleceu os parâmetros iniciais, que seriam as linhas com rotações possíveis e as interrupções. Porém, a 'decisão' de qual nível de rotação e o local das interrupções foi feita de forma randômica, ou seja, fora do controle da artista.



²Tradução livre do original em inglês: “G-art works are generated, at least in part, by some process that is not under the artist’s direct control.”

³Tradução livre do original em inglês: “This is a very broad definition. It does not specify the minimal size of the “part”. It does not lay down just what sort of generative process is in question. It does not say what counts as being outside the artist’s direct control. And it is silent on the extent (if any) to which the processes concerned may have been deliberately moulded by the artist before ‘losing’ direct control. In short, our definition of G-art is largely intuitive. In general, it picks out cases of art-making in which personal control is deliberately diminished, or even wholly relinquished, and relatively impersonal processes take over.”

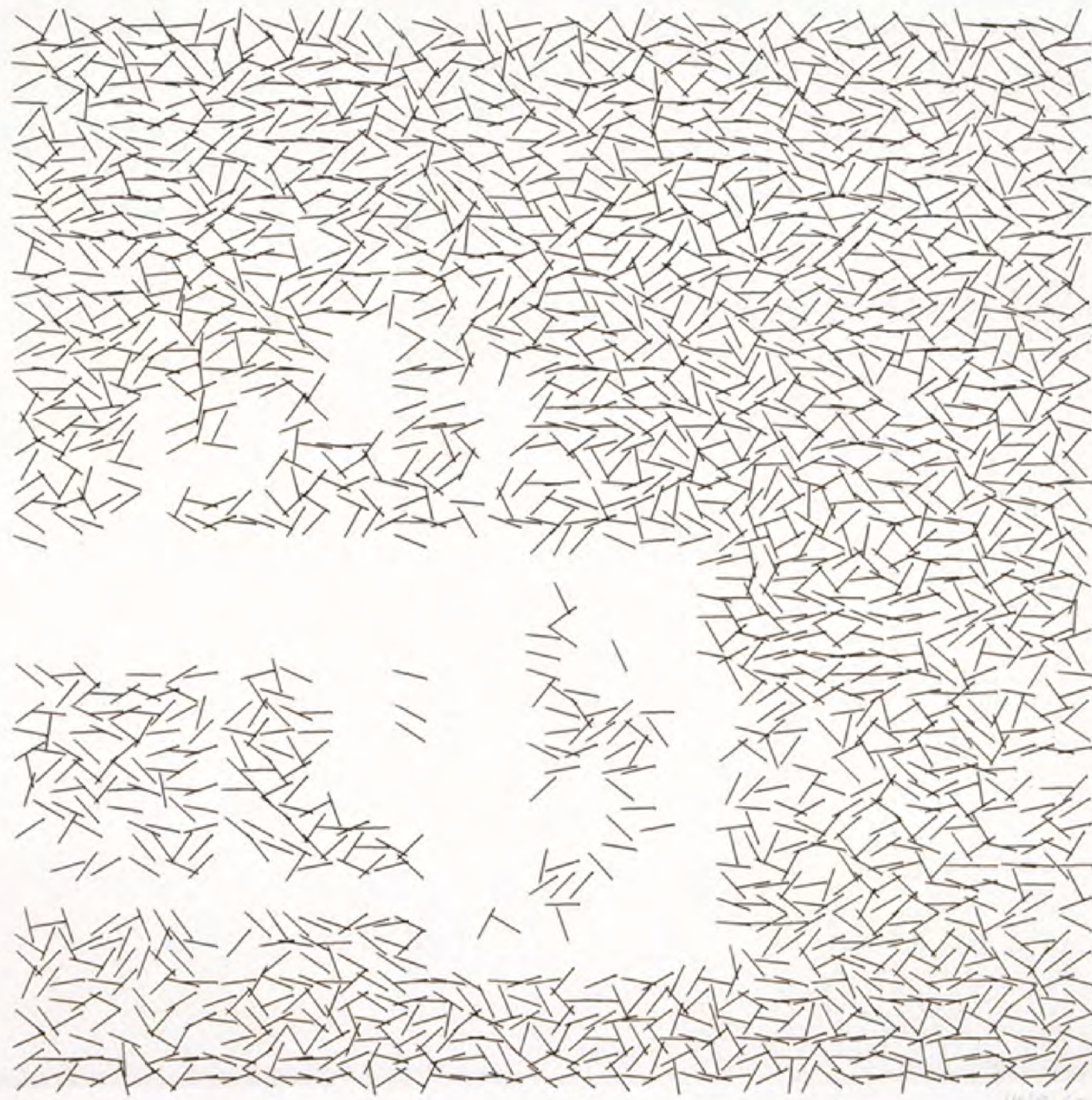


Figura 6 - Interrupções, 1968-69. Imagem gerada de forma randômica.
<https://dam.org/museum/artists_ui/artists/molnar-vera/> Acesso em julho de 2023.

Para Philip Galanter, a arte generativa é definida como um subconjunto da arte, no qual é possível produzir múltiplos resultados utilizando algum tipo de sistema generativo. Além de pesquisador, Galanter é artista e também cria trabalhos generativos, como pode ser visto na Figura 7, Sem título (f091148).

“Arte generativa refere-se a qualquer prática de arte em que o artista usa um sistema, como um conjunto de regras de linguagem natural, um programa de computador, uma máquina ou outra invenção processual, que é acionada com algum grau de autonomia contribuindo ou resultando em uma obra de arte concluída.” ⁴ (GALANTER, 2003, pg 4).

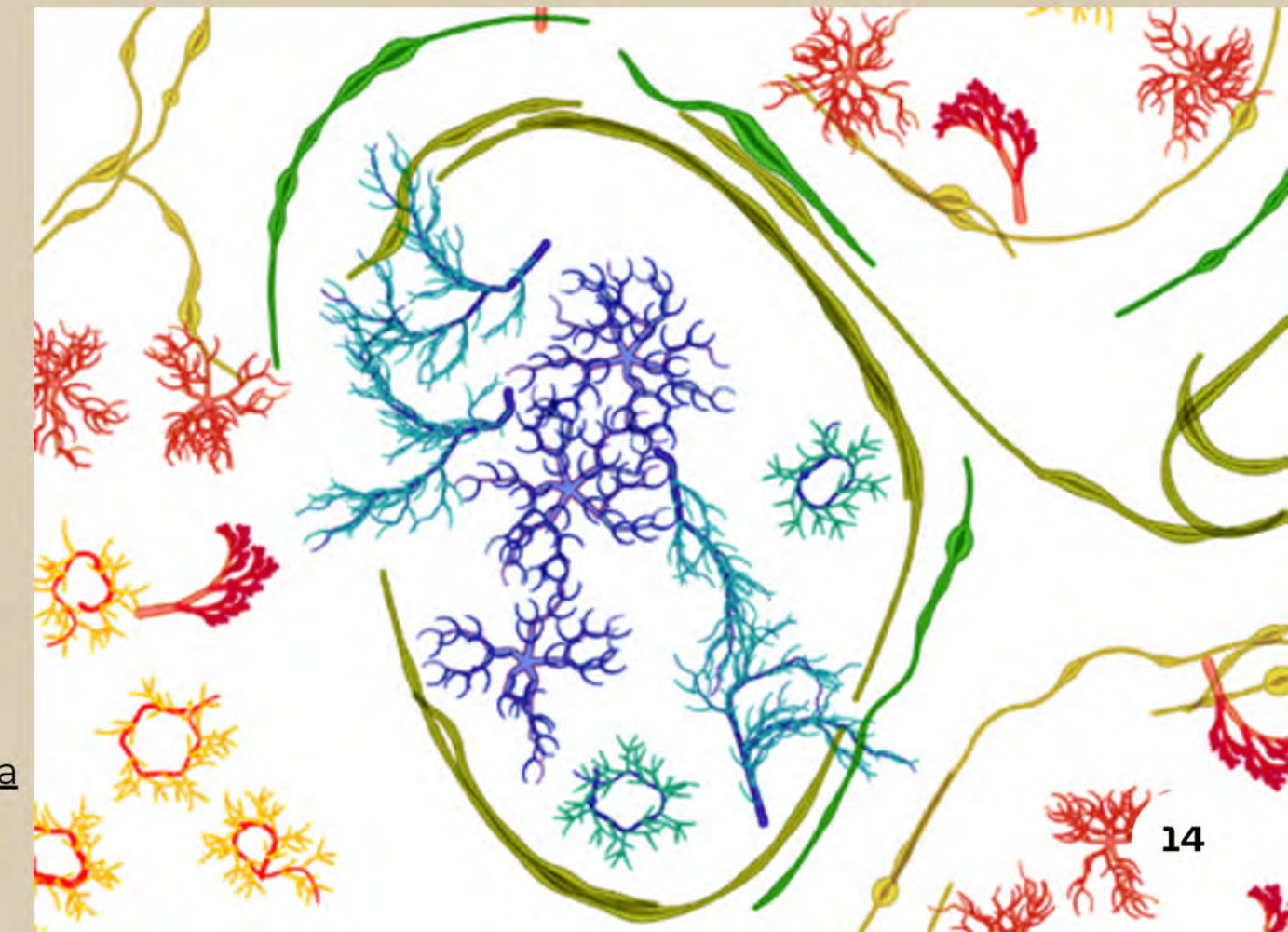


Figura 7 - Sem título (f091148), Phillip Galanter, 2002. Imagem criada através de software.
76,2 x 101,6cm .
<<http://philipgalanter.com/art/ga2lightbox/>> Acesso em julho de 2023.

Galanter cita dois artistas e suas obras para exemplificar a definição de arte generativa. Primeiro considera o artista Jackson Pollock e sua técnica de gotejamento. Pollock iniciou seus experimentos com esta técnica nos anos 40, colocando a tinta na tela e a deixando escorrer. A forma mais desenvolvida e usada, porém, contava com uma tela ou tecido grandes esticados no chão e o artista ‘jogava’ a tinta de forma estratégica, utilizando diversos materiais. Existe uma grande discussão sobre a inclusão da obra de Pollock como um sistema generativo ou não. A principal questão a ser considerada é a autonomia do sistema artístico. Claro, a tinta, no processo do gotejamento poderia interagir com o ar, a gravidade e outros sistemas naturais e com isso criar resultados diferentes. Mas essa ‘intervenção’ é quase mínima - se não for inexistente - e na maioria dos casos a tinta caía de um jeito e tomava certa forma devido ao processo guiado por Pollock, ou seja, o jeito que ele movia o pincel (ou outros materiais), a força que utilizava, a velocidade do movimento, etc. Suas pinturas não seriam criadas do jeito que foram se não houvesse esse ‘caos controlado’ por Pollock, por exemplo, como podemos perceber no quadro Number 1A (Figura 8). Em suma, Galanter não considera que as obras deste artista tenha a autonomia necessária para considerá-las generativa.

⁴ Tradução livre do original em inglês: “Generative art refers to any art practice where the artist uses a system, such as a set of natural language rules, a computer program, a machine, or other procedural invention, which is set into motion with some degree of autonomy contributing to or resulting in a completed work of art” (GALANTER, 2003, pg 4).



Figura 8 - Number 1A, Jackson Pollock, 1948. Óleo e tinta esmalte sob tela. 172.7 x 264.2 cm
<https://www.moma.org/collection/works/78699?sov_referrer=theme&theme_id=5117>
Acesso em julho de 2023.

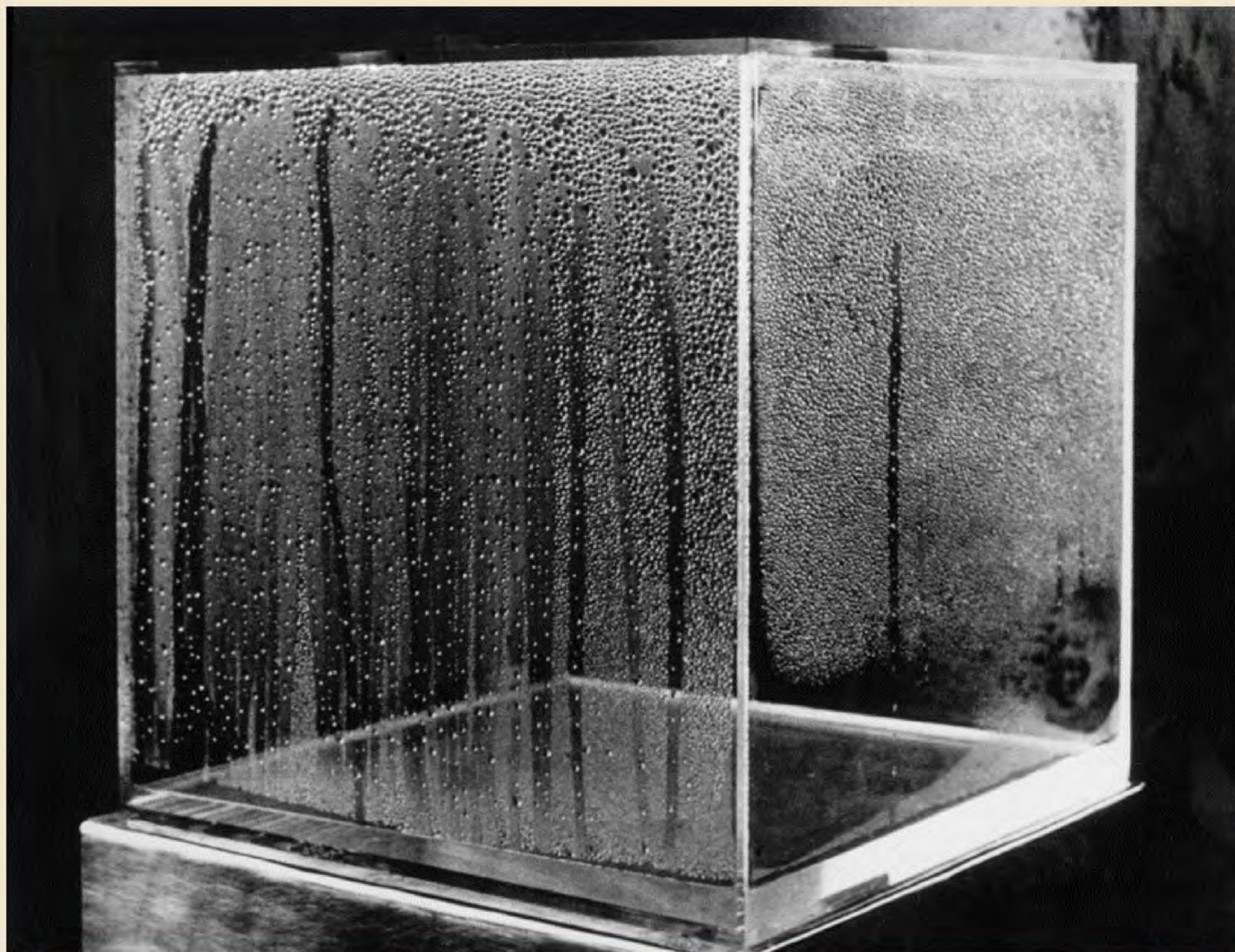


Figura 9 - Condensation Cube, Hans Haacke, 1963-1968. Escultura de Metacrilato e água, 76 x 76 x 76 cm <<https://www.macba.cat/en/art-artists/artists/haacke-hans/condensation-cube>> Acesso em julho de 2023.

A segunda consideração de Galanter é a obra *Condensation Cube* de Hans Haacke (Figura 9). Este trabalho foi exposto durante 5 anos (1963-68) e consiste em um cubo transparente contendo uma camada de água no fundo, que foi deixado no ambiente de exposição. Conforme a mudança na temperatura, da umidade, e de outros parâmetros do local, a água evaporava e se condensava, resultando em formas nas laterais e no topo do cubo. Aqui, novamente, a questão principal é a autonomia do sistema. O artista abre mão do controle do resultado da obra, colocando apenas o básico: o cubo e a água no ambiente. O resto foi gerado pela influência das condições sobre o cubo e a água dentro dele. A obra não teria o resultado que teve se não houvesse a autonomia da condensação nas paredes do cubo. Este é um ótimo exemplo de arte generativa que cabe na definição colocada por Galanter e não é feita por meio de um sistema eletrônico com computador. Assim, podemos considerar que o conceito de arte generativa não está estritamente relacionado ao uso da tecnologia digital.

1.2 A ARTE GENERATIVA HOJE E OS NFTS

Com o avanço da tecnologia, as ferramentas utilizadas para criação de arte generativa também evoluíram, além de sua popularização. A questão da comercialização da arte digital tem sido muito discutida e os NFTs têm aparecido cada vez mais como resposta a essa questão. NFT é a sigla para *non-fungible token* (token não fungível), que serve como identidade digital de um item. O NFT assegura a autenticidade daquele item, como um registro digital. A arte digital, entre outras formas de arte podem ser comercializadas como NFT.

O encontro da técnica generativa e essa forma de comércio se dá pela randomização e exclusividade de cada resultado artístico daquele processo. Desta forma, o artista pode gerar 10 imagens através de um código e todas serem únicas, com suas próprias características, e exclusivas, ou seja, pela randomização, utilizando o mesmo código, não haverá imagens iguais as 10 primeiras. E como se sabe, no mercado da arte e do comércio em geral, quanto mais rara e exclusiva é a obra, mais ela tende a ser valorizada, gerando mais lucro. Um exemplo deste processo é o conjunto de obra chamado *Fidenza*, do artista Tyler Hobbs (Figura 10).

Figura 10 - Fidenza, Tyler Hobbs, 2016 - presente. Imagem criada através de algoritmo generativo.
<<https://tylerxhobbs.com/fidenza>>
Acesso em julho de 2023.



Em *Fidenza*, o artista altera parâmetros que acabam dando características como a espessura de linhas, a paleta de cores e a direção ou intensidade das curvas presentes no trabalho. Hobbs vende suas obras como NFTs constantemente, e é um dos destaques deste mercado na atualidade.

Art Blocks é um projeto cujo objetivo é apoiar artistas na criação de NFTs. Segundo o grupo:

Art Blocks é uma plataforma para artistas publicarem edições exclusivas de arte generativa. Escolhidos por meio de um processo de inscrição altamente seletiva e orientados durante o desenvolvimento do trabalho, os artistas fazem upload de seu código criativo no Art Blocks. Quando um colecionador compra uma obra, eles acionam a criação de uma iteração nunca antes vista do projeto do artista na forma de uma obra de arte NFT.⁵ (Site ArtBlocks, 2022)

A Figura 11 apresenta um esquema simplificado feito pelos criadores de ArtBlocks para descrever o funcionamento da plataforma. O projeto é considerado um dos exemplos mais lucrativos envolvendo NFTs, juntamente com o site “*Autoglyphs*”, que tem um mecanismo de criação parecido. Esses projetos movimentam milhões de dólares anualmente.

O processo de arte generativa

Na Art Blocks, os artistas usam o código criativo para criar arte variada e generativa que evolui cada vez que um colecionador cunha uma peça.

O artista cria código para um projeto.

O colecionador de balas.

A peça se manifesta como um trabalho único na blockchain Ethereum e é compartilhada por artista, colecionador e comunidade ao mesmo tempo, juntos.

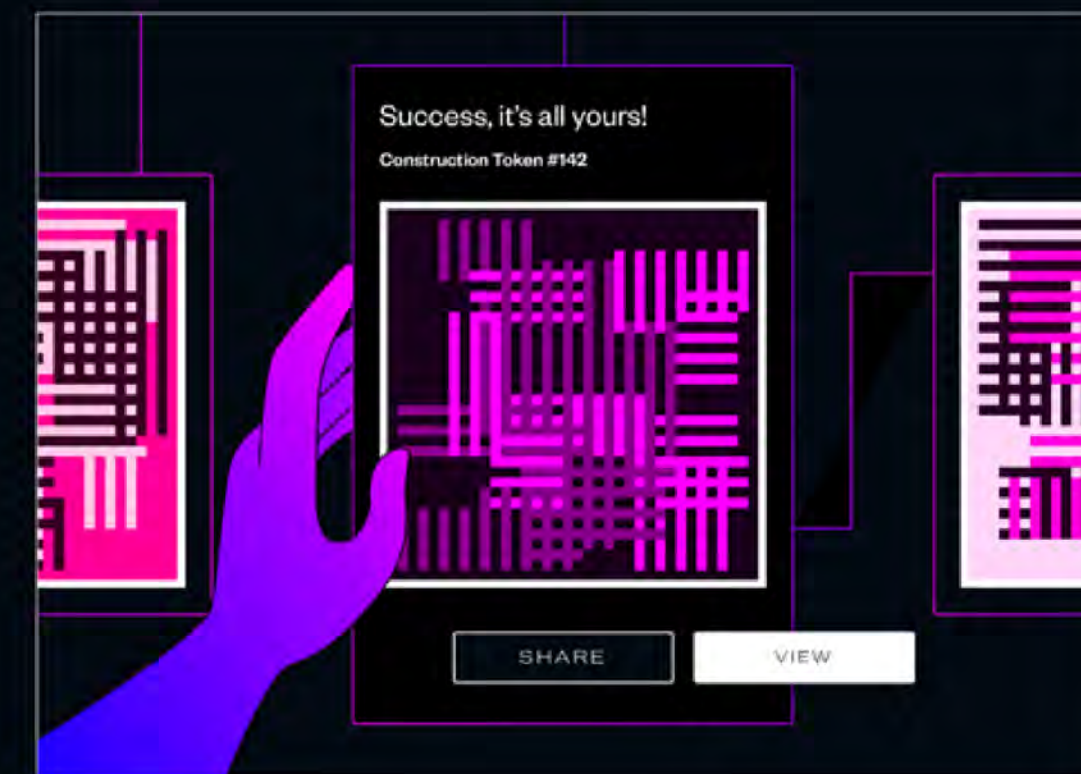


Figura 11 - Site ArtBlocks. Captura de tela da página “Como funciona” <<https://info.artblocks.io/how-it-works>> Acesso em julho de 2023.

⁵Tradução livre do original em inglês: “Art Blocks is a platform for artists to publish unique editions of generative art. Chosen through a highly selective application process, and mentored in development, artists upload their creative code on Art Blocks. When a collector purchases a work, they trigger the creation of a never-before-seen iteration of the artist’s project in the form of an NFT artwork.”

2. ESTUDOS DE CASO



Muitos artistas que exploram a arte generativa computacional utilizam formas geométricas simples para compor seus trabalhos. As formas simples são relevantes para este trabalho, pois se mostraram flexíveis, gerando uma variedade grande de resultados. Outro conjunto utilizado na arte generativa, mais especificamente nas referências deste trabalho, são itens botânicos, formas orgânicas ou que remetem à natureza. A mistura desses dois pontos iniciais gera resultados interessantes por utilizar a linguagem computacional — muitas vezes ligada a medidas exatas e controladas — para incluir elementos orgânicos, que são imprevisíveis e menos regrados. A repetição e a aleatoriedade também são relevantes para estas referências já que mostram como imagens visivelmente diferentes podem ser geradas com os mesmos itens iniciais.

Além da questão estética, o presente trabalho procura explorar o papel do artista na arte generativa e qual resultado pode ser atingido cedendo o controle após definidos os parâmetros iniciais. As artistas que serão citadas têm trabalhos em que essas questões são colocadas utilizando os fatores aleatórios, de repetição e uso de contraste entre geometria e formas orgânicas. A escolha de três artistas mulheres e brasileiras é de extrema relevância em um ambiente onde a maioria dos criadores multimídia e programadores são homens.

2.1. VERA BIGHETTI

Vera Bighetti é uma artista e pesquisadora brasileira na área de arte tecnológica e processos com rotinas autogerativas. A artista cria projetos envolvendo dispositivos de comunicação interativa desde 1998. Seus trabalhos envolvem, em sua maioria, o programa *Processing*.

Os trabalhos colocados nas Figuras 12 e 13 utilizam formas simples, como linhas e retângulos, mas a variação de cor e a sobreposição das formas caracterizam a complexidade. A randomização da posição e quantidade dos itens é uma característica da arte generativa muito presente nesses trabalhos. O esquema de cores, como pode ser observado em todas as imagens, também traz mais profundidade ao trabalho final.

O processo da arte generativa permite que sejam criadas diversas soluções para uma questão inicial, como podemos perceber nas imagens. Estes trabalhos foram feitos com o mesmo código base, com mudanças seletivas e resultados randômicos. Cada imagem pode ser apreciada individualmente como uma arte gerada por códigos. Porém, quando vistas em conjunto, é possível perceber as nuances do processo e até despertar um questionamento por parte do observador em relação ao processo criativo, principalmente quando não há muito conhecimento sobre arte computacional e generativa.

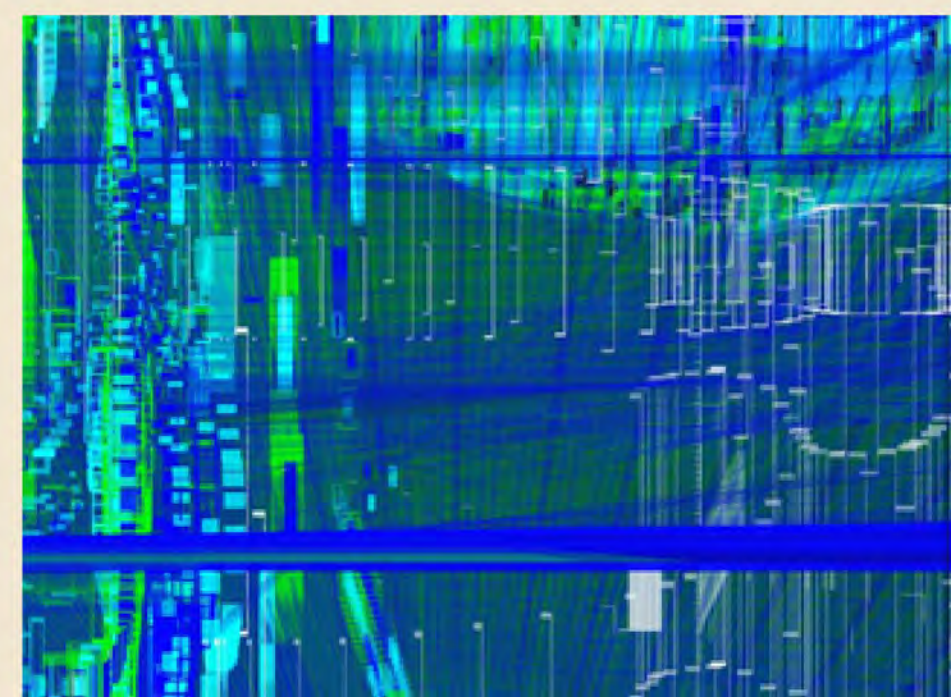
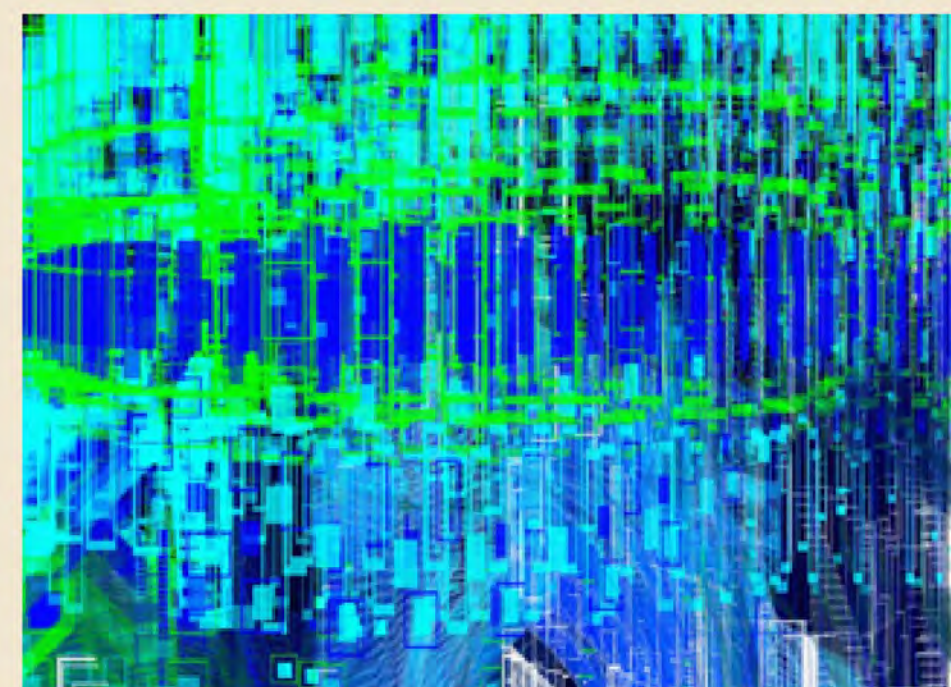


Figura 12 - Processo generativo 11, 07 e 09, Vera Bighetti, 2008.
<<https://www.flickr.com/photos/vbighetti/albums/72157622391421719/with/3981592260/>>
Acesso em julho de 2023.



Figura 13 - Processo generativo 06, Vera Bighetti, 2008
<https://www.flickr.com/photos/vbighetti/3981604412/in/album-72157622391421719/>
 Acesso em julho de 2023.

Outra dimensão nessa análise é como várias séries foram feitas com as mesmas características, formas e movimentos e ainda assim são independentes, podendo ser relacionadas entre si pelas similaridades e contraste. A arte generativa, neste caso, cobre uma gama extensa de soluções mesmo que os parâmetros iniciais sejam os mesmos, como podemos ver na comparação entre as Figuras 13 e 14.

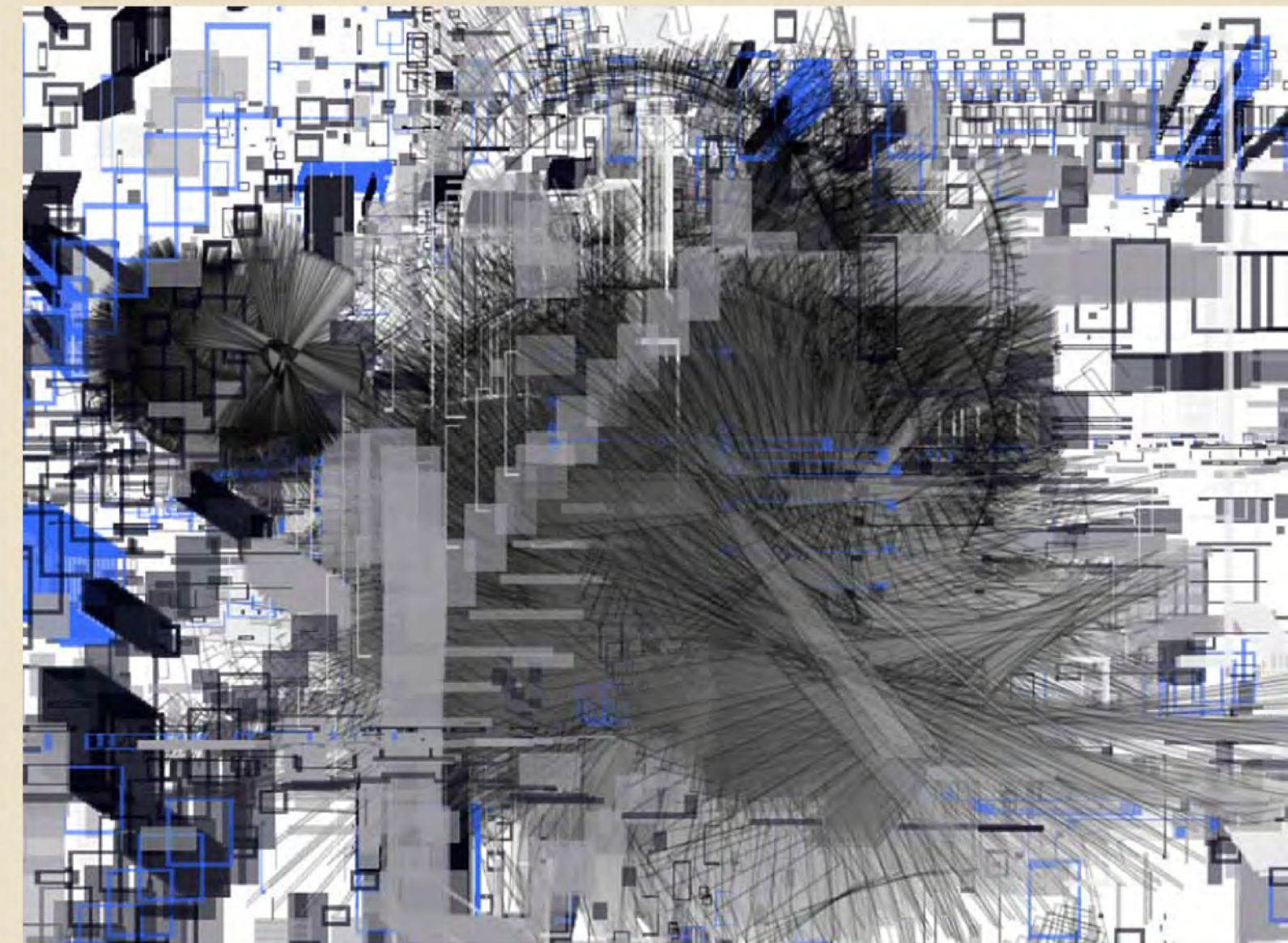


Figura 14 - Processo generativo 18, Vera Bighetti, 2008.
<https://www.flickr.com/photos/vbighetti/3981596208/in/album-72157622391421719/> Acesso em julho de 2023.

2.2. MONICA RIZZOLLI

Monica Rizzolli é uma artista brasileira, programadora, performer e ilustradora. Com carreira nacional e internacional, um dos destaques de sua produção é a exploração do mercado de NFTs. Através da plataforma Artblocks - citada anteriormente - a artista chegou a arrecadar 1.623 *ether*, que é equivalente a R\$28 milhões (SIMÕES, 2021) com a coleção *Fragments of an Infinite Field*, exemplificada na Figura 15.

Em sua página do Tumblr, a artista compartilha diversos outros trabalhos criados por meio de códigos generativos. Rizzolli também aplica códigos para criar estampas, quadros, capas de livros misturando design e arte. Um destaque da sua produção é a mistura entre imagens orgânicas e geométricas, como o exemplo da Figura 15 e o trabalho "Sem título" da Figura 16, que mistura uma textura que lembra folhagem e linhas com diferentes espessuras.

A maioria das obras de Monica têm como destaque a junção com contraste entre formas orgânicas e geométricas, como na Figura 16. Em alguns trabalhos, como na Figura 15, existe uma ideia de estampa, repetição e sobreposição. Apesar de ser parecido com uma colagem ou desenho, este é um resultado que pode ser alcançado através da arte generativa, utilizando parâmetros randômicos ou não.



Figura 15 - Fragments of an Infinite Field #840 e #11, Monica Rizzolli.
<<https://opensea.io/collection/fragments-of-an-infinite-field-by-monica-rizzolli>> Acesso em julho de 2023.

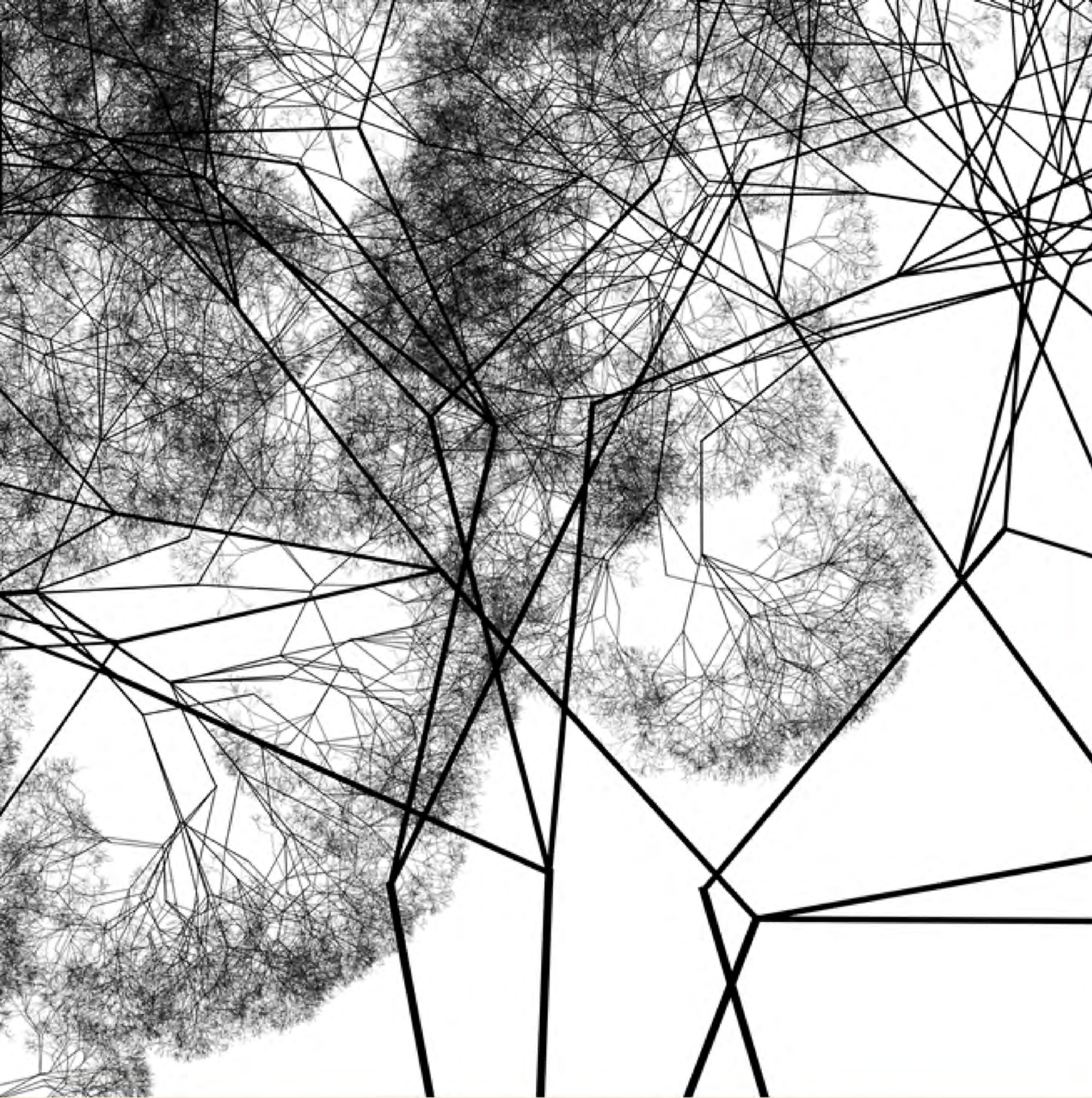


Figura 16 - Sem título, Monica Rizzolli, 2018. <<https://monicarizzolli.tumblr.com/>>
Acesso em julho de 2023.

Uma diferença entre a arte generativa e a arte 'tradicional' é o envolvimento do artista. Em uma colagem, por exemplo, existe a tomada de decisão constante durante o trabalho, selecionando e interferindo nas imagens, testando posições e montando a configuração final. Em arte generativa, o artista abre mão de certa responsabilidade dos resultados esperados. Isso gera uma gama de possibilidades em um curto espaço de tempo, e assim, diversas opções de escolha. Essa forma de criação pode ser usada tanto para diversificar as alternativas quanto para deixar a composição de itens da obra ao acaso, usando funções de aleatoriedade.



2.3. SUZETE VENTURELLI

Suzete Venturelli é uma artista e pesquisadora brasileira pioneira na arte computacional desde os anos 80. Sua produção envolve a Arte Computacional, Gamearte, Realidade Virtual, Animação, Arte digital, *Sound Art* e Imagem Interativa. Venturelli também organiza diversas exposições (inter)nacionais com trabalhos de arte computacional.

A Figura 17 apresenta um trabalho que esteve na exposição #ART de 2020 e é uma imagem generativa que aos poucos vai se formando na tela. As linhas, formas e cores são adicionadas de forma generativa com um nível randomizado, ou seja, não há ordem específica de criação e os itens podem ser sobrepostos ou não.

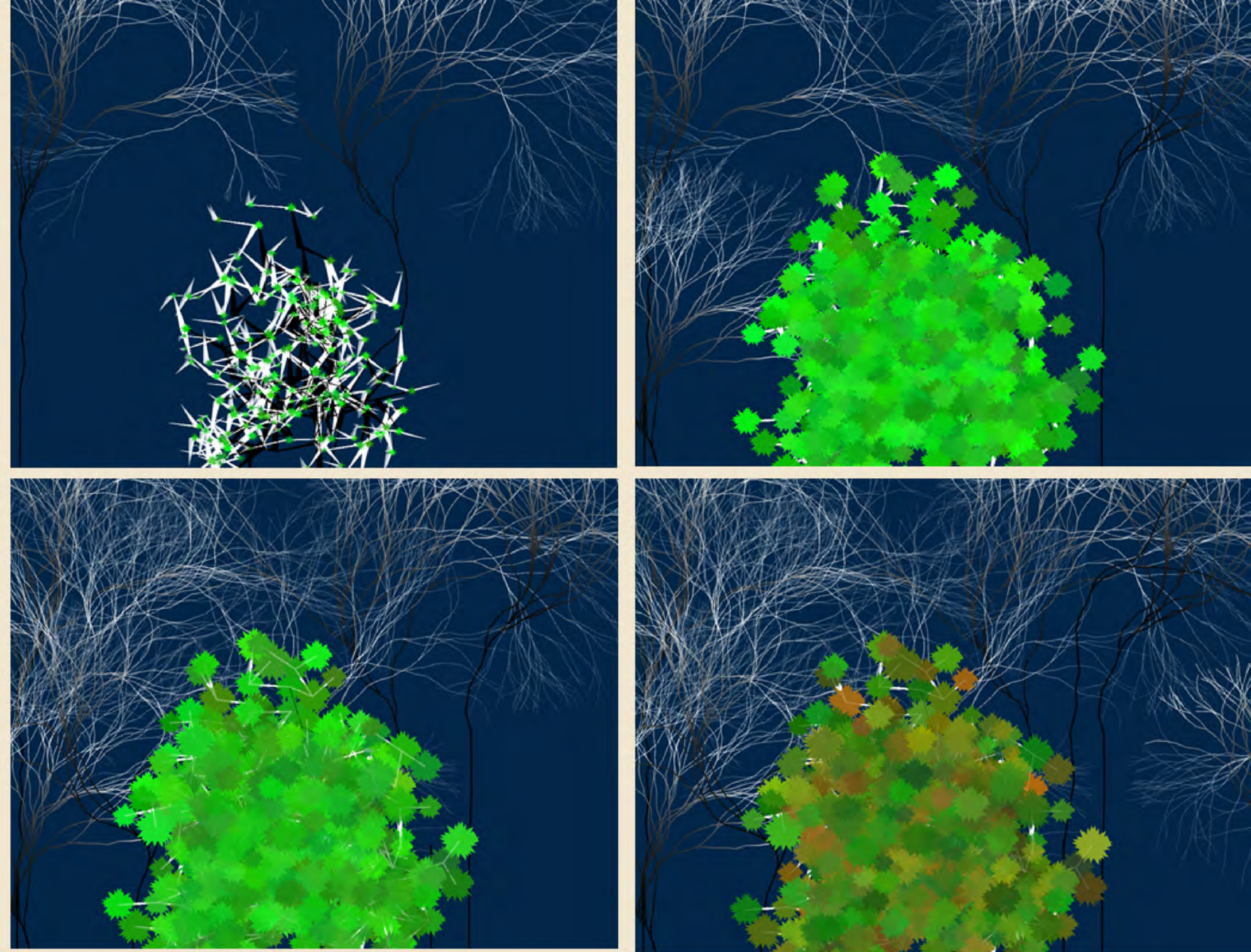


Figura 17 - Trees grow beyond-inside the screen, Suzete Venturelli, 2020. Captura de tela de parte do processo generativo. <<https://emmeio12.medialab.unb.br/trees-grow-beyond-inside-the-screen/>> Acesso em julho de 2023.

Os destaques deste trabalho, além da questão estética, é o modo em que ele é formado. A construção da imagem demonstra as fases do processo generativo, mas também é o próprio trabalho em constante mudança.

É importante citar que cada vez que a obra é acessada, a construção é iniciada de forma diferente. No exemplo da Figura 17, as cores e os posicionamentos são únicos daquela 'sessão'. Retornando no link da exposição, outra composição é iniciada com diferentes cores, ordens e posicionamentos dos itens, como pode ser visto na Figura 18.

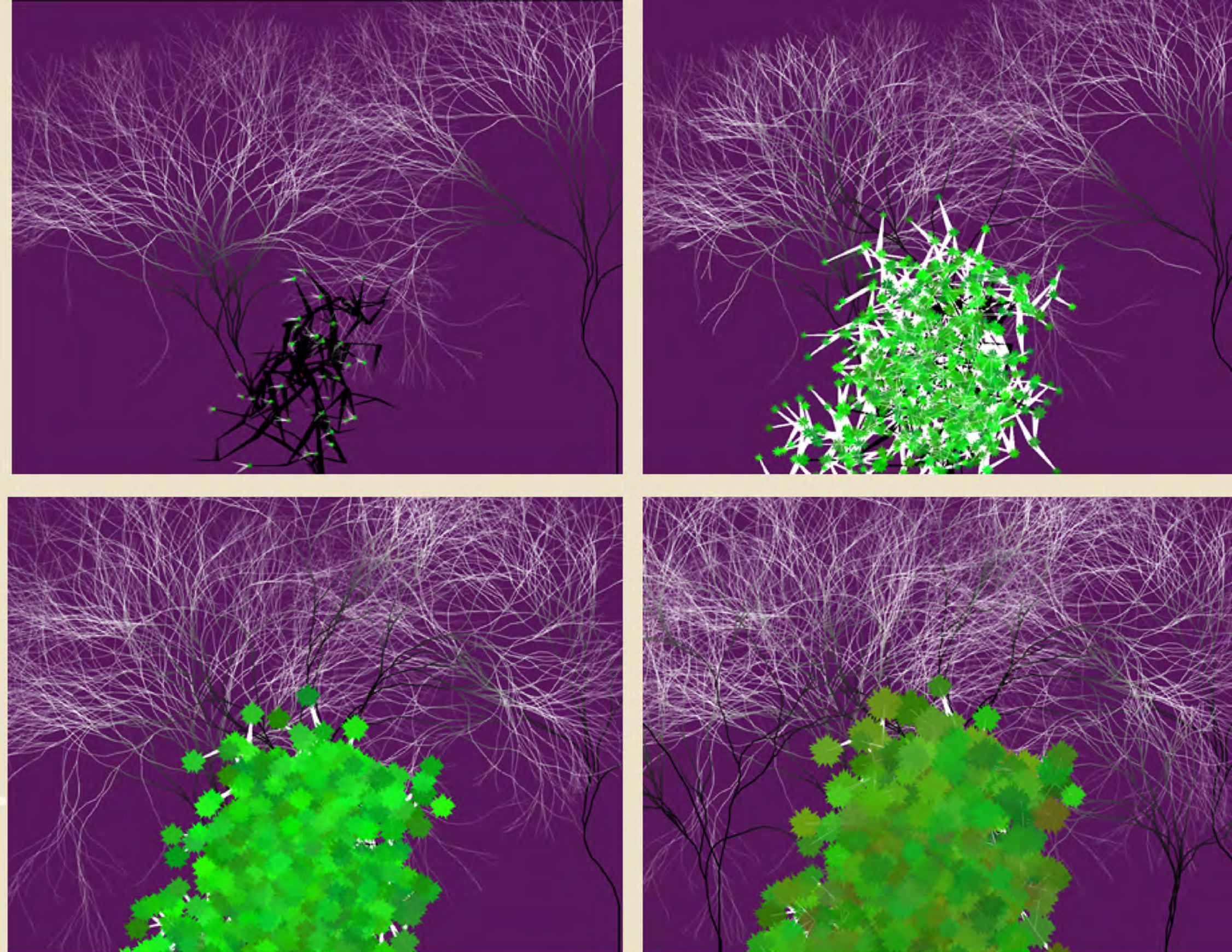
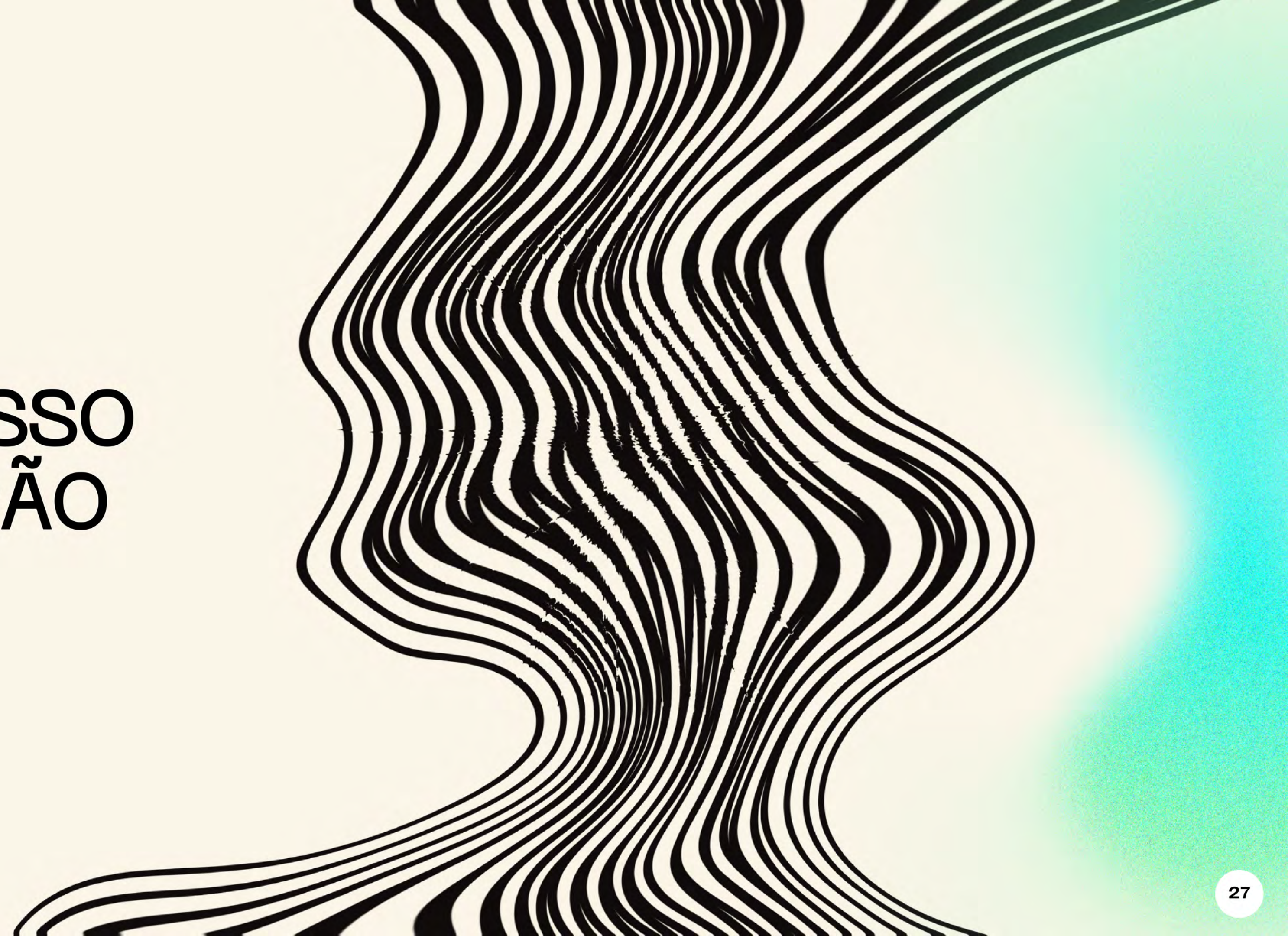


Figura 18 - Trees grow beyond-inside the screen, Suzete Venturelli, 2020. Captura de tela de parte do processo generativo. <<https://emmeio12.medialab.unb.br/trees-grow-beyond-inside-the-screen/>> Acesso em julho de 2023.

3. PROCESSO DE CRIAÇÃO



Neste capítulo, o processo de criação e experimentação será detalhado em primeira pessoa para descrever a linha de pensamento pessoal da autora e cumprir com a proposta do trabalho — fazer um conjunto de experimentações a partir da pesquisa sobre a arte generativa.

No início deste trabalho, em 2022, meu conhecimento de arte generativa era pequeno e o de programação quase nulo, tendo aprendido de forma passiva em aulas durante a graduação. Por isso, hesitei em partir para o uso de ferramentas de programação de início. O *Processing* foi a opção inicial, mas rapidamente mudei para o p5.js, que contém uma comunidade ampla nas redes sociais, principalmente no YouTube, fonte usada de forma constante para aprendizado detalhado e rápido. A Universidade de York, do Reino Unido define:

“O p5.js é uma versão do Processing baseada em JavaScript, ao invés de Java. Apesar das semelhanças de nomes, são linguagens de codificação completamente diferentes. Na verdade, é uma biblioteca JavaScript, o que significa que existem funcionalidades bônus que você pode adicionar ao código JavaScript.” ⁶ (Guia Prático de programação, Universidade de York, 2023)

Segundo os criadores do p5.js, a plataforma tem como foco tornar a codificação acessível e inclusiva para artistas, designers, educadores e iniciantes. A biblioteca de exemplos do site também ajudou no processo de aprendizagem, contendo informações mais técnicas e básicas sobre cada ferramenta.



⁶ Tradução livre do original em inglês: “p5.js is a version of Processing that is based off JavaScript rather than Java (despite the name similarities, they are completely different coding languages). It is actually a JavaScript library, which means that it is some bonus functionality you can add to JavaScript code”

3.1 DESENVOLVIMENTO

Como citado anteriormente, os interesses deste trabalho envolvem imagens com formas geométricas e botânicas trabalhando juntas na construção do resultado. O início dos experimentos foi feito com formas simples, já que o conhecimento nesse tipo de programação era limitado. Então, estudei como podem ser geradas linhas com variações aleatórias, considerando que o uso do *random* no código é fundamental para a premissa do trabalho. Ambos modelos utilizados neste código inicial, apresentado na Figura 19, fazem parte da biblioteca de exemplos o p5.js. As linhas exemplificam o próprio uso do *random*⁷ e a flor é encontrada em um pacote de formas simples⁸.

Neste momento ficou claro que as linhas me interessavam, mas a abordagem da parte botânica não me agradava por ser muito ilustrativa. Procurei alternativas para modificar as linhas e itens que remetem à botânica para adicionar ao código. O código do programador Steve's Makerspace⁹ deu uma perspectiva de como incluir fatores orgânicos que mudariam cada vez que o código fosse acionado.

⁷ Exemplo de linhas criadas pela função random: <https://p5js.org/reference/#/p5/random>

⁸ Exemplo de como construir formas simples: <https://p5js.org/examples/hello-p5-simple-shapes.html>

⁹ O artista e programador usa o nome "Steve's Makerspace" para os créditos de seus códigos, sendo o mesmo nome de seu canal do YouTube encontrado em: <https://www.youtube.com/@StevesMakerspace/videos>

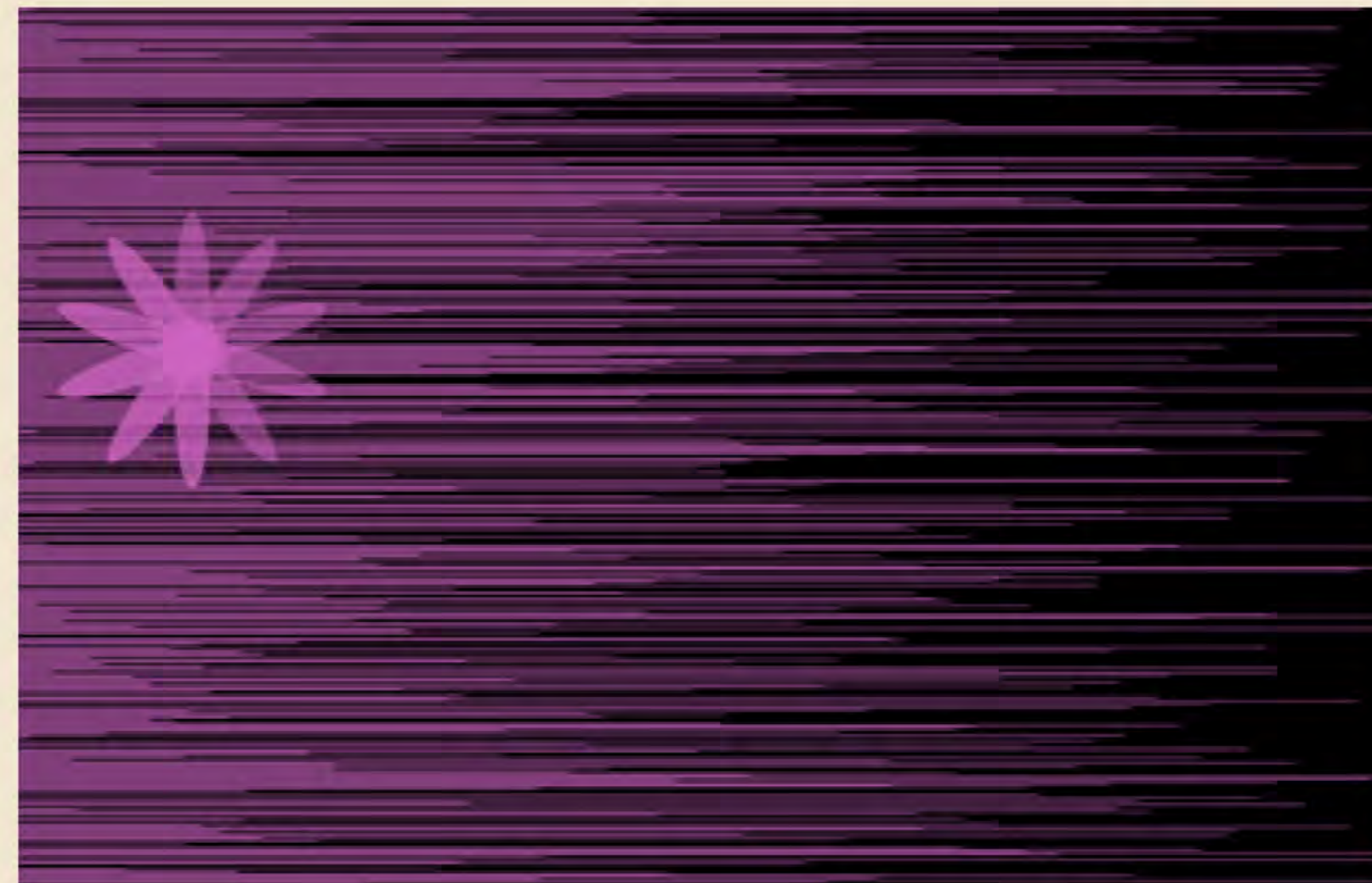
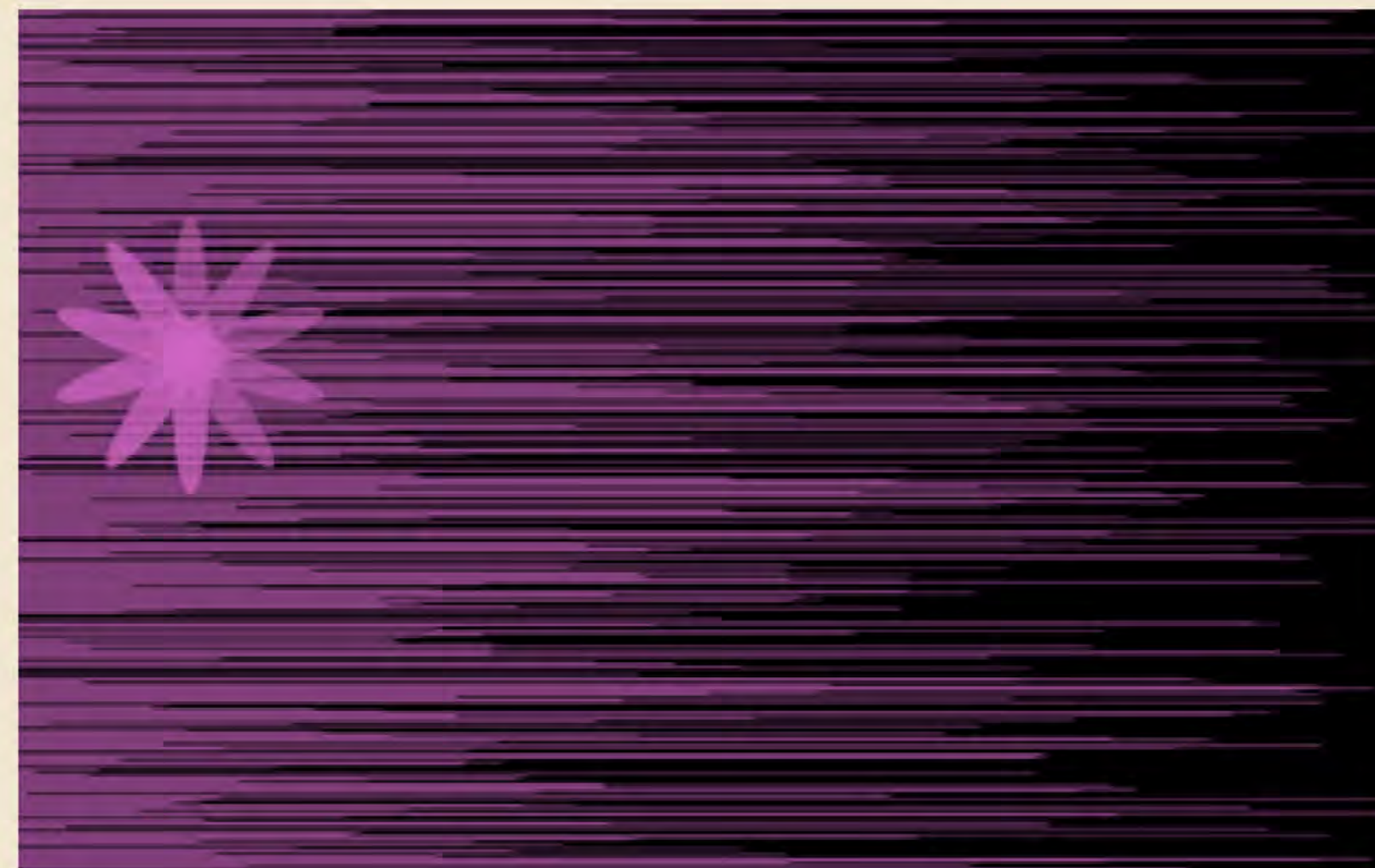


Figura 19 - Teste 1, Beatriz Costa, 2023. Imagem criada por código generativo. Acervo pessoal.

Steve faz tutoriais no YouTube, então apesar de seu código apresentar resultados como na Figura 20, o artista apresentou seu processo de criação e me ajudou a modificar seu código para alcançar os resultados das Figuras 21 e 22.

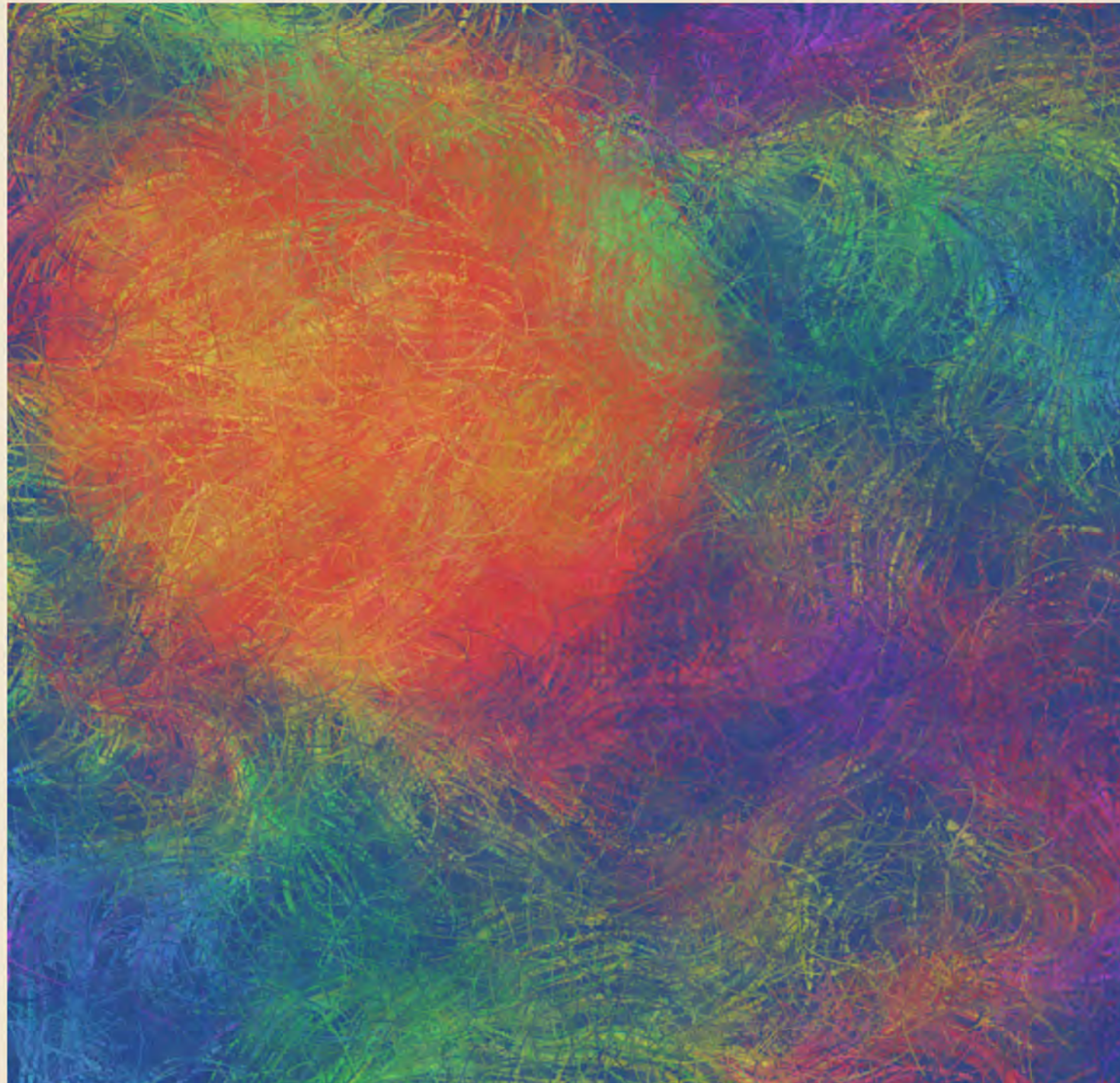


Figura 20 - Flow Fields for YT, Steve's Makerspace,, 2023. Imagem criada por código generativo. <<https://editor.p5js.org/StevesMakerspace/sketches/4zn-XY-FH>> acesso em julho de 2023.

Figura 21 - Flow Fields for YT, Steve's Makerspace, modificado por Beatriz Costa, 2023. Imagem criada por código generativo. <<https://editor.p5js.org/beacoxta/sketches/t5dyMN4le>> acesso em julho de 2023.

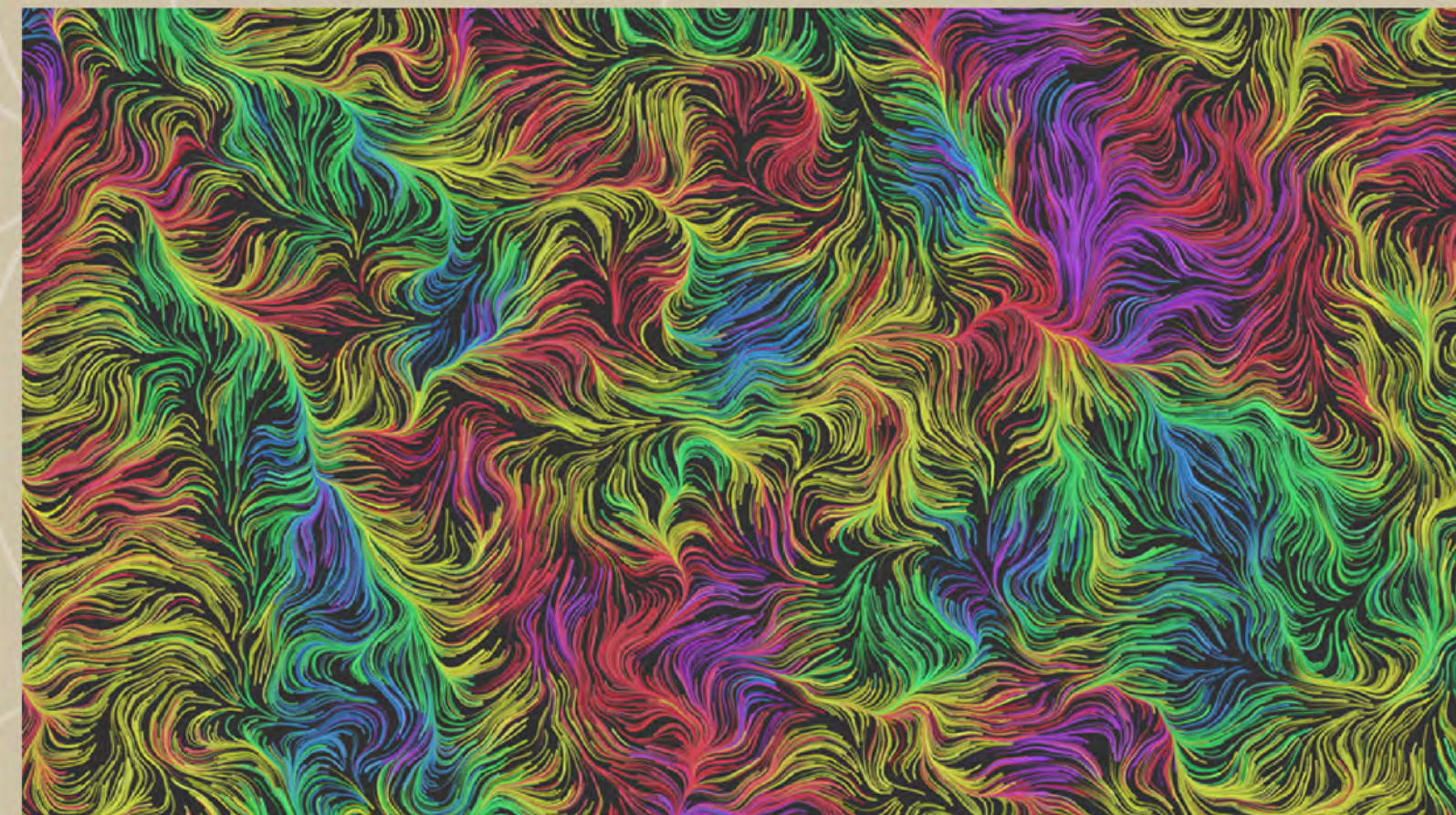
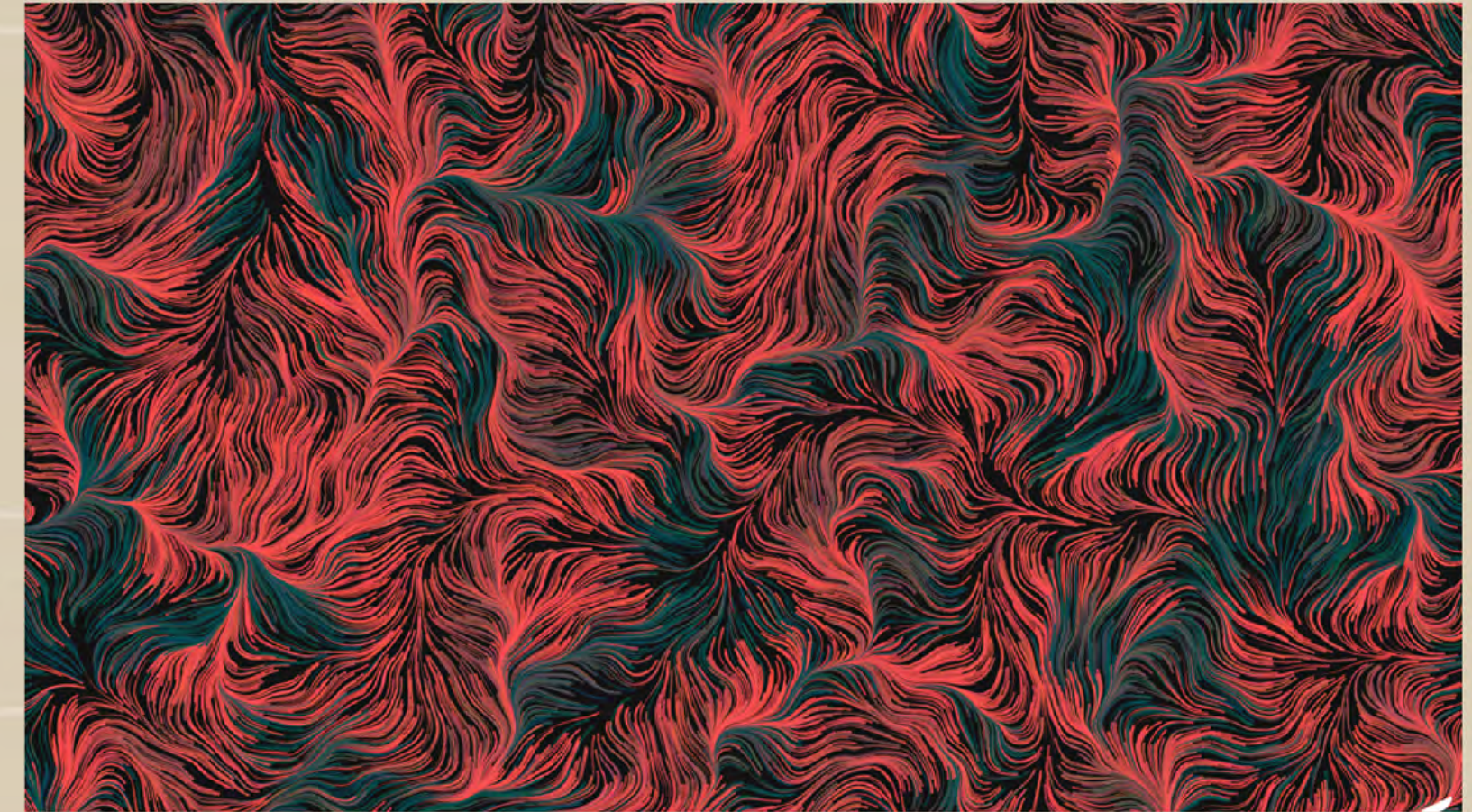
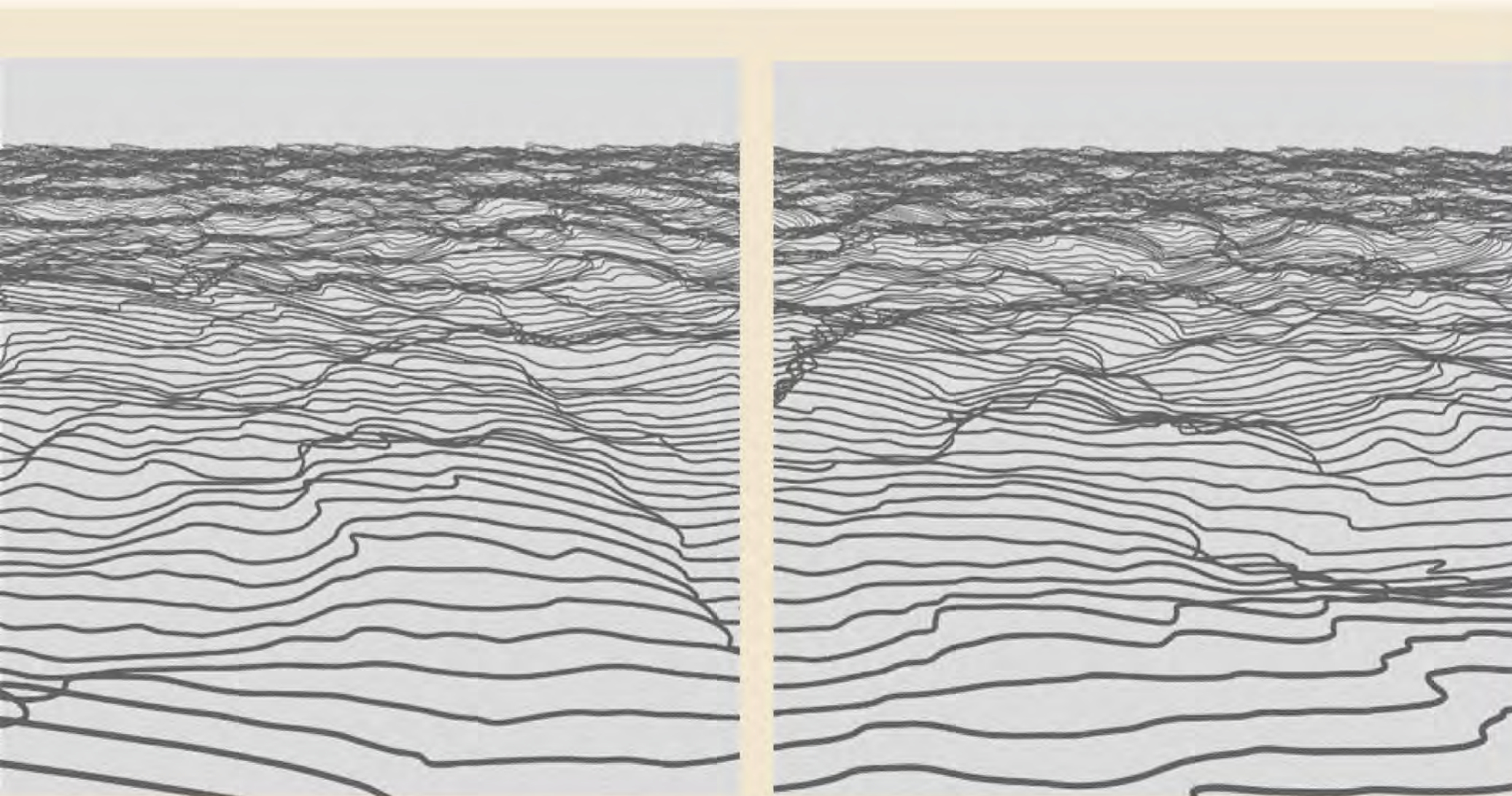


Figura 22 - Flow Fields for YT, Steve's Makerspace, modificado por Beatriz Costa, 2023. Imagem criada por código generativo. <<https://editor.p5js.org/beacoxta/sketches/t5dyMN4le>> Acesso em julho de 2023.

Inspirada nisso resolvi testar as mesmas ondulações, mas em 3 dimensões (Figura 23). As opções de código até agora tinham sido criadas no espaço padrão do p5.js, que só permite criações em 2 dimensões. Para iniciar os experimentos em 3 dimensões, foi necessário adicionar a função *WEBGL*, que permite a construção tridimensional.



Figuras 23 - Teste 2, Beatriz Costa, 2023. Imagem criada por código generativo. Acervo pessoal.

O código criado para essa primeira iteração ficou muito pesado por conter muitas linhas e não seria prático realizar diversos testes e adicionar os outros itens à imagem. Então, inspirada nas criações das Figuras 24, 25 e 26, resolvi utilizar um código que daria a mesma ideia de ondulações em 3 dimensões, porém com uma quantidade menor de linhas.

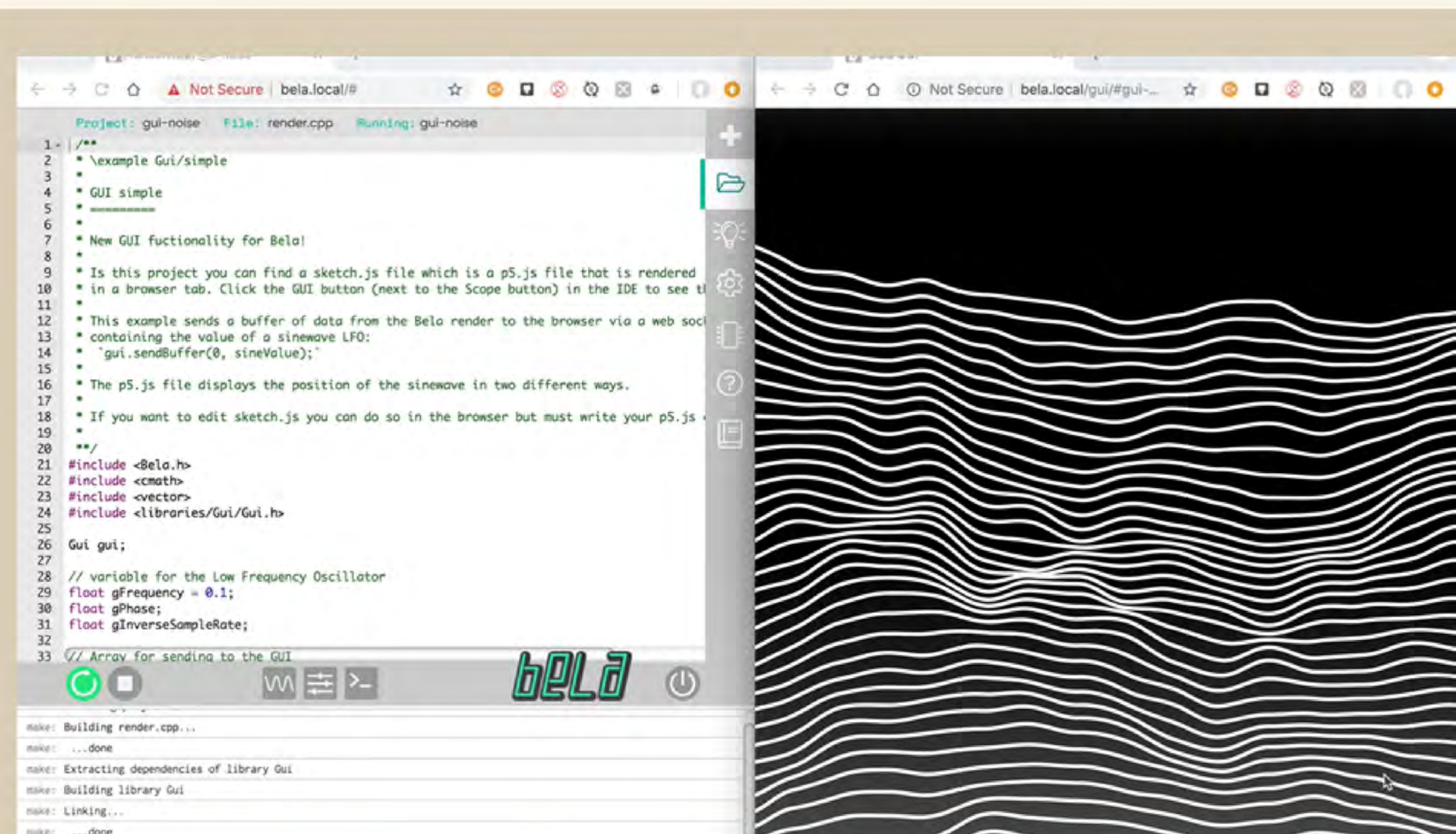


Figura 24 - Bela IDE – p5js Integration, Bela Platform, 2020. Captura de tela de vídeo.. <<https://www.youtube.com/watch?v=XZ9eGP94sKY>> acesso em julho de 2023.

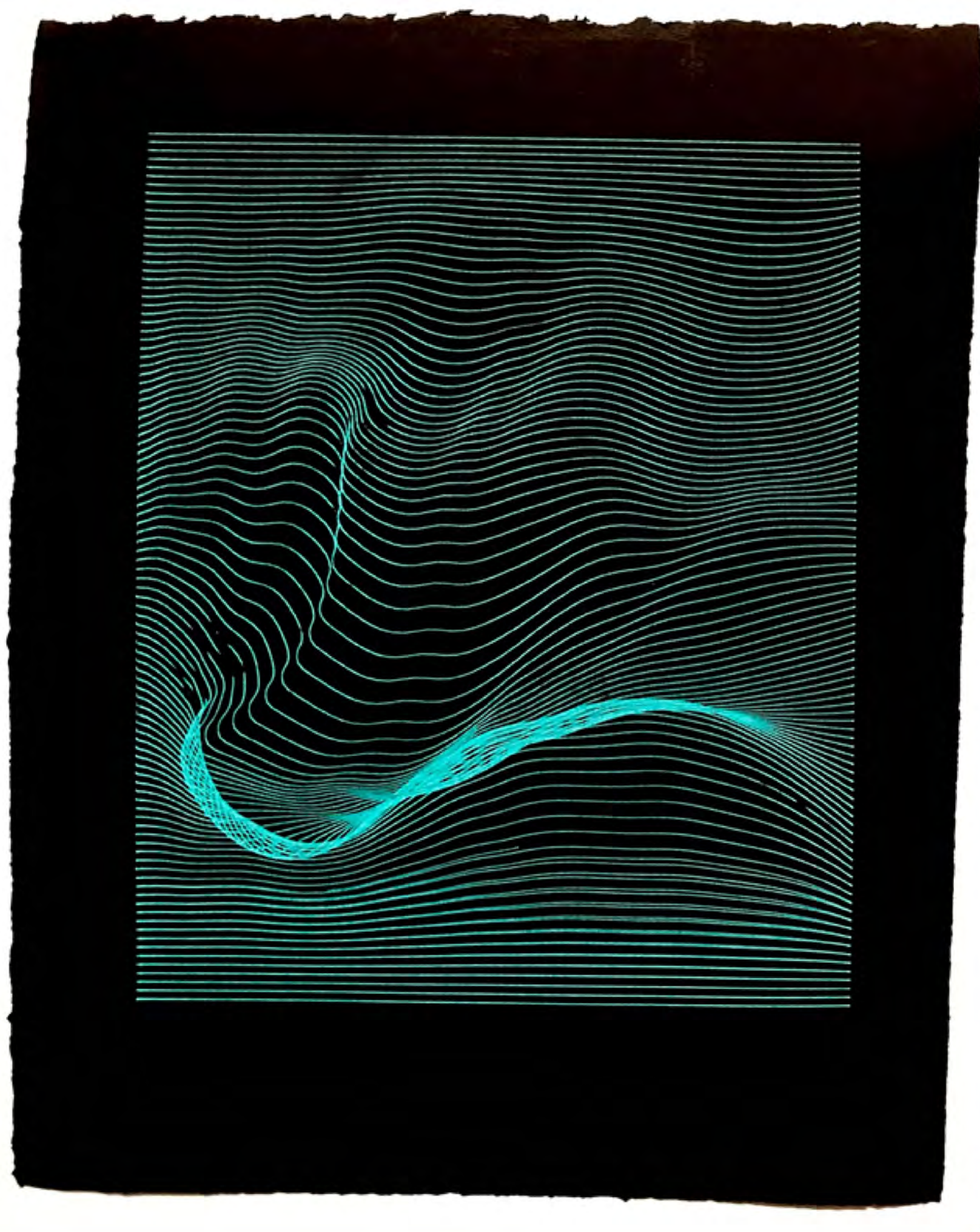


Figura 25 -
Programmed Plotter
Drawings, Roni
Cantor. Imagem
gerada por código e
impressa por
plotadora
<<https://p5js.org/showcase/featuring/roni-cantor.html>> acesso
em julho de 2023.

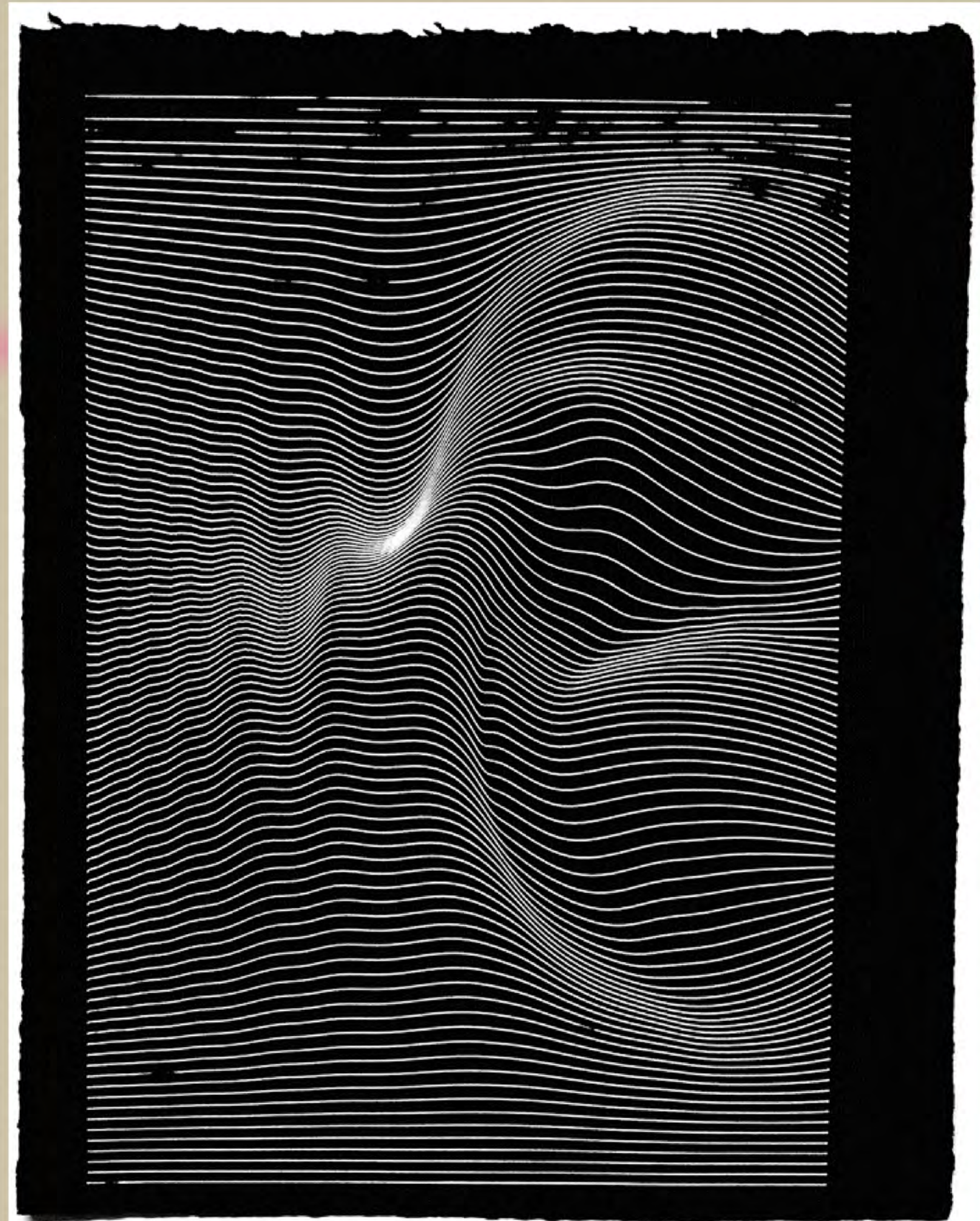


Figura 26 -
Programmed Plotter
Drawings, Roni
Cantor, s/d. Imagem
gerada por código e
impressa por
plotadora
<<https://p5js.org/showcase/featuring/roni-cantor.html>> acesso
em julho de 2023.

Inicialmente, a quantidade de linhas era constante, como pode ser visto nas imagens 27 e 28, ambas contêm cerca de 400 linhas. As ondulações foram feitas através da função *Perlin Noise* e fiz algumas simulações mudando a quantidade de noise que pretendia colocar. Quando muito pequeno, ele fica extremamente suave, como na Figura 28 e quando mais perto de zero, menor fica a ondulação. Quanto maior é o valor, mais interrupções são adicionadas às linhas, como no exemplo da Figura 27.

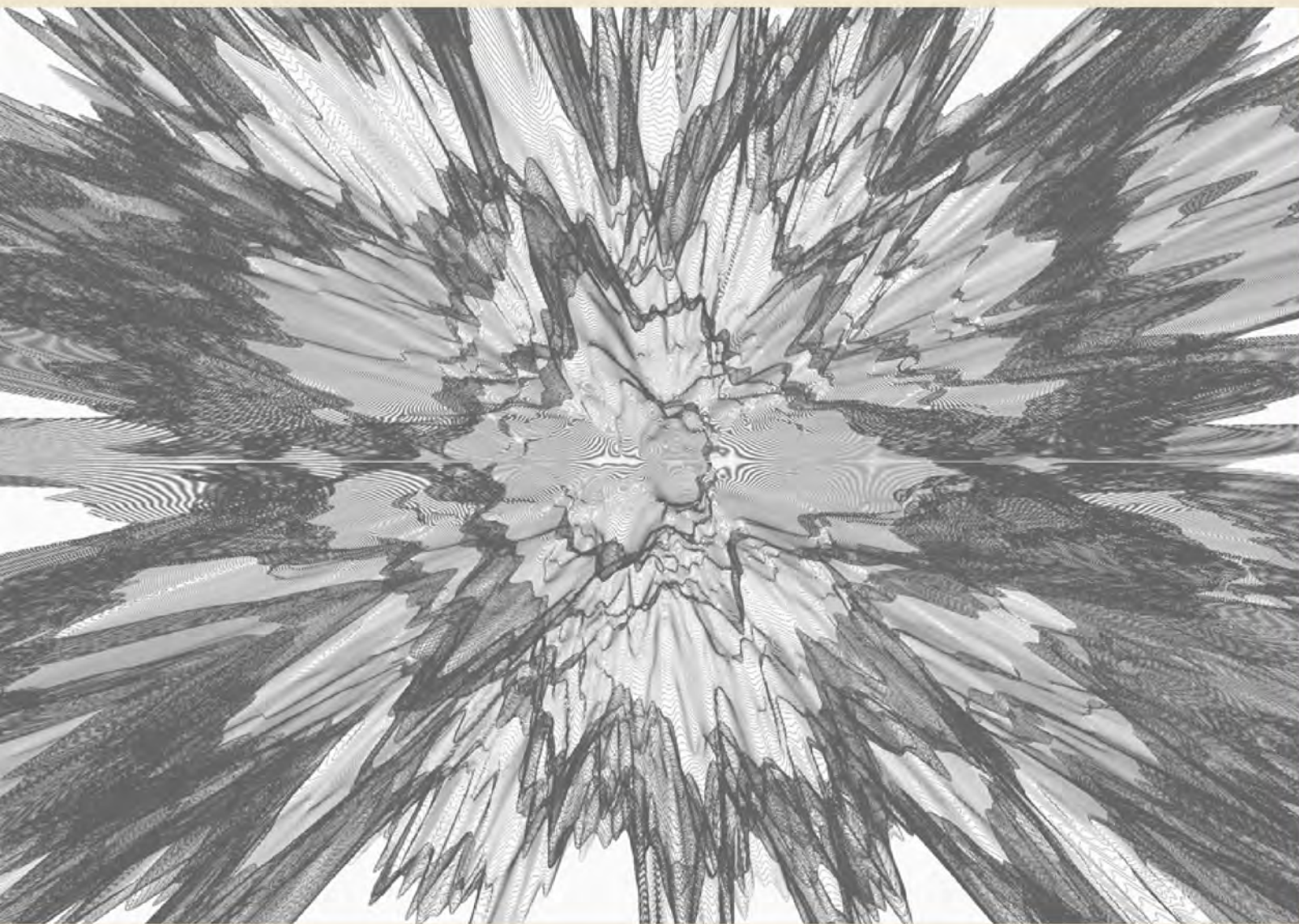


Figura 27 - Sem título, Beatriz Costa, 2023. Imagem gerada por código generativo. Acervo pessoal.

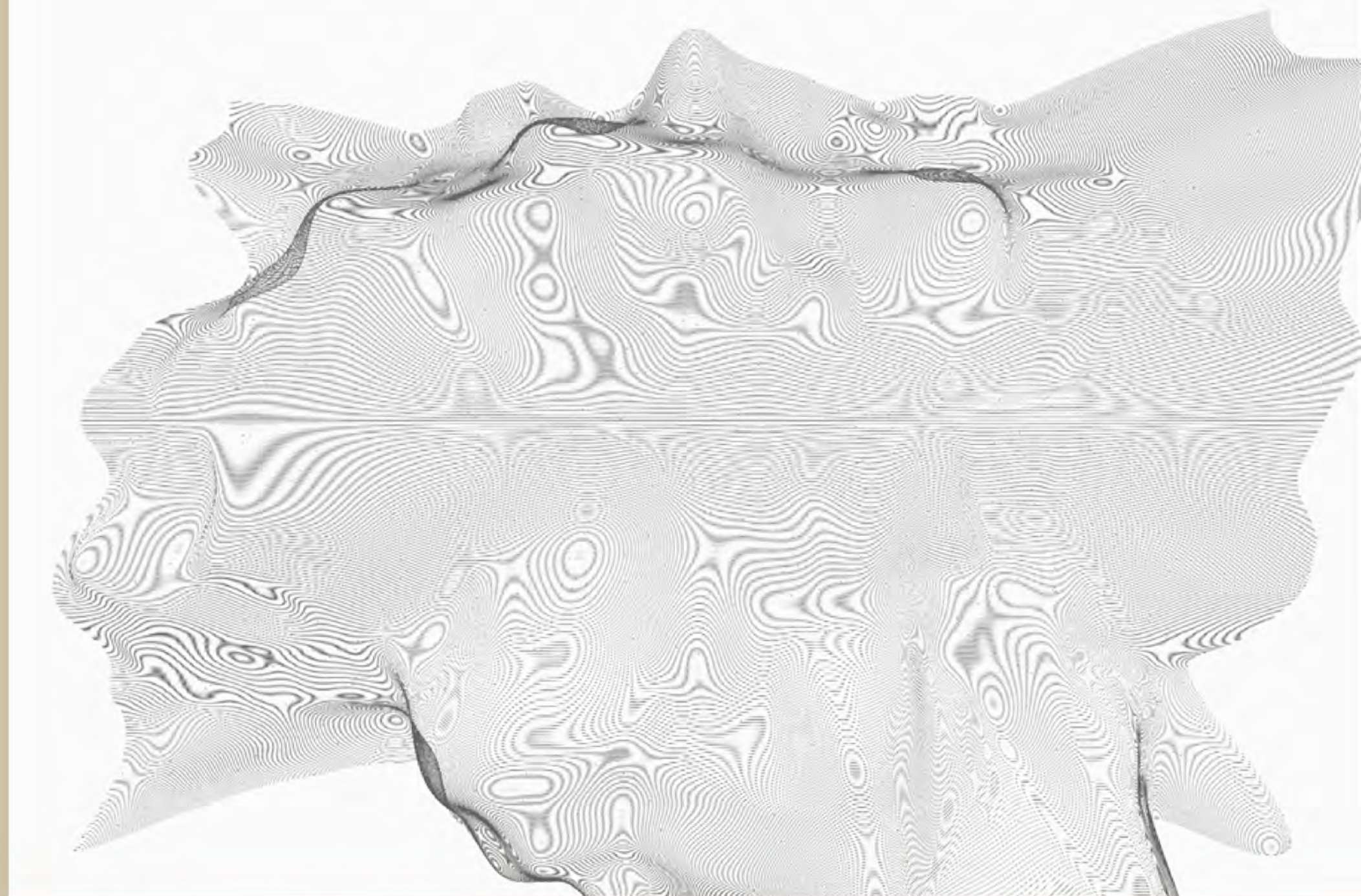


Figura 28 - Sem título, Beatriz Costa, 2023. Imagem gerada por código generativo. Acervo pessoal.

A decisão do valor do noise foi feita por meio de testes para decidir o equilíbrio entre a suavidade e algo mais dramático. O valor decidido foi 0.005. O código conta com uma função de aleatoriedade, o *random*, que é aplicado ao *Perlin Noise* a fim de criar um conjunto diferente cada vez que o código for acionado. As Figuras 29 e 30 são exemplos desta função que cria uma variação de formas, mesmo o parâmetro de noise estando constante entre elas.

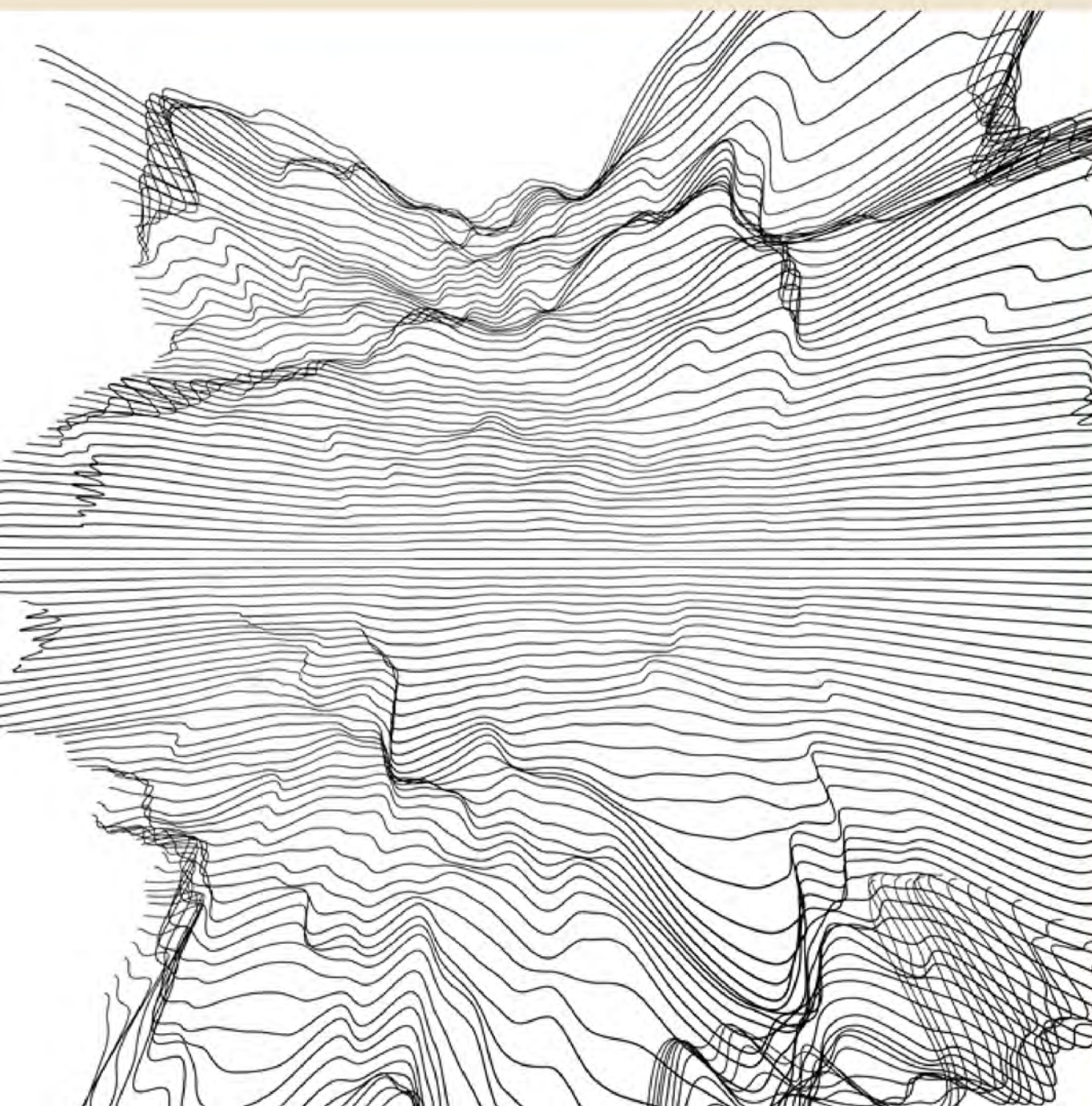


Figura 29 - Sem título, Beatriz Costa, 2023. Imagem gerada por código generativo. Acervo pessoal.

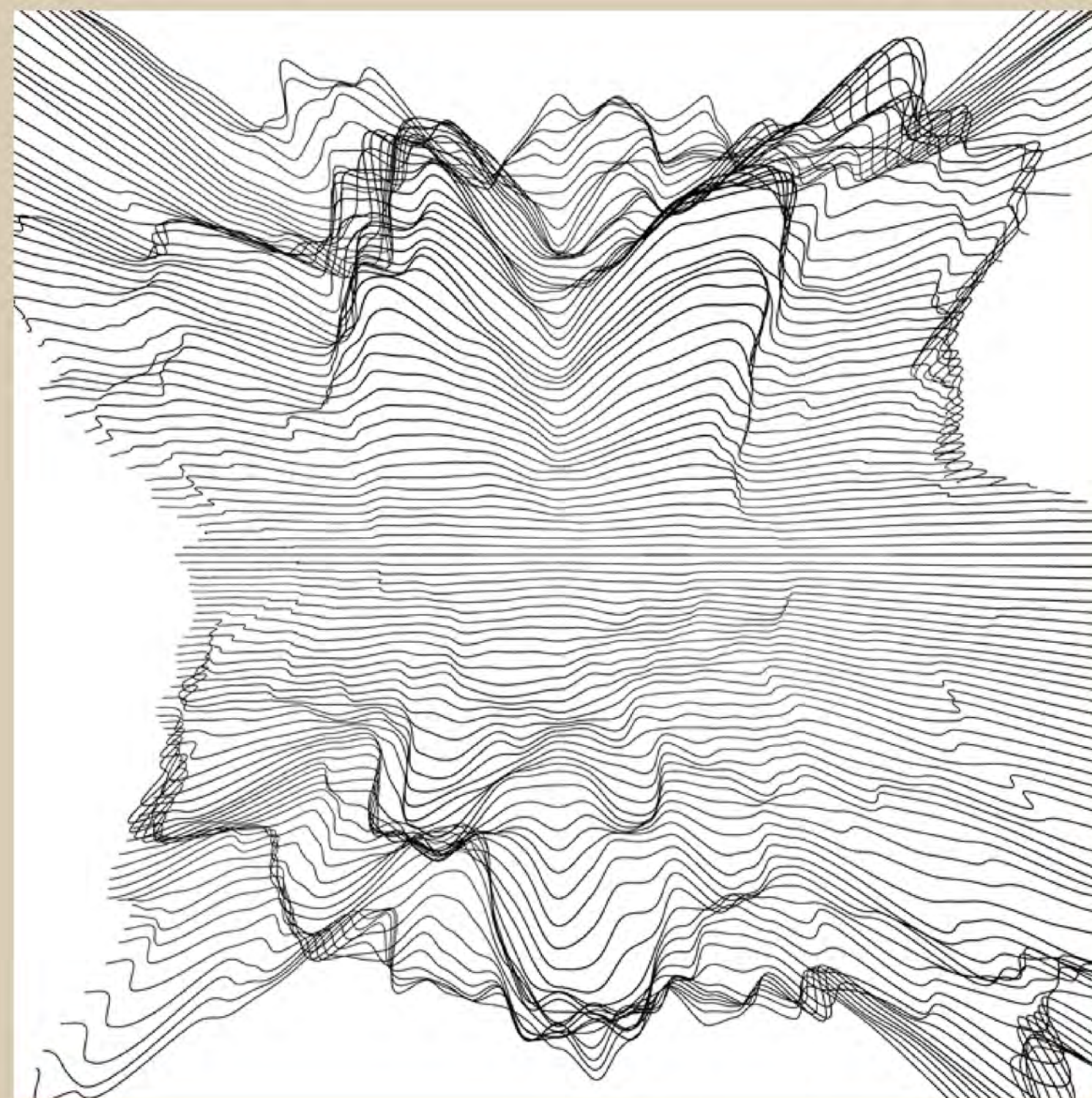


Figura 30 - Sem título, Beatriz Costa, 2023. Imagem gerada por código generativo. Acervo pessoal.

Após encerrar a base da imagem, que seria o conjunto de linhas, procurei adicionar algum dos outros fatores que me interessavam esteticamente, como a questão botânica e geométrica. Durante a pesquisa, aprendi a adicionar múltiplos itens de uma vez através da página de referências¹⁰ do p5js para a função *Array*. Utilizei essa ferramenta para acrescentar diversos pontos no fundo da imagem final, como pode ser visto na Figura 31. Esse recurso foi retirado posteriormente por não interagir bem esteticamente com os outros itens colocados no código.

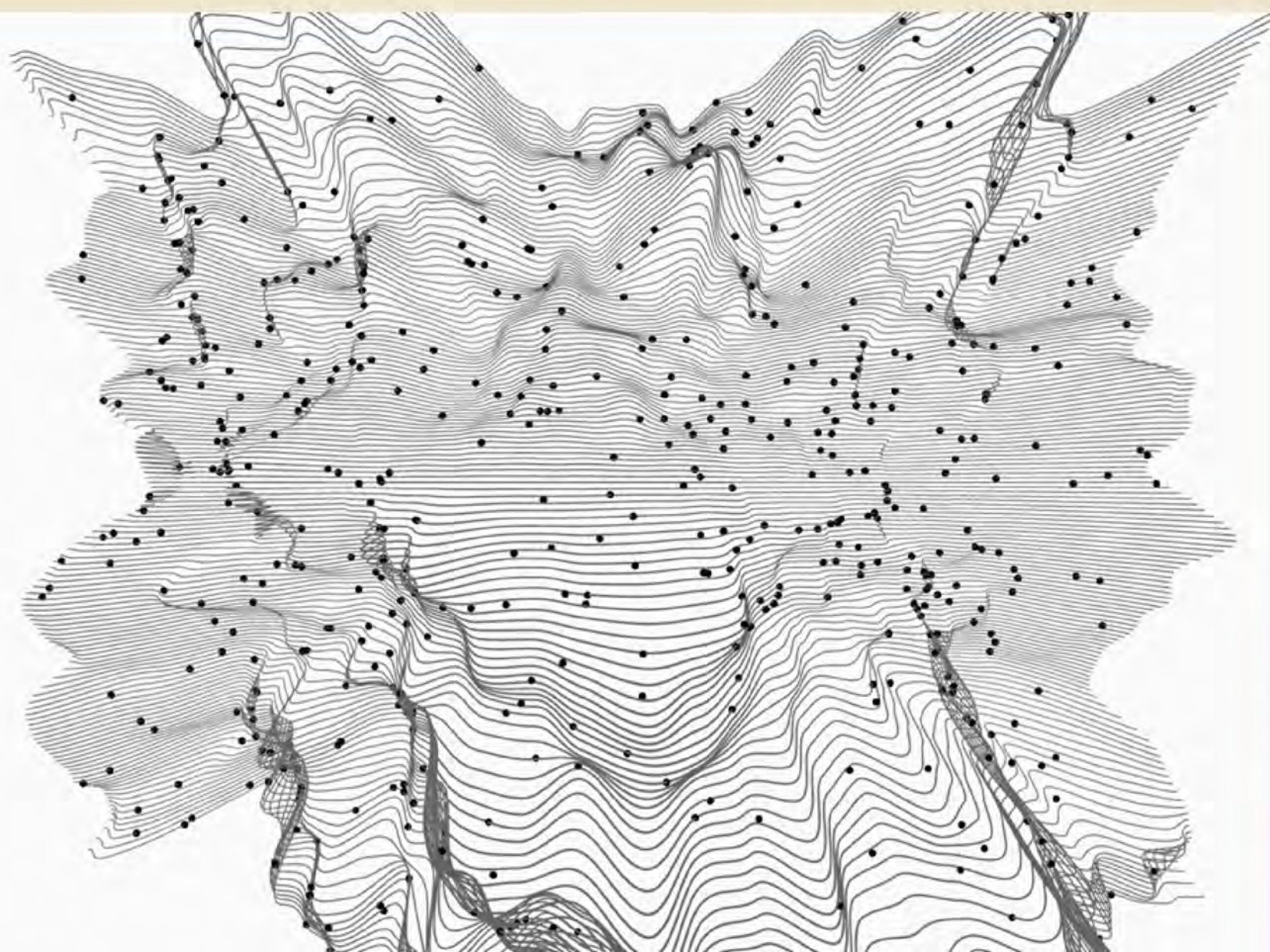


Figura 31 - Sem título, Beatriz Costa, 2023. Imagem gerada por código generativo. Acervo pessoal.

Continuando a busca por elementos botânicos e geométricos, adicionei elipses para simular pétalas que iriam mudar de posição cada vez que o código fosse acionado. Essas formas teriam alguma transparência para não interferir na construção das ondas no fundo, e estar presente através da sobreposição. Os parâmetros da elipse utilizavam um fator *random* para mudar a forma, levando-a mais 'aberta' (Figura 32), se aproximando de um círculo, ou mais 'fechada' (Figura 33), ficando próxima de uma linha ou faixa. Após mudar e testar alguns parâmetros, tive a vontade de ver aquela forma como um círculo, ao invés de uma elipse. Então decidi mudar esta forma para um círculo perfeito.

Aqui, o uso do *random* para criar várias possibilidades voltou já que o círculo tem uma forma constante, diferente da elipse que admitia maior variação em sua construção. Para ajudar com isso, testei adicionar uma função de *random* em um intervalo de tamanhos do círculo, e através de testes, o resultado mais interessante foi de círculos que variavam de $\frac{1}{3}$ até $\frac{1}{2}$ do tamanho da tela.

¹⁰ Página com explicação e exemplos da ferramenta Array: <https://p5js.org/examples/arrays-array.html>

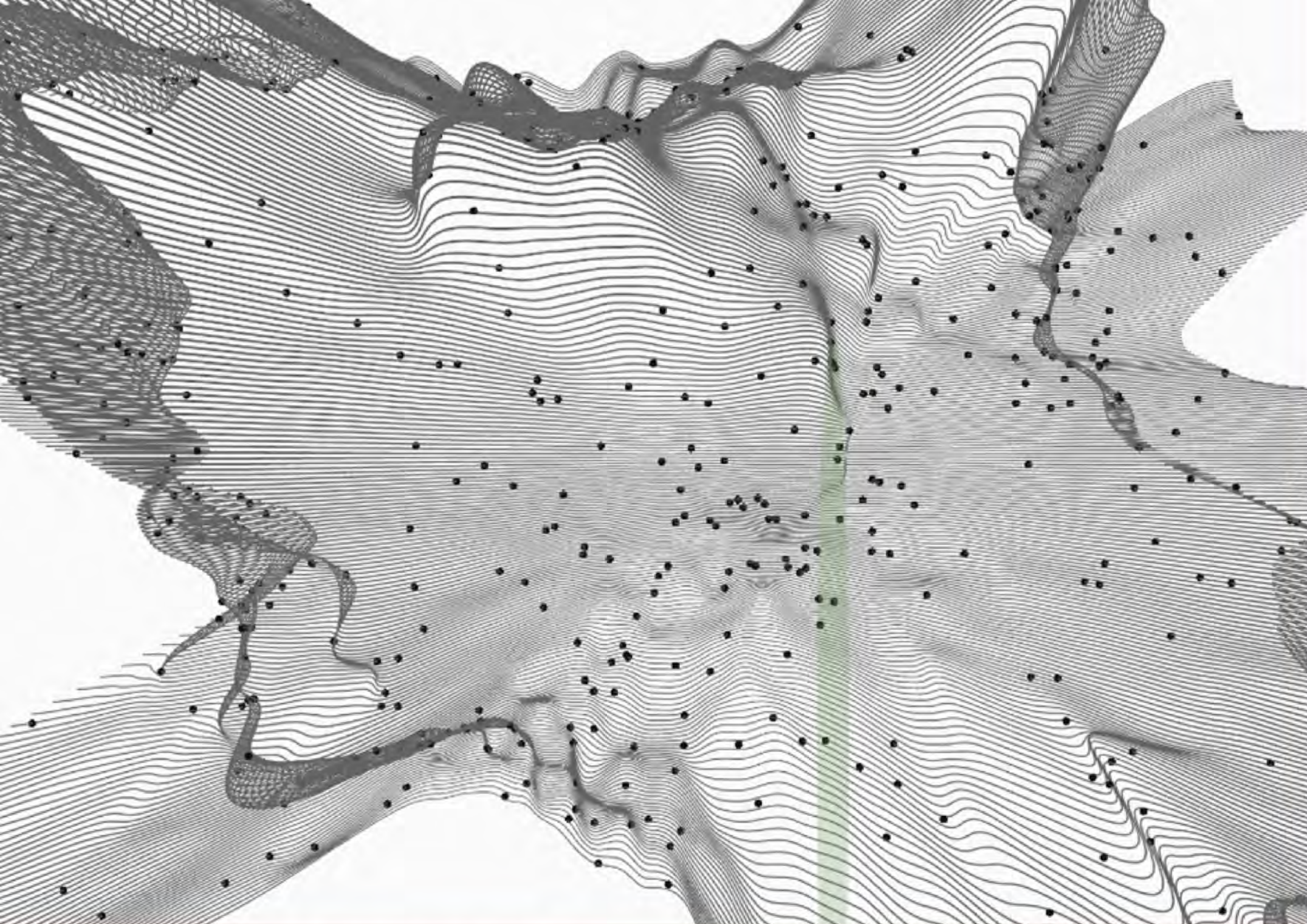


Figura 32 - Sem título, Beatriz Costa, 2023. Imagem gerada por código generativo. Acervo pessoal.

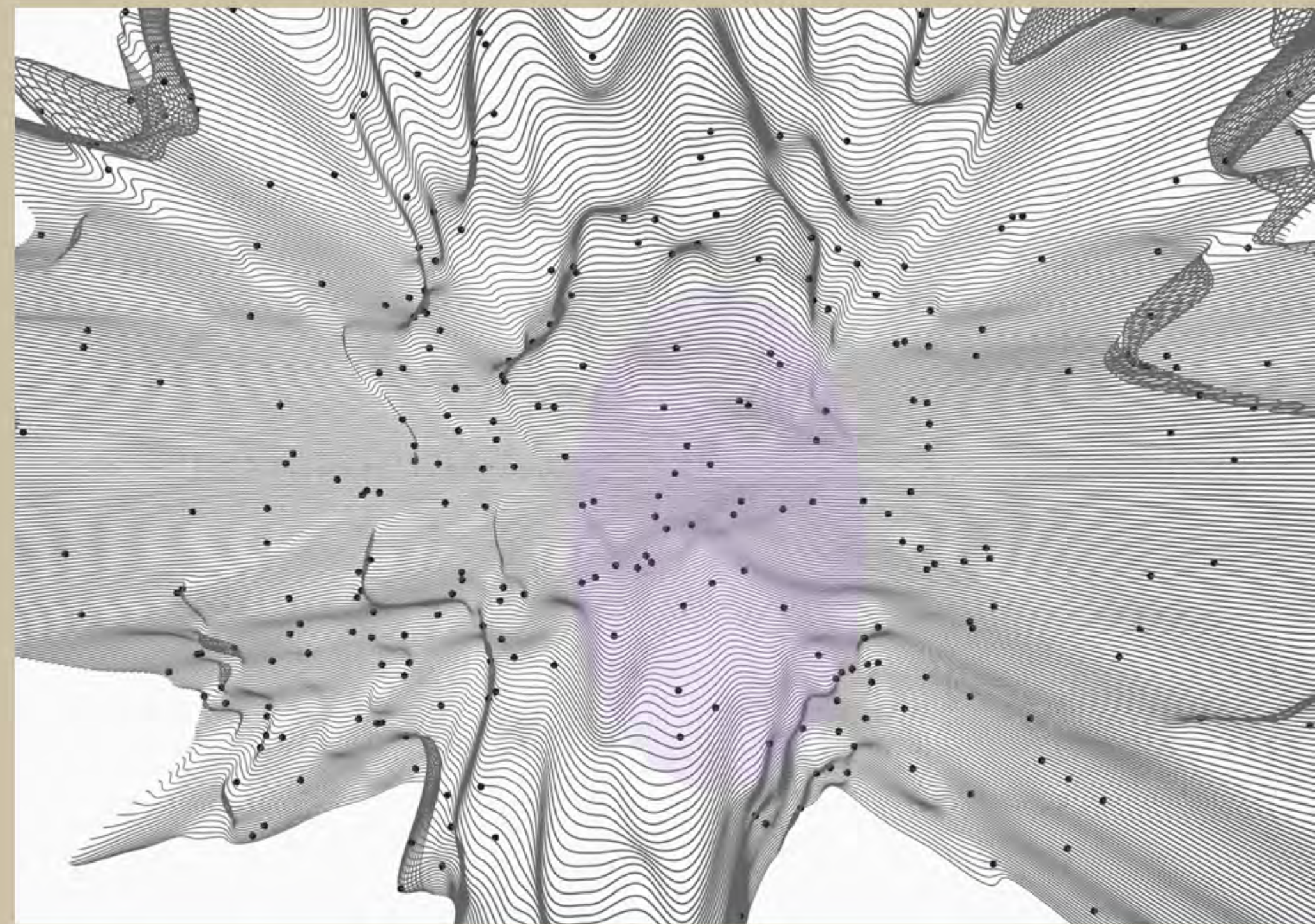


Figura 33 - Sem título, Beatriz Costa, 2023. Imagem gerada por código generativo. Acervo pessoal.

Adicionando a variação de tamanhos com o posicionamento aleatório já presente anteriormente, as imagens começaram a se formar com maior diferença e variações entre elas, como pode ser visto na Figura 34.

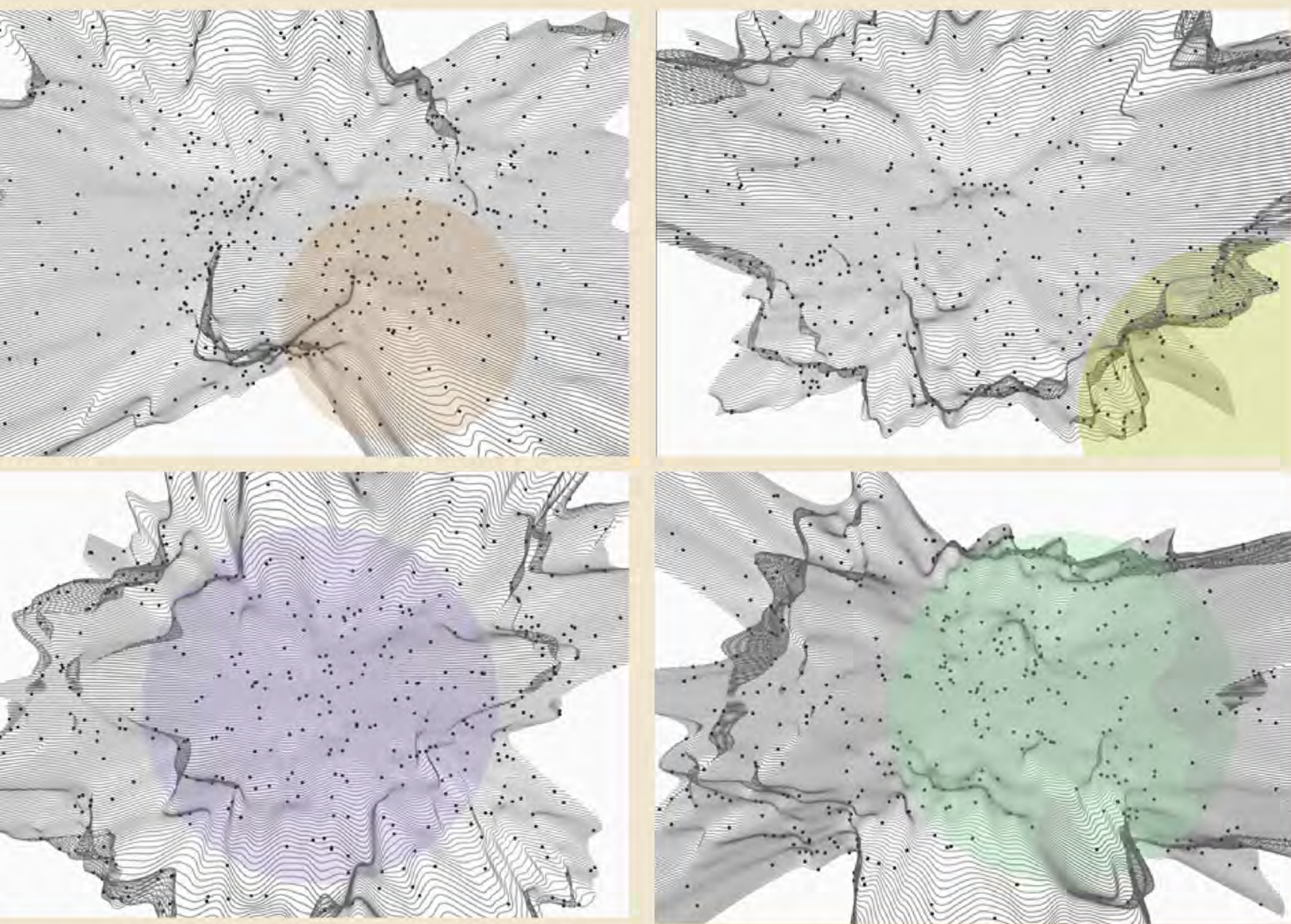


Figura 34 - Sem título, Beatriz Costa, 2023. Imagem gerada por código generativo. Acervo pessoal.

Um fator que não tinha sido explorado nos itens anteriores é o das cores. Com esse objetivo, considerei adicionar um esquema elaborado com matizes do meu interesse e talvez com vários 'temas' - cores quentes, frias, paletas de opostos etc. Mas ao retornar a ideia da minha proposta inicial, pareceu mais coerente abdicar do controle das cores e deixar novamente esta 'decisão' para o fator *random* do código. Testei então adicionar, no esquema RGB valores aleatórios para cada cor, continuando com a transparência que estava presente desde o início. O resultado dessa aplicação também pode ser visto na Figura 34, e a variação de cores trouxe um aspecto singular para a imagem. Nas ondas, existe uma certa previsibilidade nos resultados, algo parecido entre as imagens, com uma diferença mais sutil. Com o círculo, os 3 parâmetros de cores - Vermelho, Verde e Azul - são decididos de forma aleatória e se misturam, formando uma cor final quase imprevisível. Esse comportamento que gera a cor seria um guia para os próximos passos da criação do código.

A partir deste momento, foram realizadas diversas tentativas para adicionar formas parecidas com pétalas na imagem que não foram bem-sucedidas. Em algumas pesquisas, encontrei a criação de Dan¹¹ que utilizou curvas Bézier para criar florais com coloração inspirada em aquarelas (Figura 35), sendo que seu objetivo final era criar um conjunto com essas formas e imprimir com uma plotadora (Figura 36)

Figura 35 -
Generative Flower
Pen Plot with
Watercolor, Dan,
2022. Flores criadas
por códigos no p5.js
<<https://www.seekingoutside.com/using-bezier-curves-in-p5-js-to-make-a-generative-flower-pen-plot-with-watercolor/>> acesso em julho de 2023.



Figura 36 - Generative Flower Pen Plot with Watercolor, Dan, 2022. Flores criadas por códigos e impressas por plotadora. <<https://www.seekingoutside.com/using-bezier-curves-in-p5-js-to-make-a-generative-flower-pen-plot-with-watercolor/>> acesso em julho de 2023.

¹¹ O autor destas composições com curvas Bézier utiliza apenas o nome “Dan” em suas obras e compartilha experimentos e processo criativo em um blog pessoal que pode ser encontrado em: <https://www.seekingoutside.com>

A partir do código de Dan, modifiquei algumas questões como a linha preta no contorno original e o esquema de cores, assim como a transparência dos itens. A linha preta foi retirada e o esquema de cores foi construído assim como no círculo, citado anteriormente, com a função aleatória em RGB. Para criar um efeito diferente nesses itens, decidi mudar o modo de imagem para *MULTIPLY*, a fim de criar um efeito em que as cores se alterassem quando sobrepostas (Figura 37). Apesar deste resultado me agradar muito, quando juntei com as linhas e o círculo que estariam presentes no fundo da imagem, a composição não ficou harmônica, mesmo diminuindo a quantidade de pétalas ou seu tamanho.

Após pesquisar outras opções de flores ou pétalas, resolvi testar a flor dos primeiros experimentos indicada na Figura 19. Ao colocá-la no centro das ondas, o conjunto passou a lembrar uma forma botânica. De longe, a flor parecia um núcleo e as ondas uma forma solta de pétalas. De perto, a flor inicial se destacava e as ondas serviam como um ‘ambiente’ remetendo a um campo ou o próprio mar. Mudando o tamanho das pétalas e a rotação para ter variação através do *random*, o uso desse núcleo auxiliou na desconstrução daquela imagem muito figurativa presente no primeiro experimento. A Figura 38 mostra as possibilidades testadas para o núcleo de flores até encontrar uma faixa de valores equilibrados. O valor final será decidido através do *random* a partir dessa faixa. As aplicações de cores criadas no círculo foram adicionadas a esse objeto também, com a mesma transparência.



Figura 37 - Generative Flower Pen Plot with Watercolor, Dan, modificado por Beatriz Costa, 2023. Imagem criada por código generativo. <<https://editor.p5js.org/beacoxta/sketches/NpDqFpIZj>> Acesso em julho de 2023.

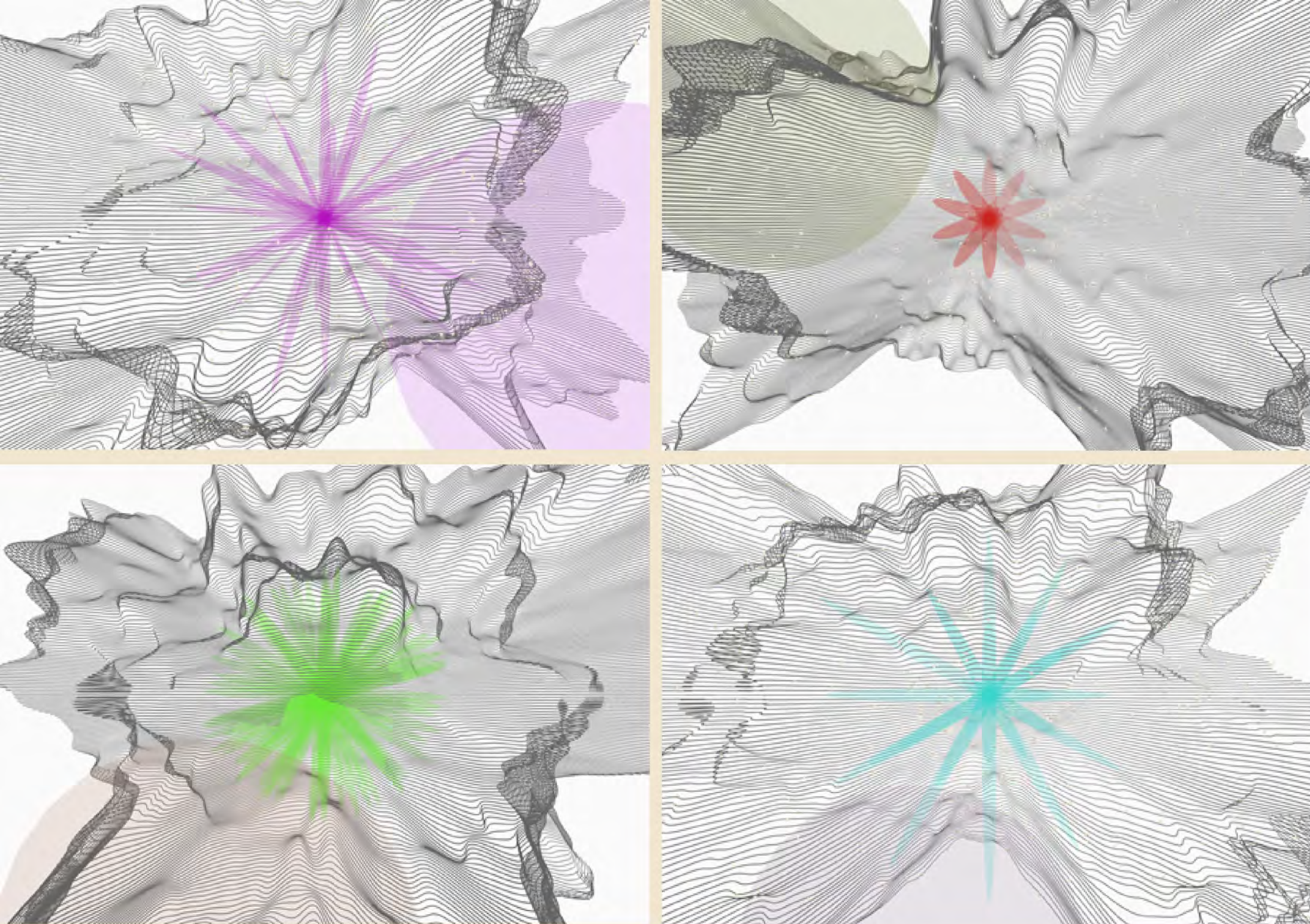


Figura 38 - Sem título, Beatriz Costa, 2023. Imagem gerada por código generativo. Acervo pessoal.

Para testar outras combinações de cores do fundo e das ondas, foram realizados alguns experimentos limitando valores, já que deixando as decisões completamente no *random* fez com que alguns itens ‘desaparecessem’ por falta de contraste Figuras 39, 40 e 41. Aqui ressalto que o item da flor foi retirado para teste, mas retornou logo depois.

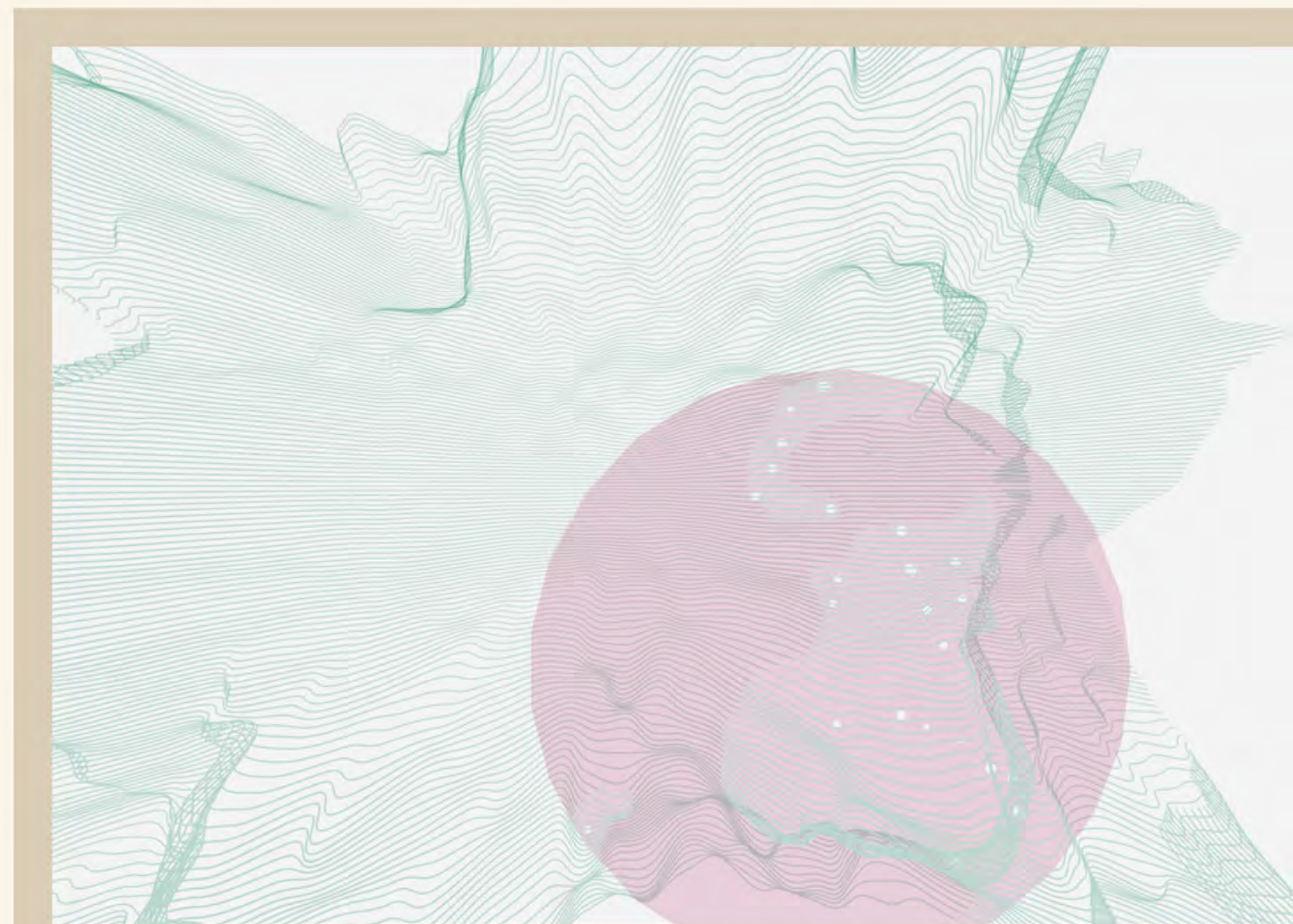


Figura 39 - Sem título, Beatriz Costa, 2023. Imagem gerada por código generativo. Acervo pessoal.



Figura 40 - Sem título, Beatriz Costa, 2023. Imagem gerada por código generativo. Acervo pessoal.



Figura 41 - Sem título, Beatriz Costa, 2023. Imagem gerada por código generativo. Acervo pessoal.

Para o grupo de cores ‘controlado’ selecionei uma faixa de cores que ficariam escuras para as ondas e o fundo da imagem em um tom de cinza constante. A Figura 42 é um exemplo deste momento no processo. A decisão que ficou foi a de manter o cinza escuro por apresentar mais contraste tanto em relação ao fundo quanto aos outros elementos com transparência. Os testes, porém, revelaram a possibilidade de utilizar o fundo escuro com elementos claros, o que resultou na criação da ‘versão escura’ ou oposta do código original (Figura 43). Não senti a necessidade de escolher entre uma ou outra versão, portanto todas as características adicionadas no futuro seriam compatíveis com as duas opções de código.

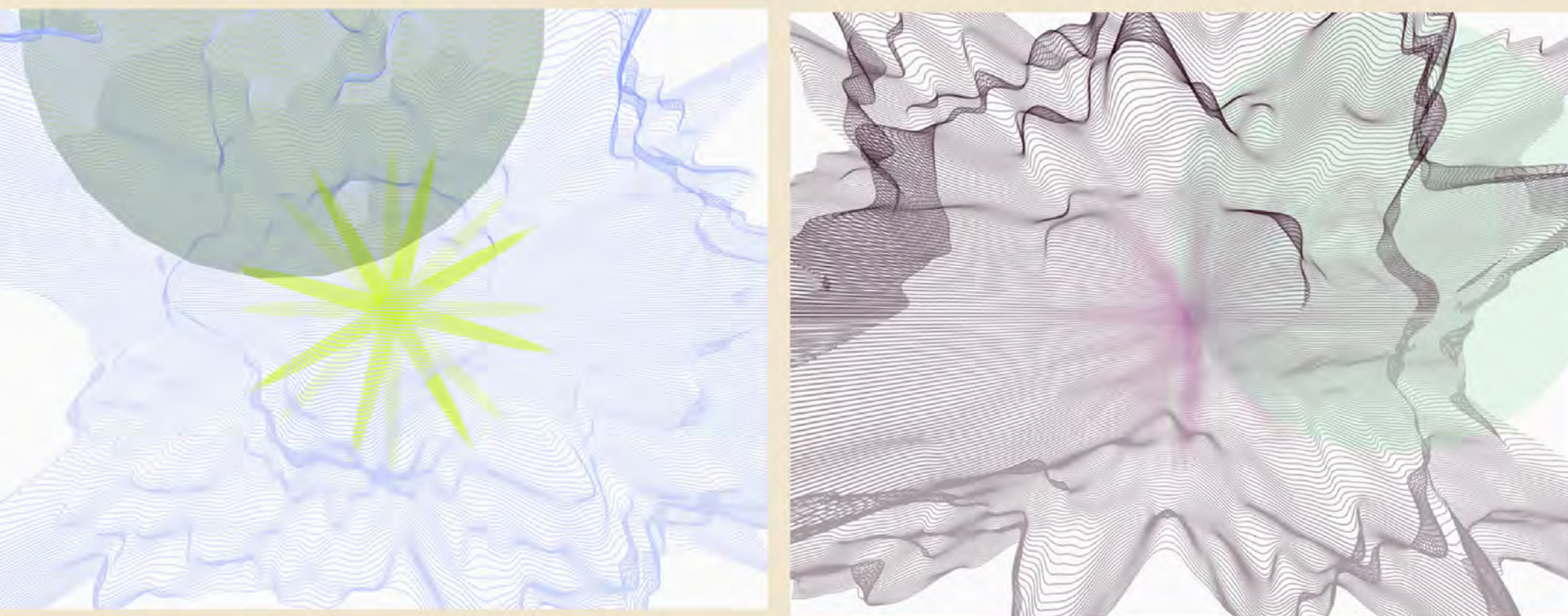


Figura 42 - Sem título, Beatriz Costa, 2023. Imagem gerada por código generativo. Acervo pessoal.

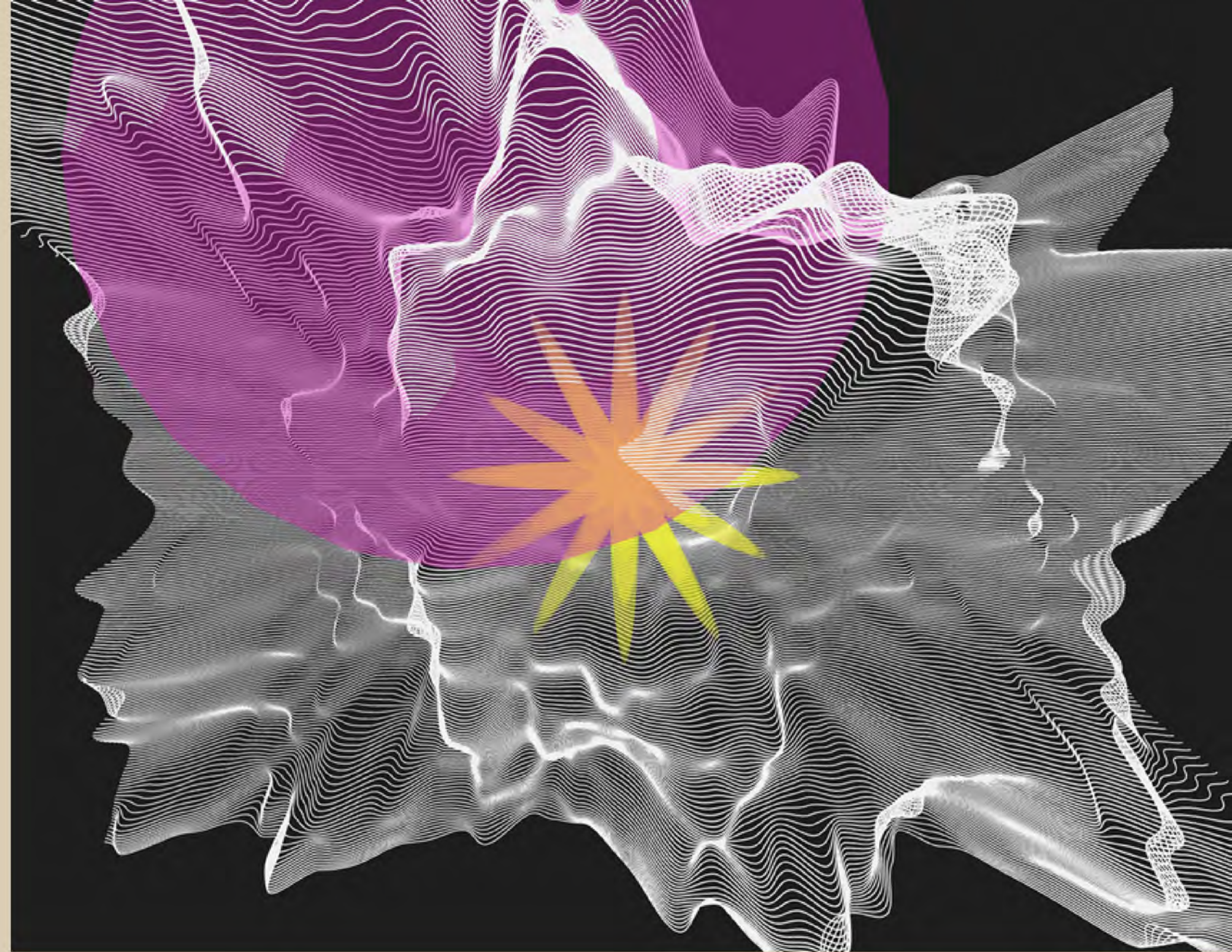


Figura 43 - Sem título, Beatriz Costa, 2023. Imagem gerada por código generativo. Acervo pessoal.

Para as adições finais, resolvi experimentar acrescentar um efeito granulado como textura. Já que o código foi criado no espaço 3D, com o *WEBGL*, as técnicas comuns de manipulação de píxeis não funcionam. Desta forma, utilizei os *shaders*, uma ferramenta mais recente e complexa do p5.js. Os *shaders* são programas executados na GPU do computador que permitem a criação de ‘filtros’ ou tarefas que exigiriam a manipulação de muitos píxeis de uma vez só e que normalmente levariam muito tempo. Como referência, utilizei o código de Barney Codes¹² no YouTube, um programador que faz tutoriais utilizando o *Processing* e p5.js. Com as direções dadas por ele e recursos presentes no próprio site do p5.js, consegui adicionar o efeito granulado na imagem. As Figuras 44 e 45 mostram as duas versões do código com o efeito adicionado.

¹² O canal com tutoriais no YouTube se encontra em: <https://www.youtube.com/@BarneyCodes/videos>; A referência do p5.js se encontra em: <https://p5js.org/learn/getting-started-in-webgl-shaders.html>

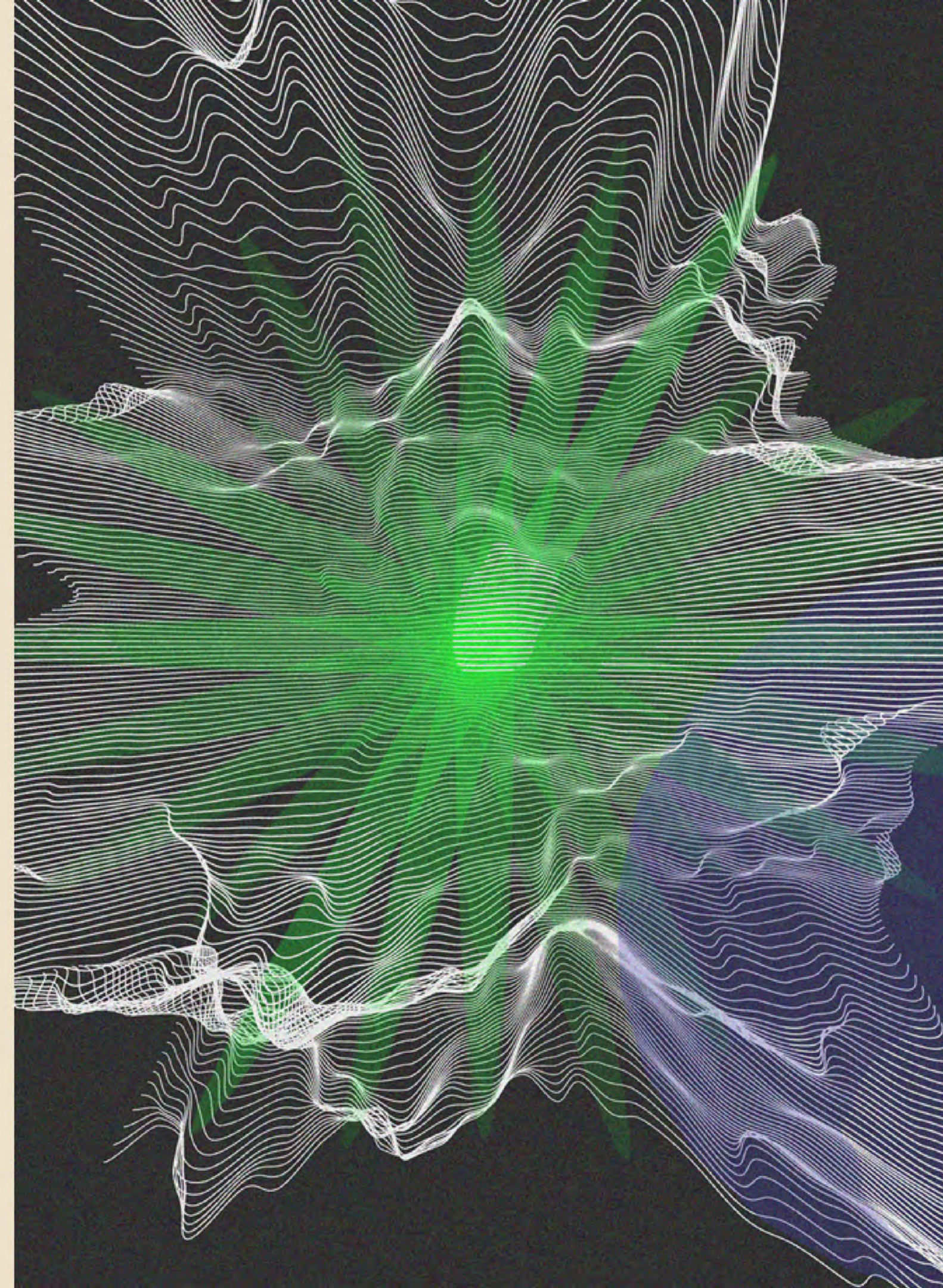


Figura 44 - Sem título, Beatriz Costa, 2023. Imagem gerada por código generativo. Acervo pessoal.

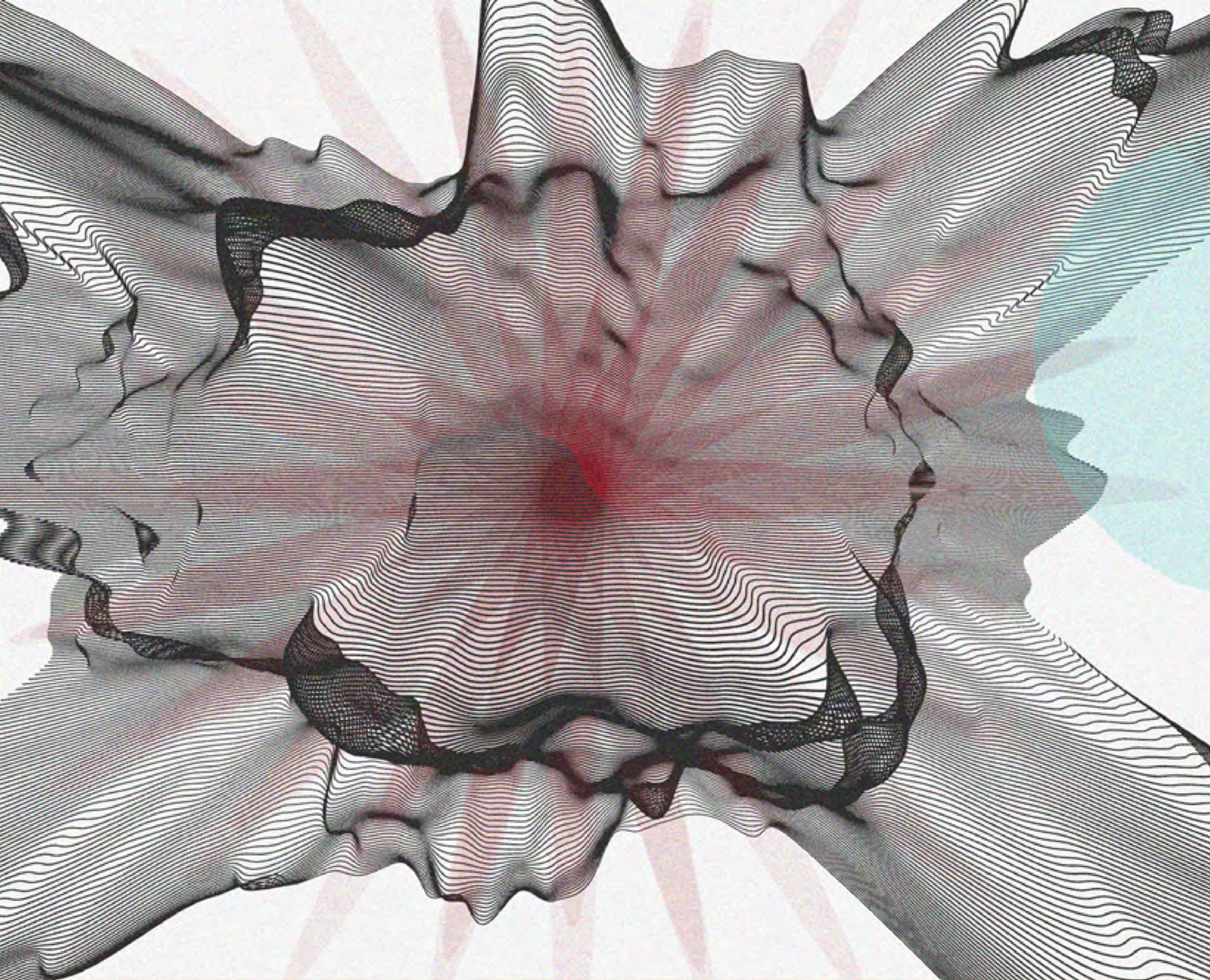


Figura 45 - Sem título, Beatriz Costa, 2023. Imagem gerada por código generativo. Acervo pessoal.

Explorando mais o canal de Barney Codes encontrei um tutorial de como fazer árvores fractais e achei que seria outro bom item para adicionar nos cantos da tela tanto como uma textura quando algo para remeter ao tema botânico. A árvore fractal também esteve presente na pesquisa do capítulo 2, nos trabalhos de Suzete Venturelli (Figuras 17 e 18) e Mônica Rizzolli (Figura 16). Tanto o código dado por Barney quanto outros que encontrei são para árvores fractais chamadas recursivas, que por natureza vão 'abrindo' os galhos e no final se transformam em um triângulo com a medida fractal, como pode ser visto na Figura 46. Esse efeito não me interessava então, com ajuda do ChatGPT adicionei uma função de mínimo e máximo de galhos - pelo menos 3 galhos e no máximo 10 para não haver a curvatura - e estabeleci a quantidade de árvores para até 3, podendo assim ter de 0 a 3 árvores quando o código fosse acionado. Todos esses parâmetros são selecionados com a função *random*. A Figura 47 mostra duas possibilidades de resultado desse código modificado. O ChatGPT foi utilizado em algumas partes da criação do trabalho e essa relação com a tecnologia será discutida no final deste capítulo.

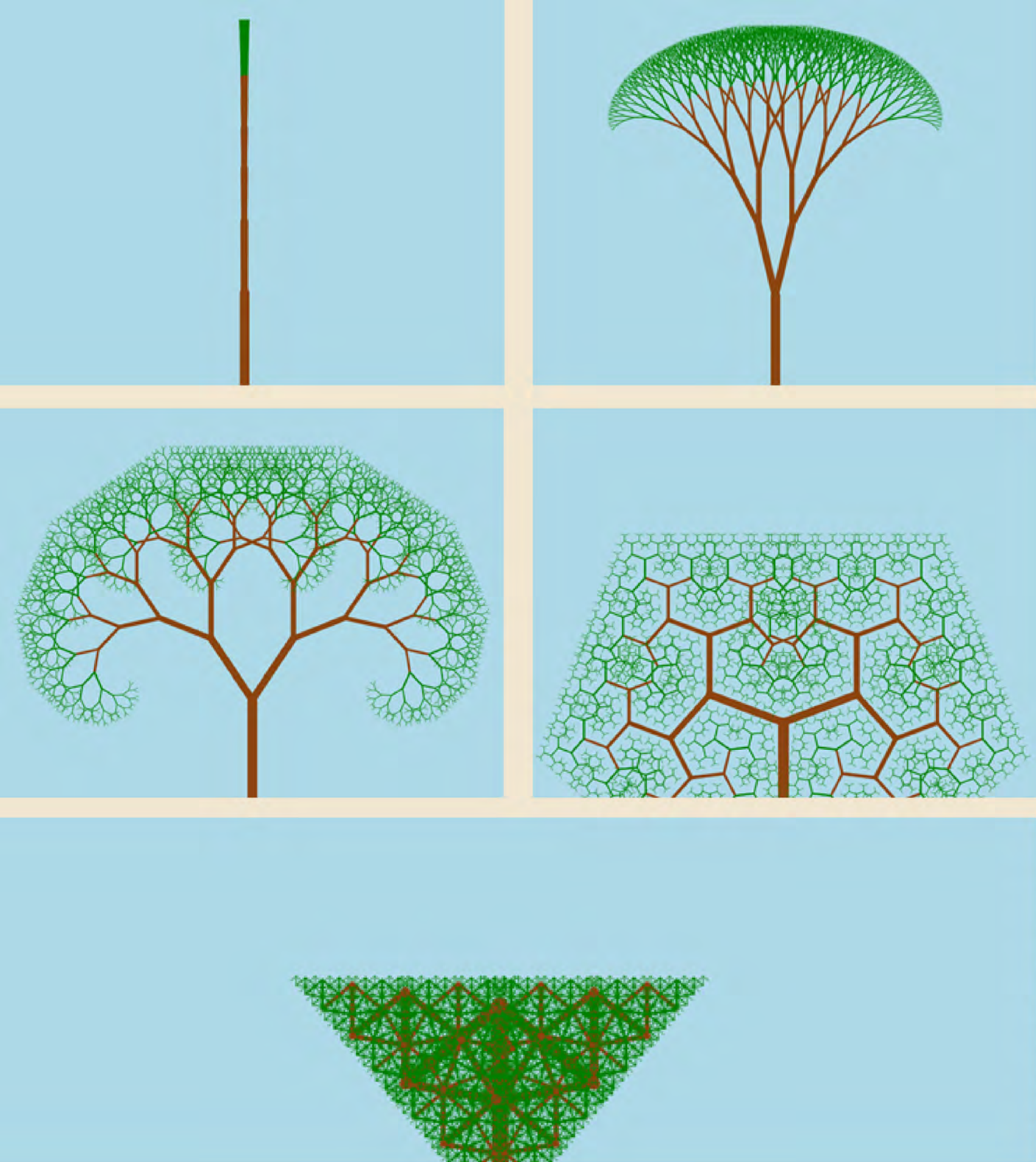


Figura 46 - Fractal Fun In Javascript, Mark Davis, 2018. Imagem interativa de uma árvore fractal gerada por código. <https://coderdojoathenry.org/2018/04/17/fractal-fun-in-javascript/> acesso em julho de 2023.

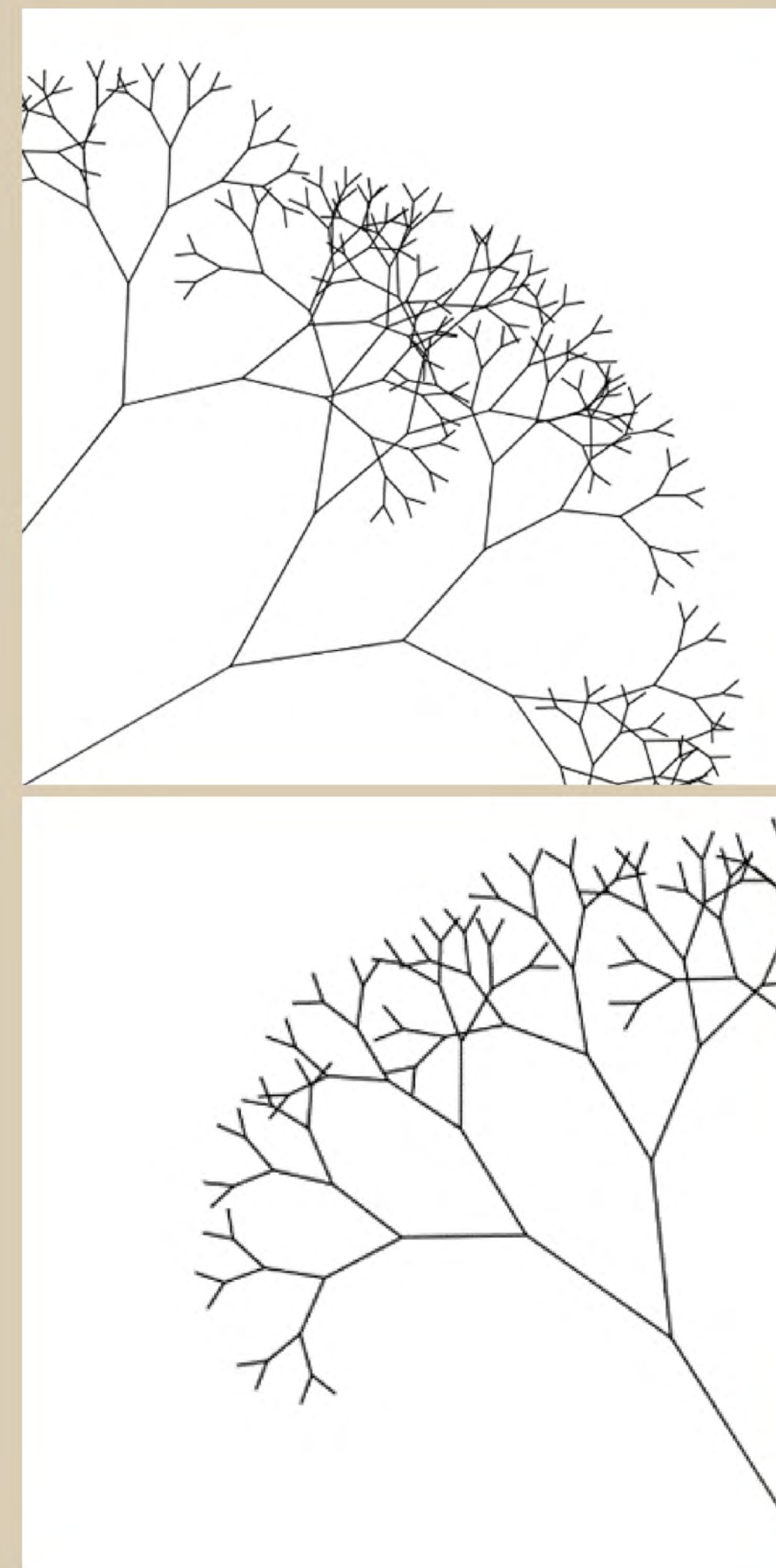


Figura 47 - Sem título, Beatriz Costa, 2023. Imagem gerada por código generativo. Acervo pessoal.

O código da Figura 47 foi criado em 2D e quando transportado para o código em 3D houve uma mudança na disposição das árvores. Em duas dimensões, o ponto (0,0) se encontra no canto esquerdo. Já no ambiente *WEBGL*, o ponto (0,0,0) se encontra no centro da imagem. Como as árvores foram criadas para variar a partir do ponto inicial em 2D, quando transportadas para o 3D, o ponto inicial se tornou outro, e as árvores mudaram de disposição na tela. As Figuras 48 e 49 mostram algumas das variações da construção das árvores fractais no código final.

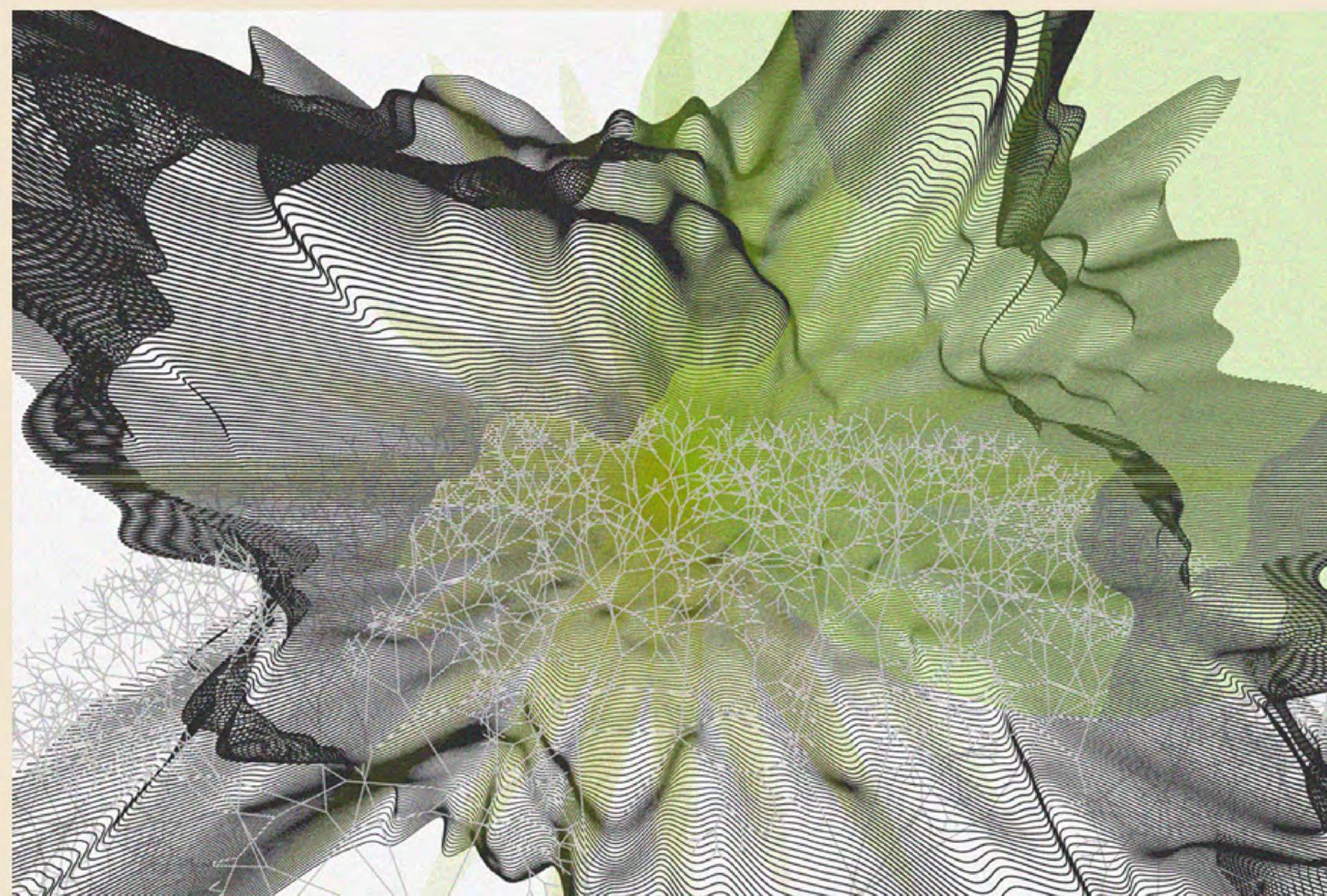
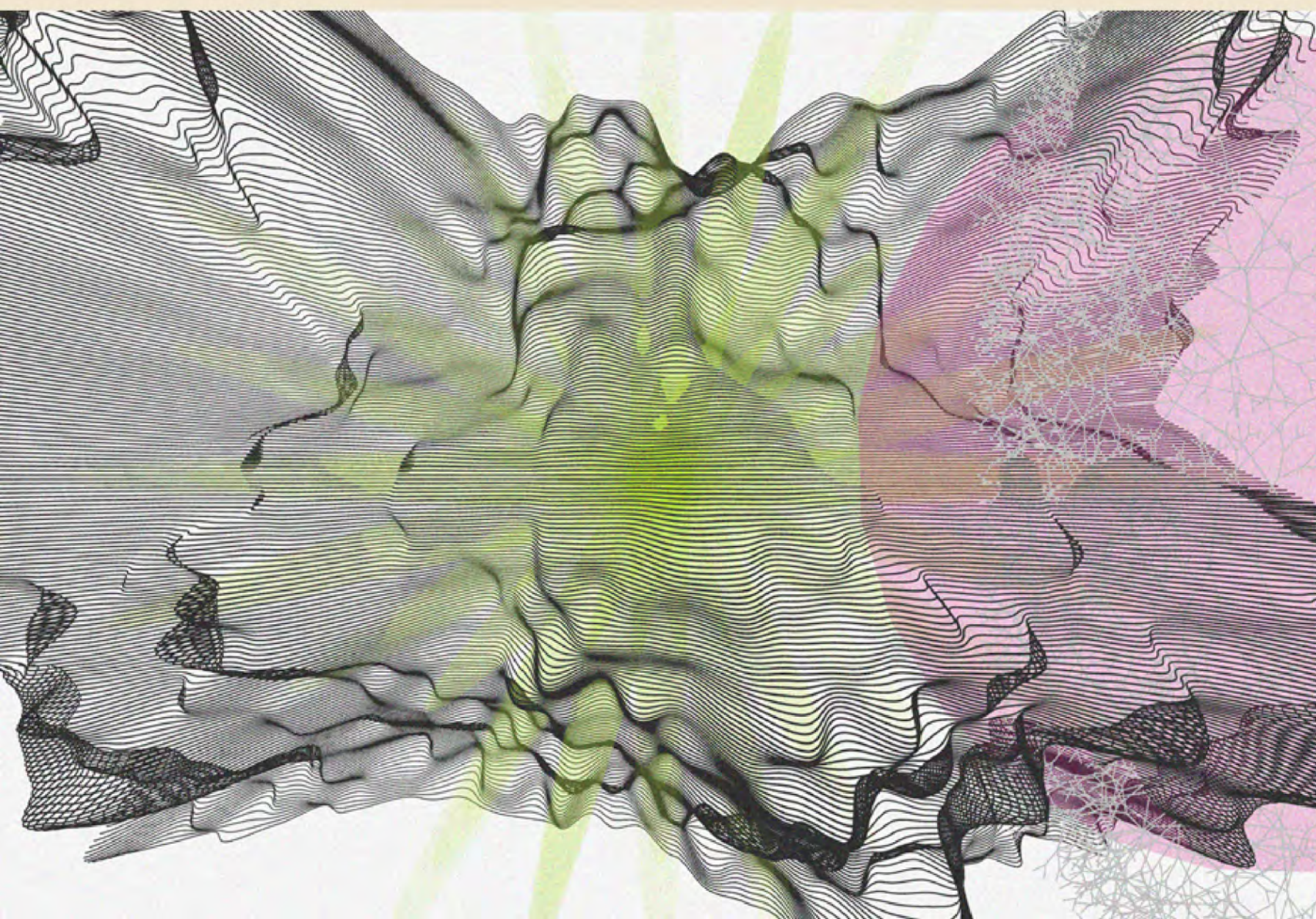
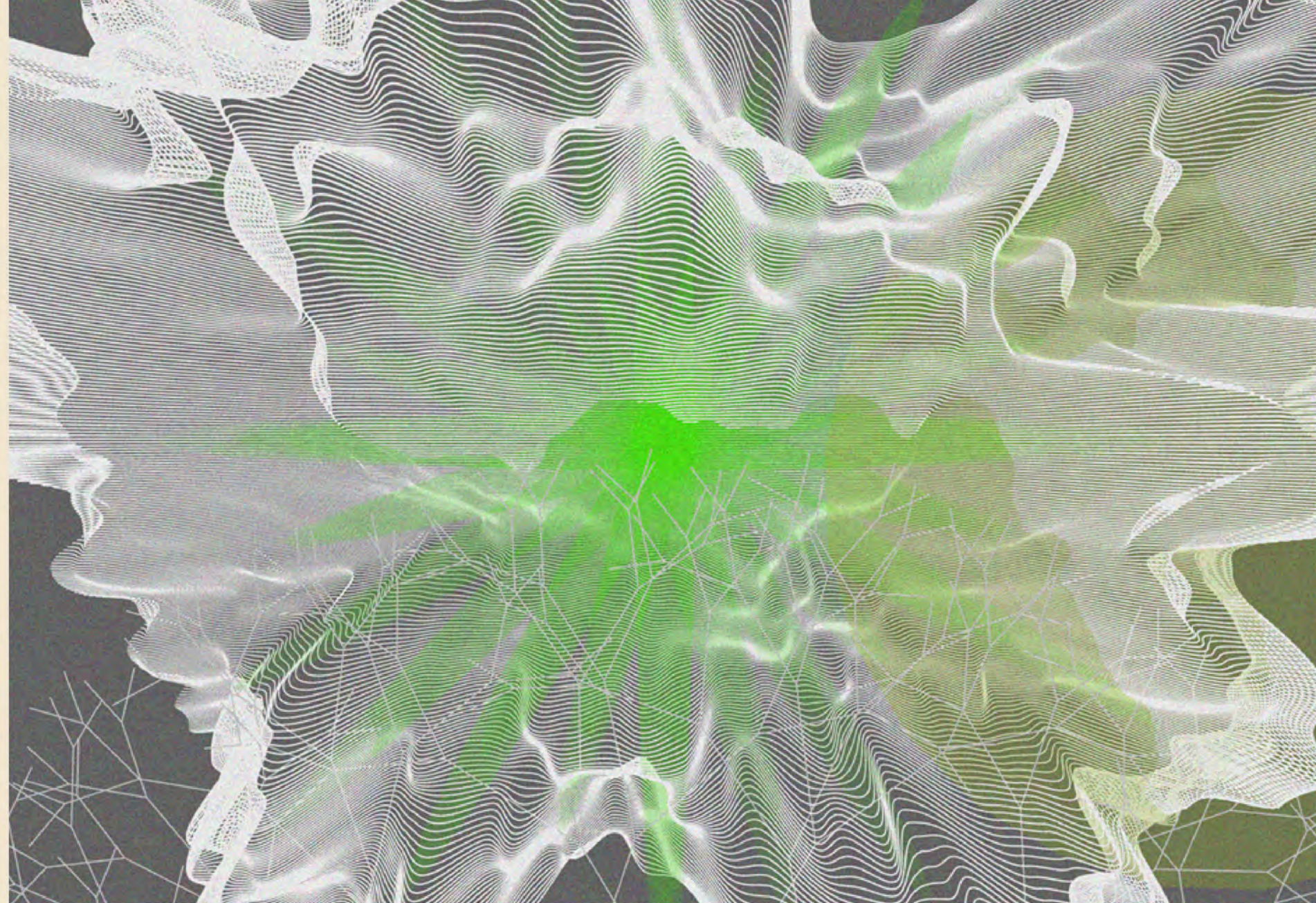


Figura 48 -
Sem título,
Beatriz Costa,
2023. Imagem
gerada por
código
generativo.
Acervo
pessoal.

Figura 49 -
Sem título,
Beatriz Costa,
2023. Imagem
gerada por
código
generativo.
Acervo
pessoal.



Uma vez que todos os itens foram colocados no código, foram iniciados testes a fim de decidir a orientação – vertical ou horizontal – e o tamanho final da imagem. Até o momento, os experimentos foram feitos em uma tela de aproximadamente 800x600 pixels para facilitar o processo de criação. A Figura 50 mostra um conjunto de resultados dos testes de orientação. A decisão foi difícil, pois cada opção dá um efeito diferente e não há necessariamente um item melhor do que o outro.

Dessa forma, a escolha levou em conta as referências iniciais de elementos botânicos, destacando as opções quadradas por apresentar ondas uniformes que podem ser interpretadas, na imagem geral, como flores. As opções retangulares perdem essa característica que considero importante para as camadas de significado do trabalho. Em relação ao tamanho final, considere as dimensões de uma tela de computador comum que tem 1080px de altura e dupliquei essa medida para que as imagens possam ser vistas em seus detalhes sem que se tornem pixeladas ou de baixa qualidade.

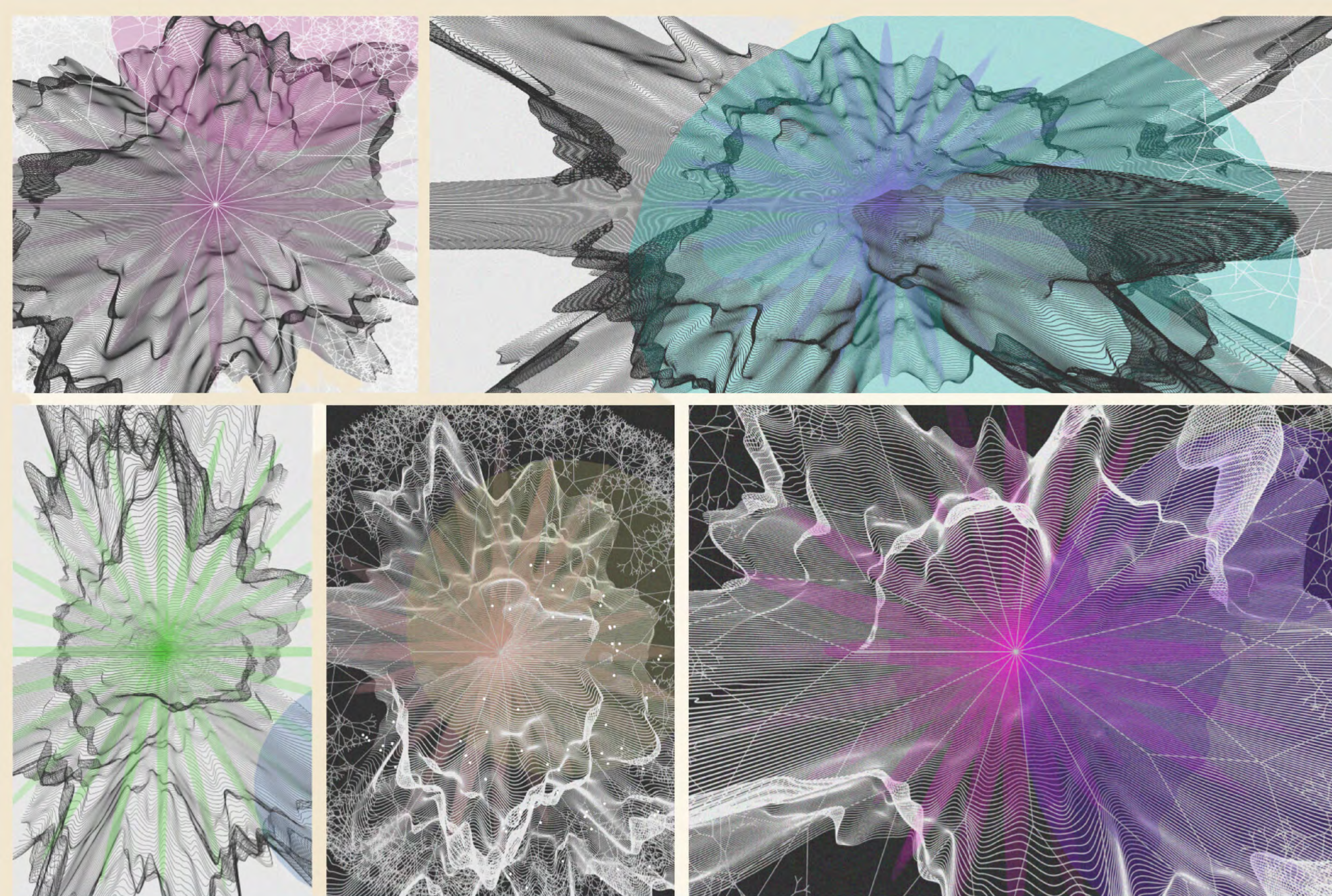


Figura 50 - Sem
título, Beatriz
Costa, 2023.
Imagem gerada
por código
generativo.
Acervo pessoal.



3.2 O USO DO CHATGPT

A presença da ferramenta de Inteligência Artificial, *ChatGPT*, foi importante no processo criativo já que não possui muito conhecimento em programação. Em alguns momentos houveram erros com o código que eu não sabia resolver e o Chat dizia onde estavam esses erros e como consertá-los, como o exemplo da Figura 51. A ferramenta de Inteligência Artificial foi também uma opção para adquirir conhecimento quando não havia recurso suficiente nas pesquisas feitas durante o processo de criação.

Um grande exemplo foi o da textura *grain* colocada a partir da Figura 44. O *grain* é relativamente fácil de ser colocado quando trabalhado em um ambiente 2D. Inclusive, por dias tentei seguir um artigo¹³ que ensinava três formas de adicionar o *grain* em uma imagem sem saber que não funcionaria no meu código por usar o render *WEBGL*. Depois de colocar o código completo para o *ChatGPT* tentei pesquisar soluções para o código funcionar com a textura. Foi em um desses momentos que a ferramenta de IA afirmou que pro código rodar no *WEBGL* era necessário usar os *shaders*.

¹³ O artigo de título All about that grain escrito pelo usuário “Gorilla Sun” no site fxhash detalha três maneiras de adicionar o efeito grain em um código e pode ser encontrado em: <https://www.fxhash.xyz/article/all-about-that-gain>

B

```
Can you find the error in this code and fix it? let noiseScale = 0.005;
let yStart = 0;
let yStep;
let randSeed;
let minNumOfLines = 200;
let maxNumOfLines = 350;
let myFilterShader;
```



There are multiple errors in the given code. Here's the corrected version:

javascript

Copy code

```
let noiseScale = 0.005;
let yStart = 0;
let yStep;
let randSeed;
let minNumOfLines = 200;
let maxNumOfLines = 350;
let myFilterShader;
let numOfLines; // Declare numOfLines variable
```

Figura 51 - Sem título, Beatriz Costa, 2023. Imagem gerada por código generativo. Acervo pessoal.

B

Can you please add to my code:

- a black and white smoke texture in the background that changes everytime the code is run
- a grain effect on top of the whole code that works on the WebGL renderer

In this code, I've added the necessary functions to generate a black and white smoke texture using Perlin noise. The `'smokeTexture'` is created as a `'p5.Graphics'` object and drawn onto a plane in the WebGL renderer. The grain effect is implemented using a custom shader (`'grainShader'`) that overlays the texture with a grainy pattern.

Please make sure to have the `'grain.vert'` and `'grain.frag'` shader files in the same directory as your HTML file or p5.js sketch. You can customize the grain effect further by modifying the shader code in those files.

Let me know if you have any further questions!

Figura 52 - Sem título, Beatriz Costa, 2023. Imagem gerada por código generativo. Acervo pessoal.

Em uma das conversas, colocada em parte na Figura 52, pedi por uma textura estilo 'fumaça' no fundo da imagem e o efeito *grain*. O Chat respondeu com uma opção de código e complementou explicando como essas adições foram colocadas; este foi meu primeiro contato com o conceito de *shaders*.

O código não funcionou diversas vezes então procurei saber se havia um conceito que eu desconhecia. Para confirmar, perguntei o que é um 'diretório' no p5.js, e a resposta do Chat deixou clara que era necessária a organização do diretório e adição de novos arquivos para o *shader* funcionar corretamente (Figura 53).

B

what is a directory in p5js?



In p5.js, a directory refers to a folder that contains your project files, including your HTML file, JavaScript files, and any additional assets like images, sounds, or shaders. It's a way to organize and structure your project files.

When you create a p5.js sketch, all the files related to that sketch are typically stored in the same directory. This includes the HTML file that displays the sketch in a web browser, the JavaScript file that contains the p5.js code, and any additional files you may need, such as shader files or image assets.

Figura 53 - Sem título, Beatriz Costa, 2023. Imagem gerada por código generativo. Acervo pessoal.

Continuando a explicação, a ferramenta exemplificou os itens que existem em um diretório comum (Figura 54). Dessa forma pude entender o funcionamento básico do conjunto de arquivos existentes e os necessários para realizar as mudanças desejadas.

Apesar da grande ajuda, chegou um momento em que o *ChatGPT* não conseguiu mais ser útil, pois haviam falhas no código dado por ele e falta de conhecimento da minha parte para resolver as questões colocadas. Foi assim que pesquisei mais sobre os *shaders*, encontrando o tutorial de Barney Codes no YouTube que permitiu — finalmente — a mudança presente no código final.

In this example, `index.html` is the HTML file that loads the p5.js sketch and includes any necessary JavaScript and shader files. `sketch.js` is the p5.js code file that contains your sketch's setup, draw, and other functions. The `shaders` folder contains the vertex and fragment shader files used in the sketch, and the `assets` folder contains image and sound files that the sketch may use.

By organizing your project files into directories, it becomes easier to manage and locate specific files within your p5.js project.

Figura 54 - Sem título, Beatriz Costa, 2023. Imagem gerada por código generativo. Acervo pessoal.

B Can you create a code in p5js that has a fractal tree with the starting point at either corner of the canvas? I'd like the corner to change everytime the code is run as well as the amount of branches of the tree, so every time I run the code the tree has a different number of branches

 Certainly! Here's an example code in p5.js that generates a fractal tree with the starting point at a random corner of the canvas and a random number of branches each time it is run:

In this code, the `canvasSize` variable determines the size of the canvas. The `branchMin` and `branchMax` variables define the minimum and maximum number of branches for the fractal tree.

The `setup()` function sets up the canvas and the `draw()` function is called repeatedly to draw the tree.

In the `draw()` function, a random starting corner is selected, and a random number of branches is chosen within the specified range. Based on the selected corner, the starting point is set. Then, the `drawBranch()` function is called to draw the tree.

The `drawBranch()` function takes parameters for the starting coordinates, length, angle, and number of branches. It draws a line from the starting point and recursively calls itself to draw the subsequent branches.

Feel free to modify the code and experiment with different parameters to achieve the desired effect.

Figura 55 - Sem título, Beatriz Costa, 2023. Imagem gerada por código generativo. Acervo pessoal.

Como citado anteriormente, a ferramenta de IA também foi utilizada para criar e alterar o código da árvore fractal, colocado na Figura 55. O pedido específico envolvia adicionar a árvore nos cantos da tela para inicialmente não intervir muito com a imagem das ondas no centro. A Figura 56 mostra um dos momentos em que o *ChatGPT* não foi bem-sucedido na criação, sendo fundamental a troca de informações nessa ‘conversa’.

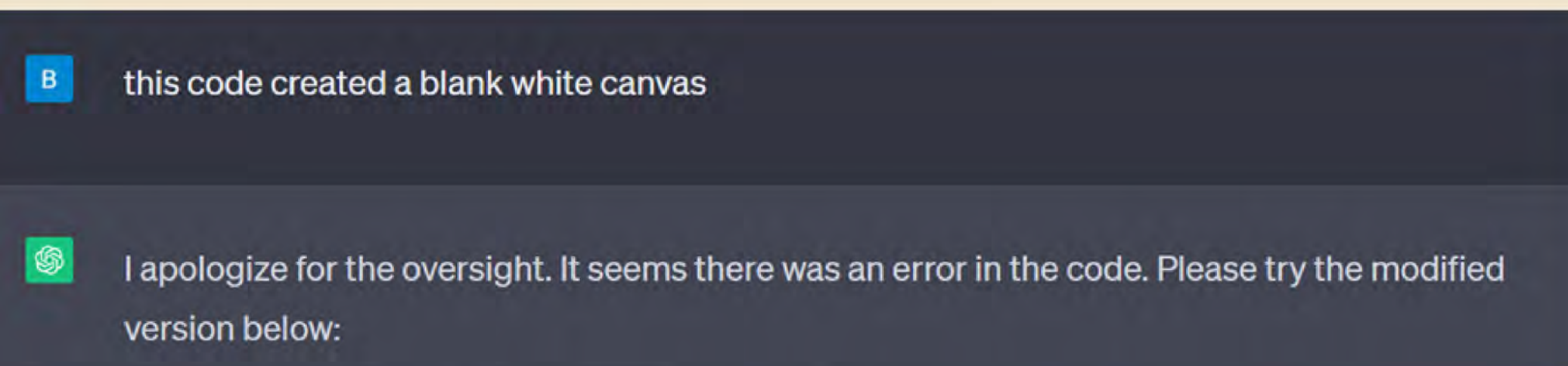


Figura 56 - Sem título, Beatriz Costa, 2023. Imagem gerada por código generativo. Acervo pessoal.

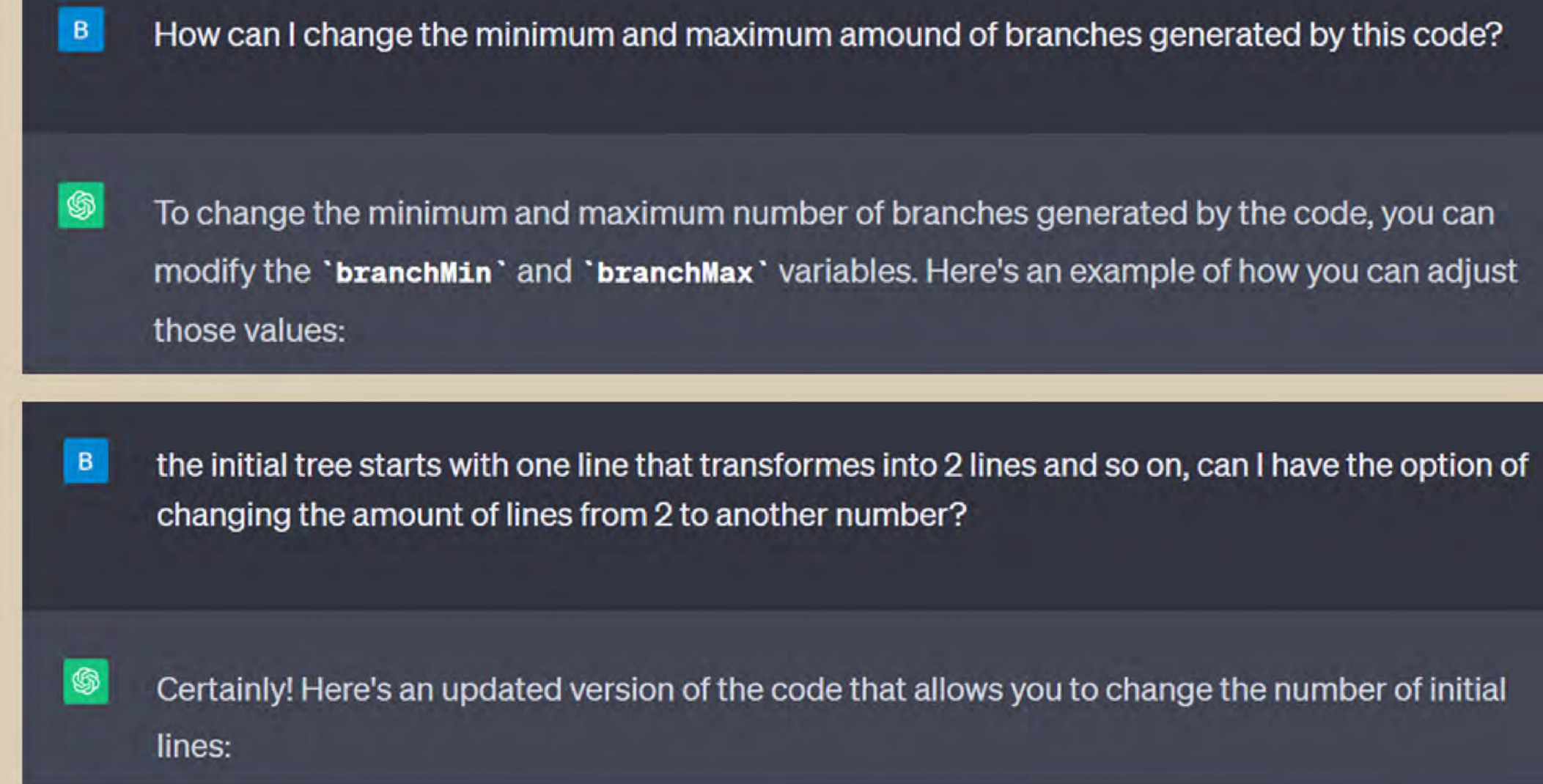


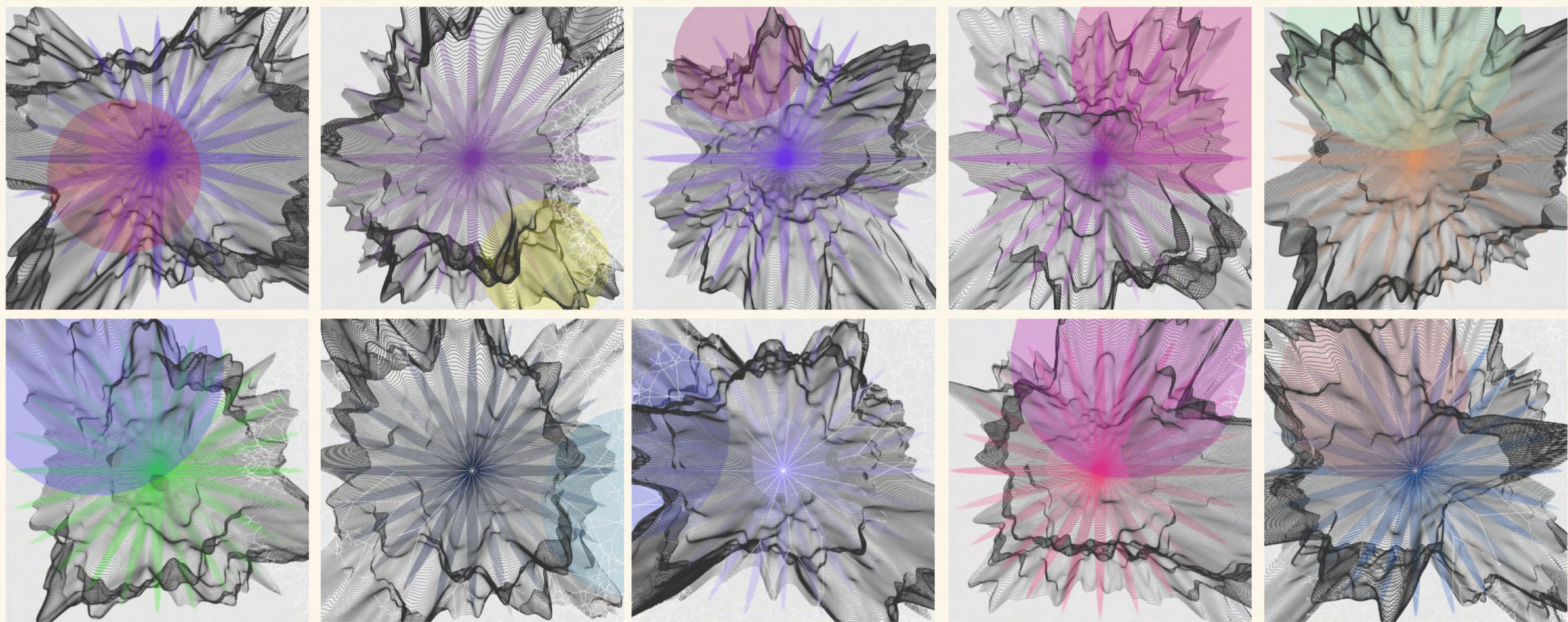
Figura 57 - Sem título, Beatriz Costa, 2023. Imagem gerada por código generativo. Acervo pessoal.

Experimentando com o código de árvores criado decidi adicionar algumas outras variações, como selecionar um número mínimo e máximo de galhos assim como a quantidade de árvores, já que os códigos consultados sempre apresentavam uma árvore principal que ia se formando. O *ChatGPT* também ajudou nessas alterações, como mostrado na Figura 57. Esse detalhe das árvores pode ser visto nas Figuras 48 e 49. É relevante citar que o uso do *ChatGPT*, além de ser útil para o meu processo de criação e aprendizado, também é uma ferramenta generativa, complementando o tema deste trabalho.

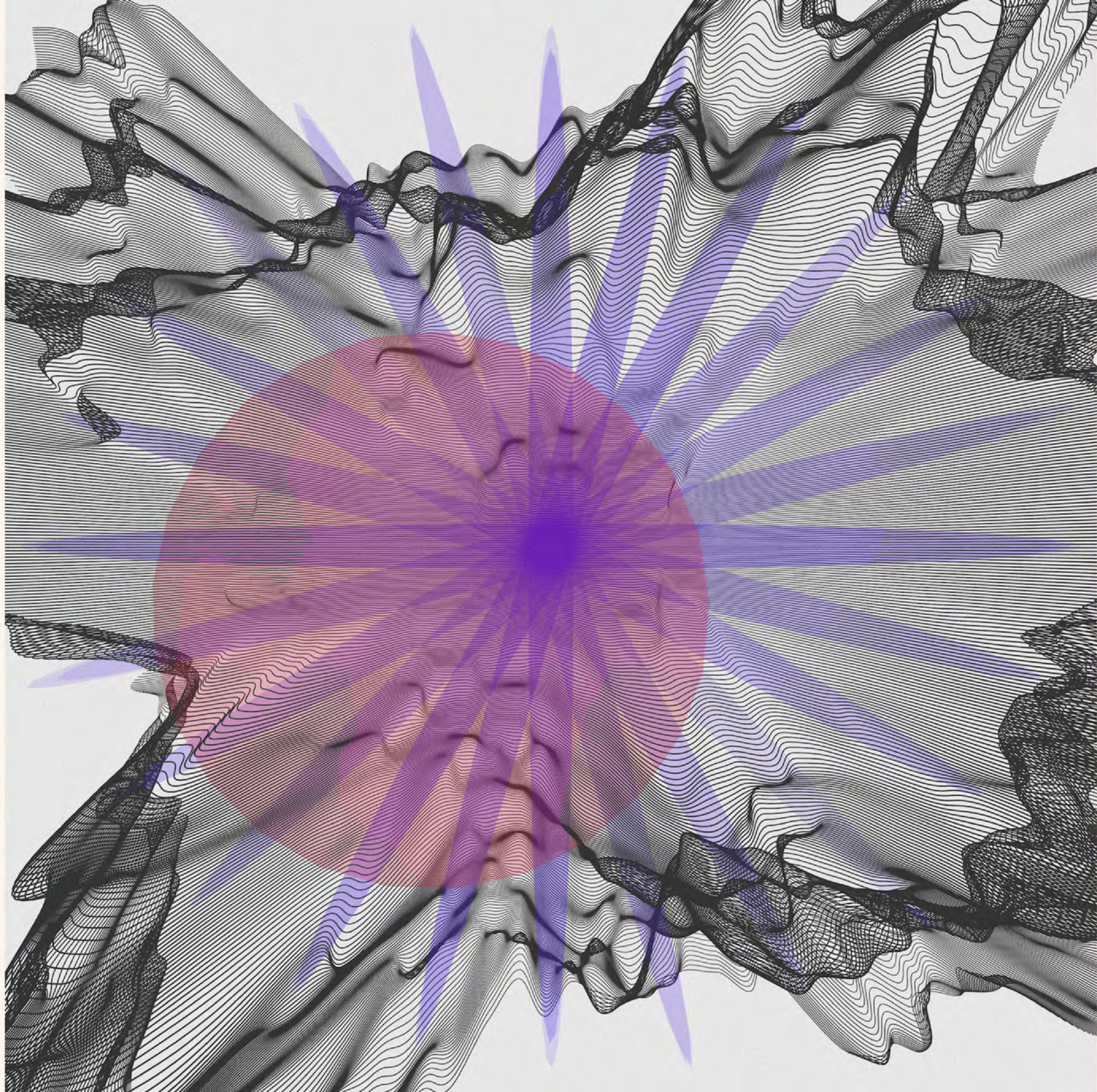


4. RESULTADOS

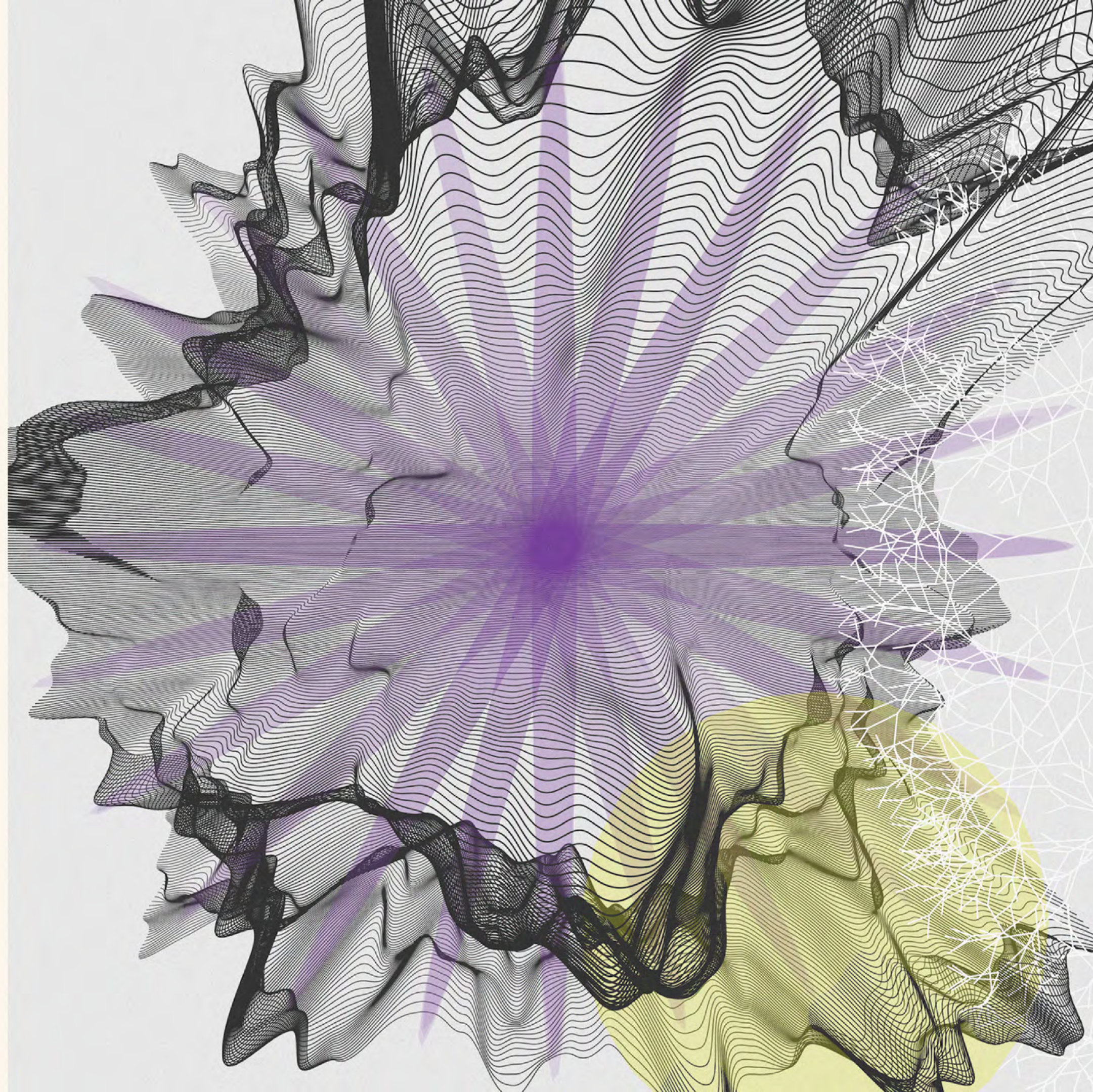
Após concluir a etapa de desenvolvimento do código, dei início ao processo de geração das imagens. Essa fase desempenha um papel de extrema relevância, uma vez que as 10 primeiras imagens produzidas pelo código serão utilizadas e não sofrerão alterações e nem serão descartadas. O objetivo dessa abordagem é conceder ao programa a responsabilidade ‘final’ e, assim, solidificar o trabalho conjunto entre o artista e o código. Cada série mantém a cor de fundo e das ondas consistentes, a fim de garantir um contraste adequado na composição das imagens. Como mencionado anteriormente, a formação e posicionamento dos demais elementos são realizados com o auxílio da ferramenta *random*, que possibilita a geração de inúmeras variações.



LÚCIDO

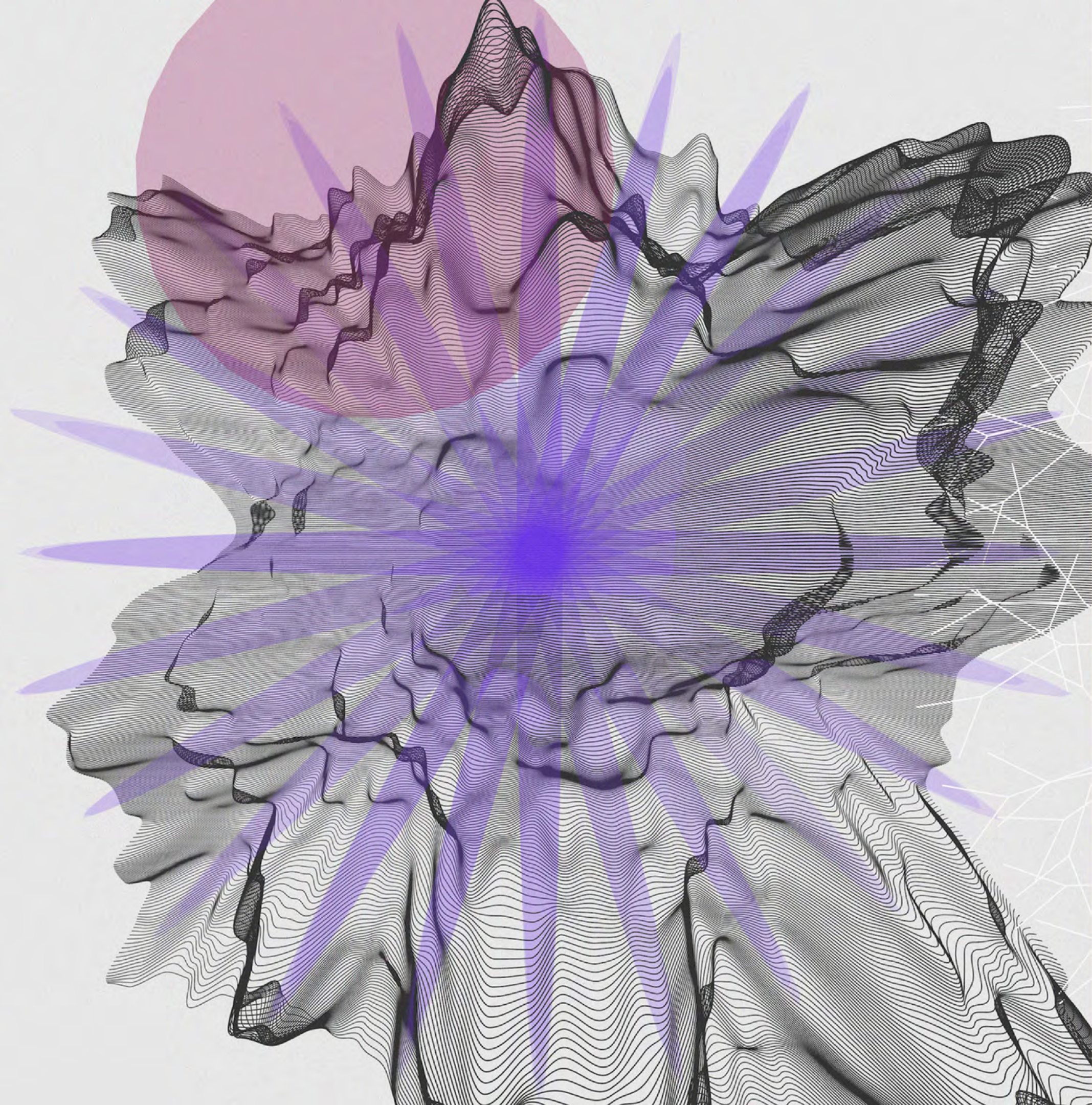


#01

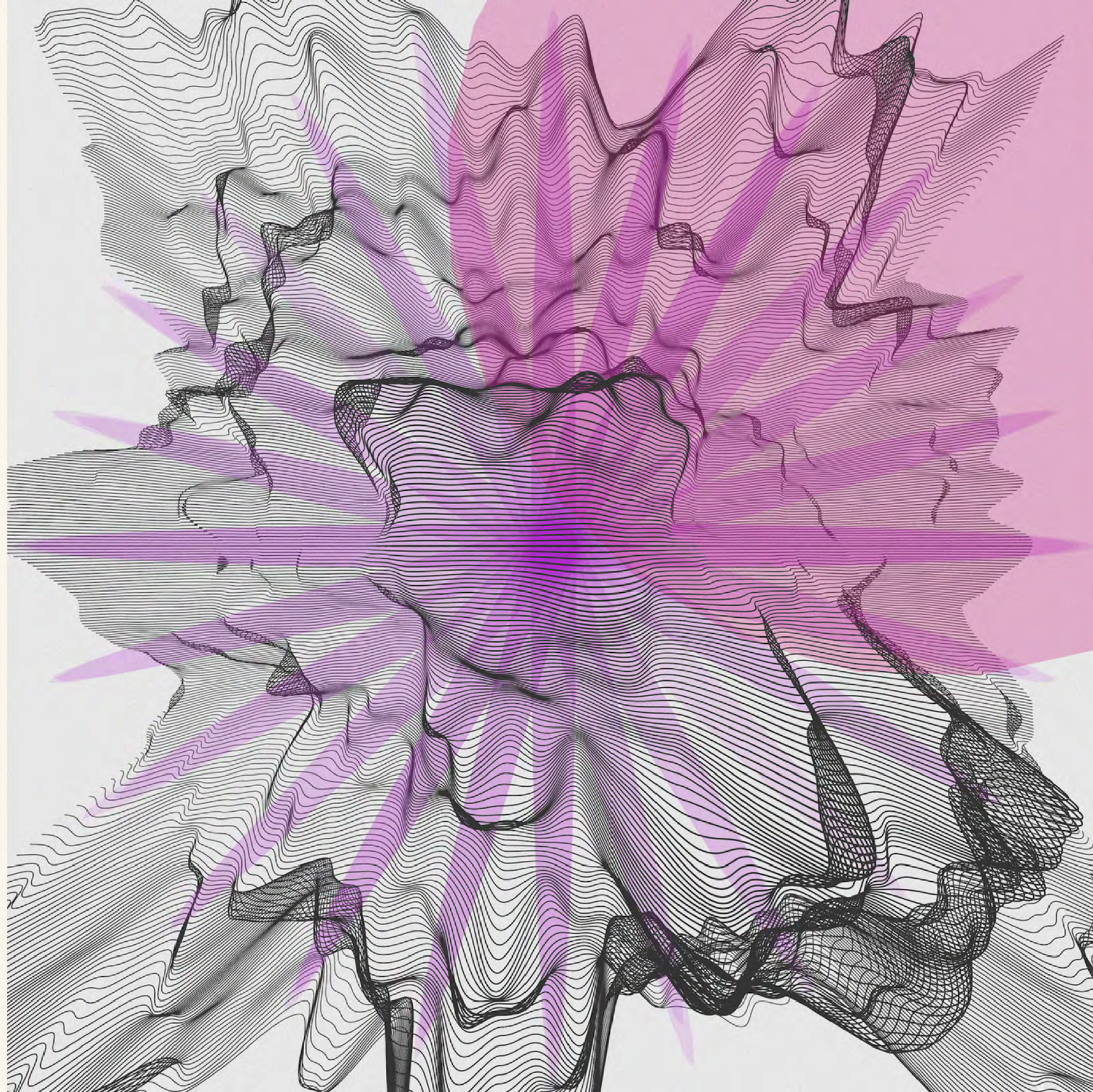


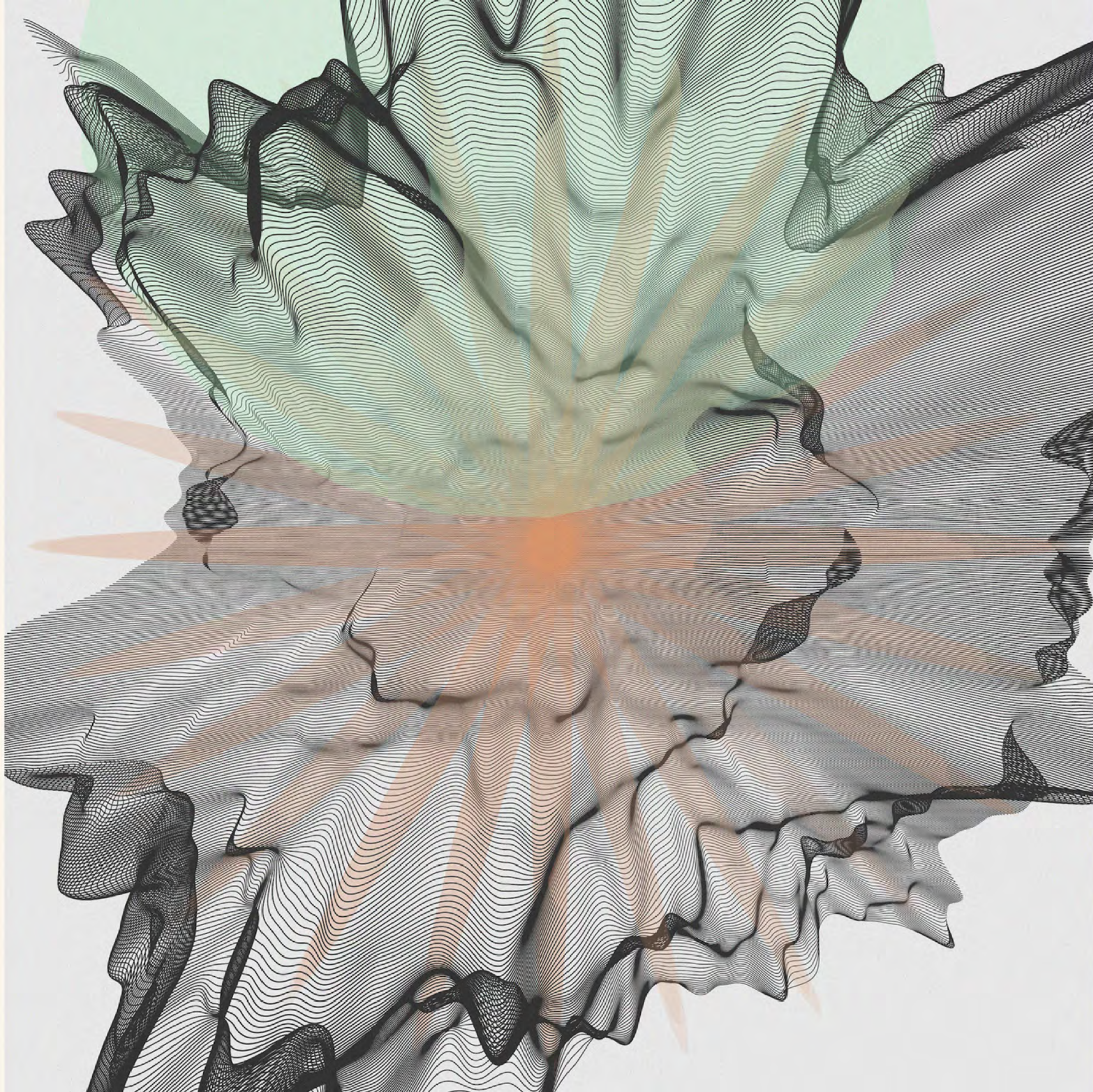
#02

#03

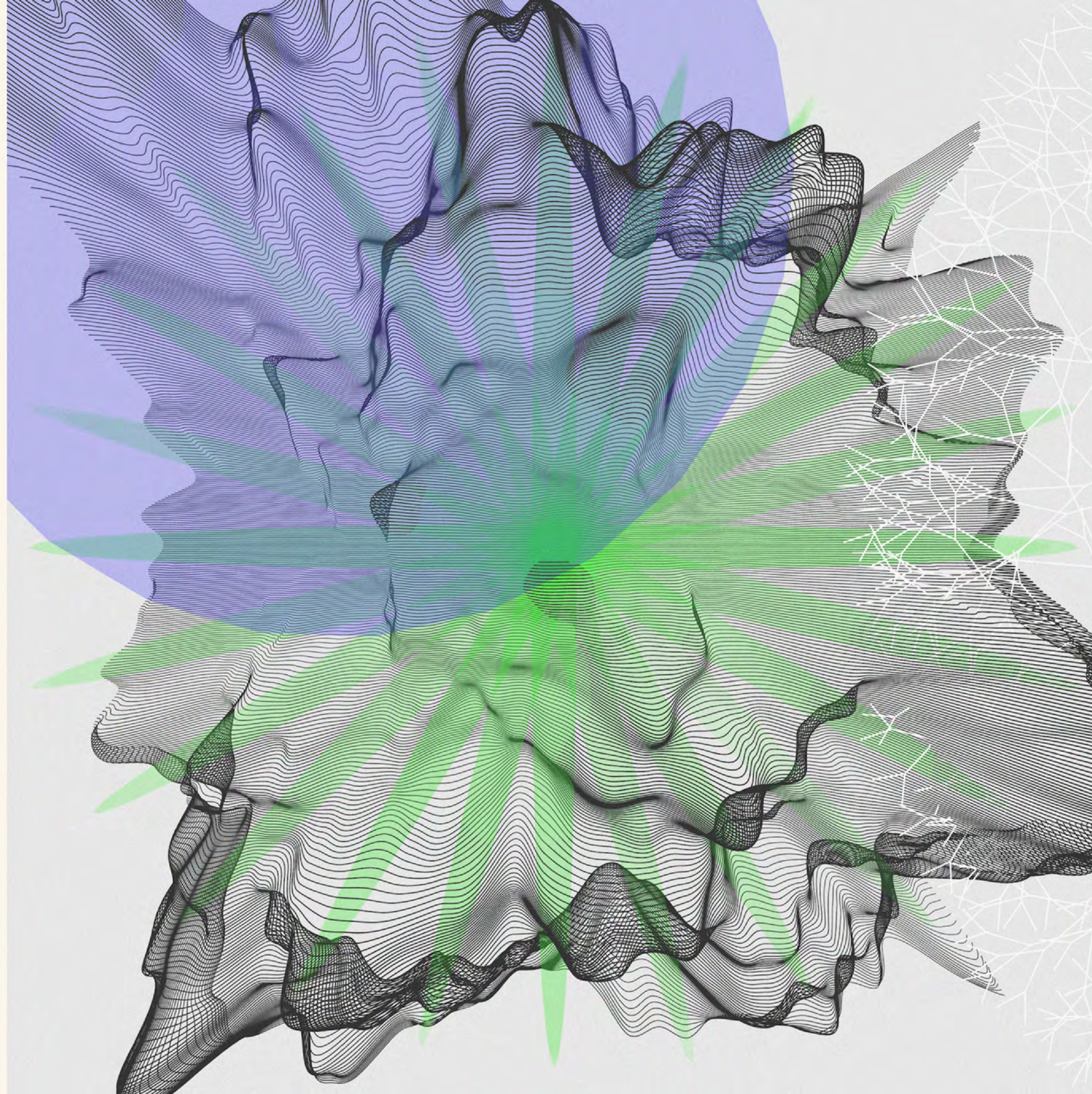


#04



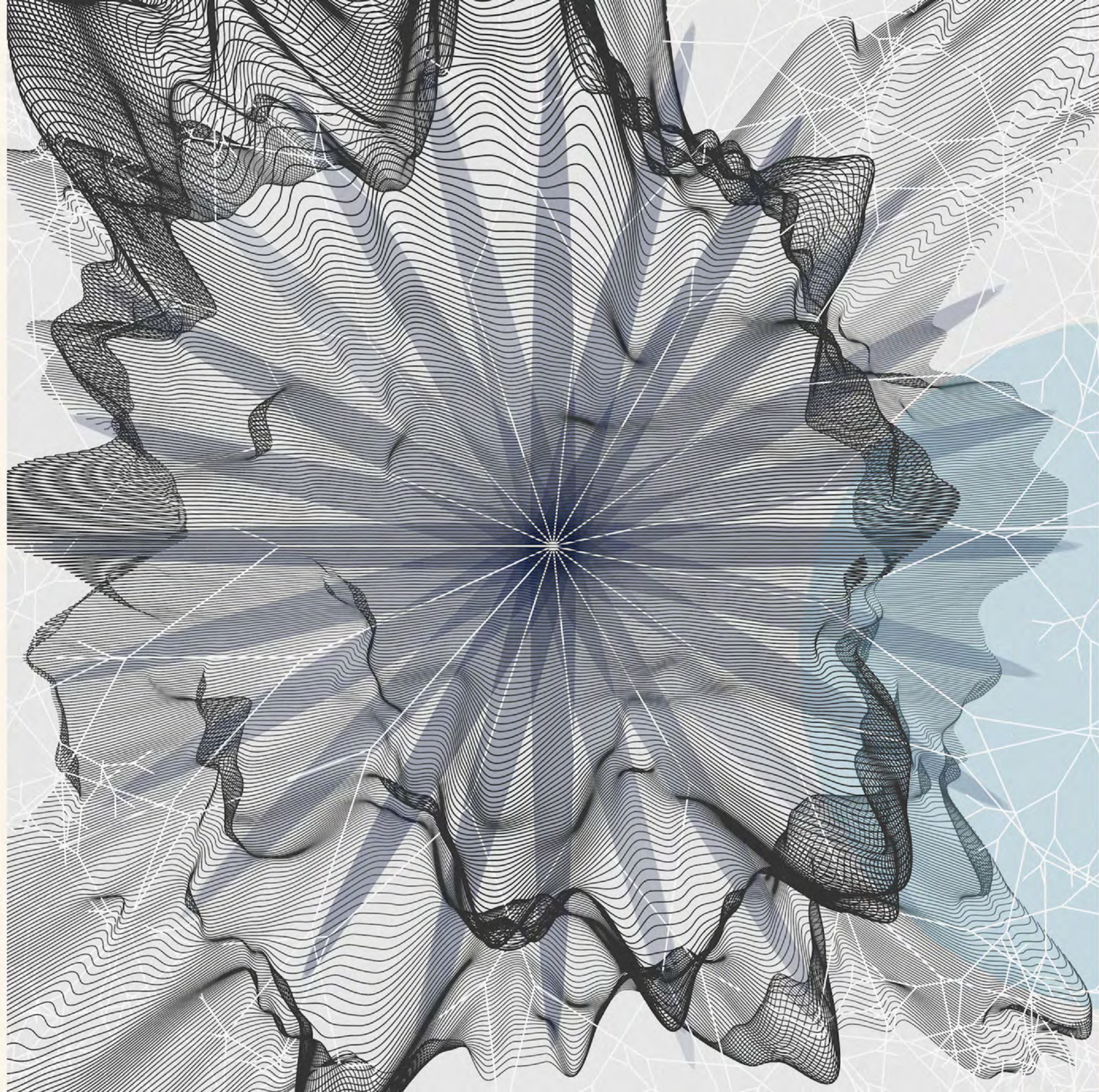


#05

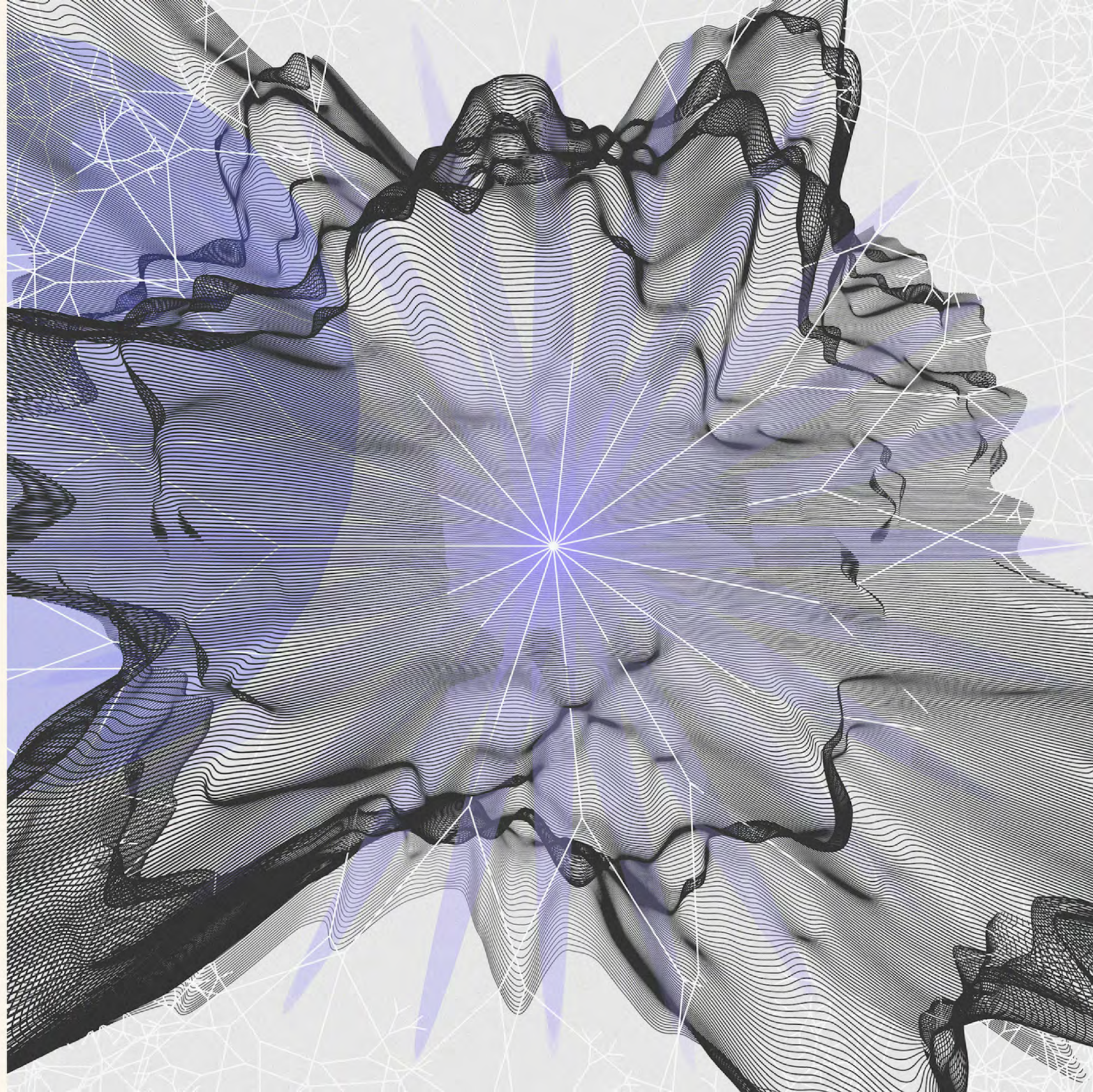


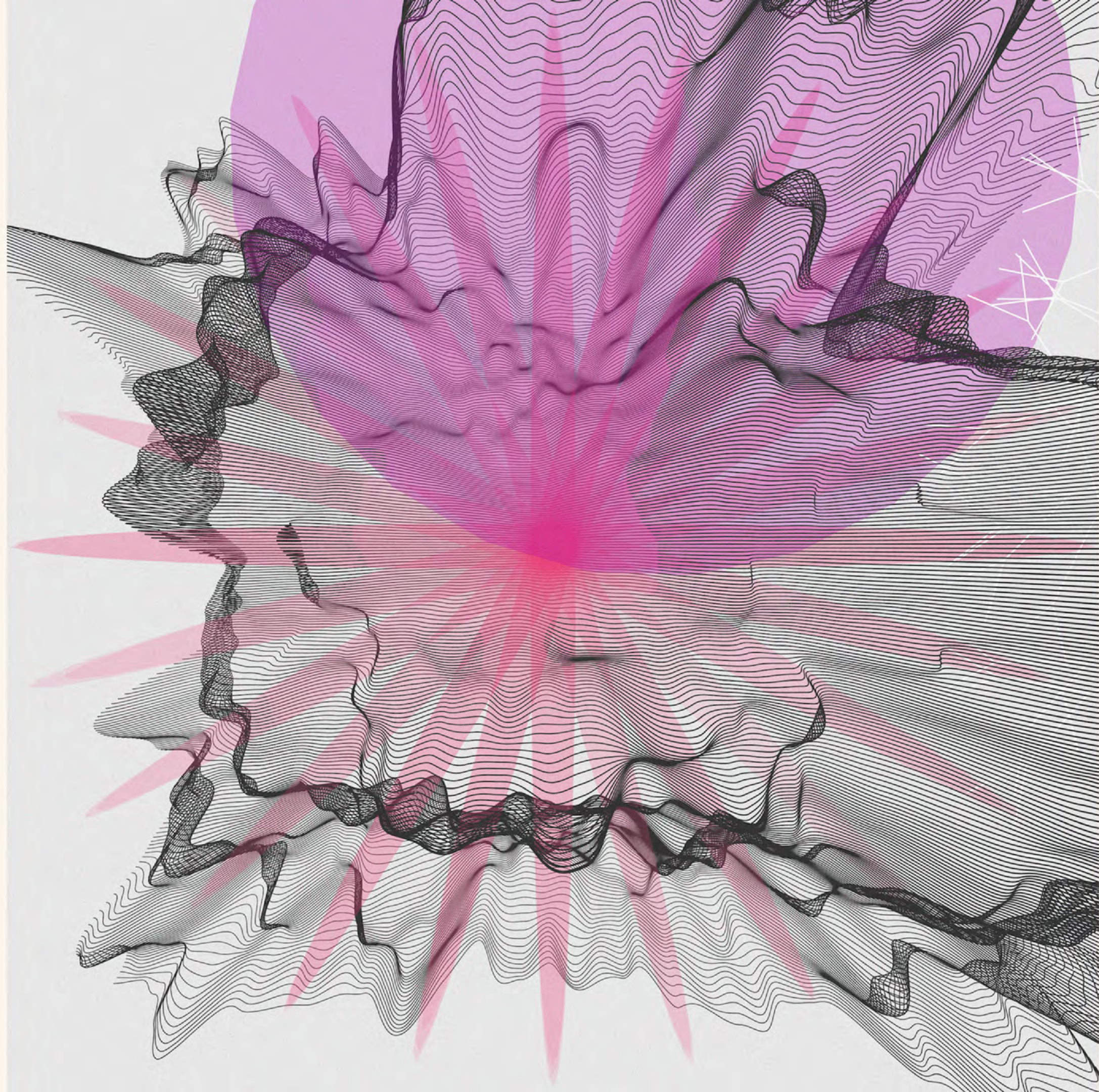
#06

#07

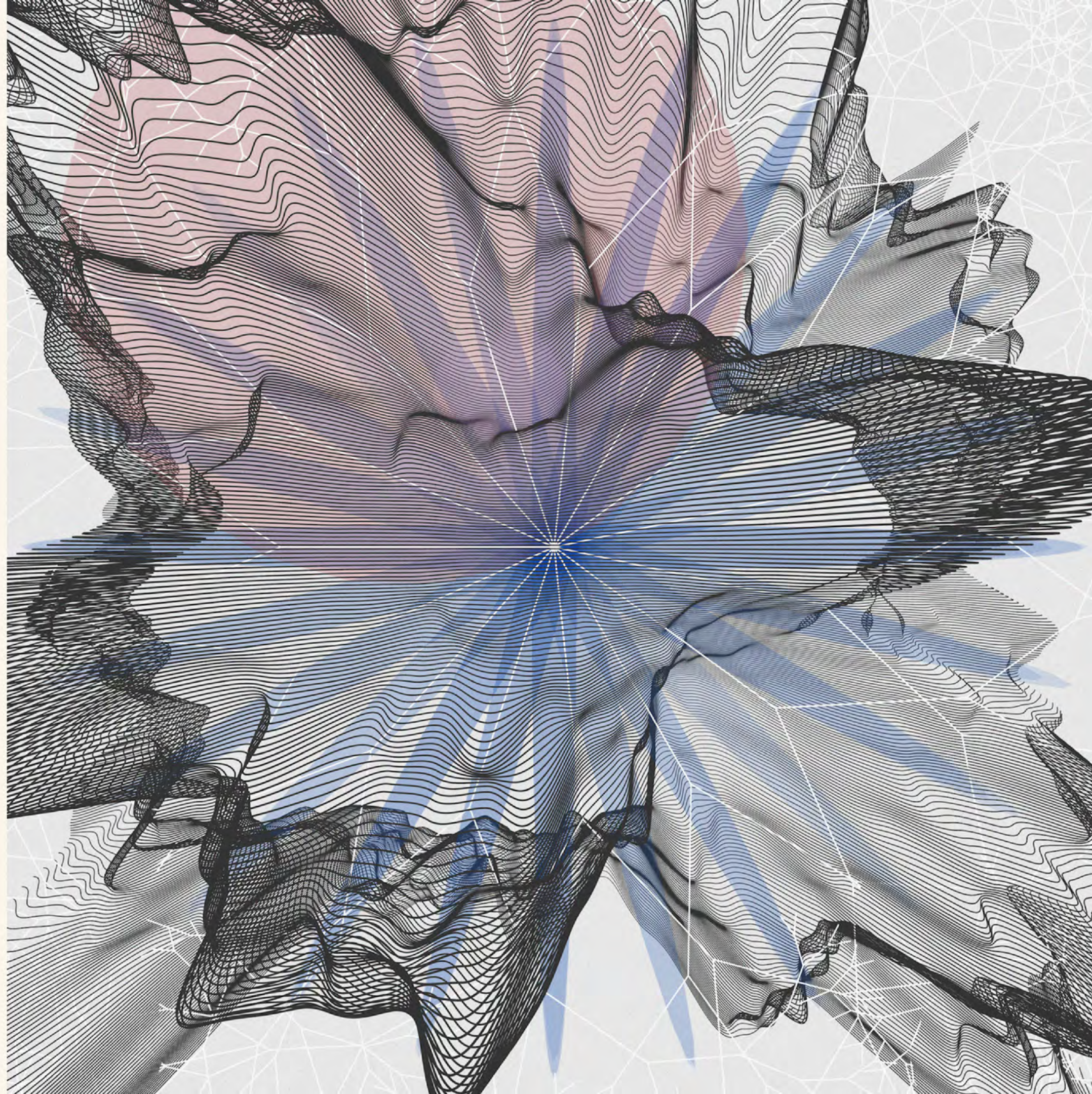


#08

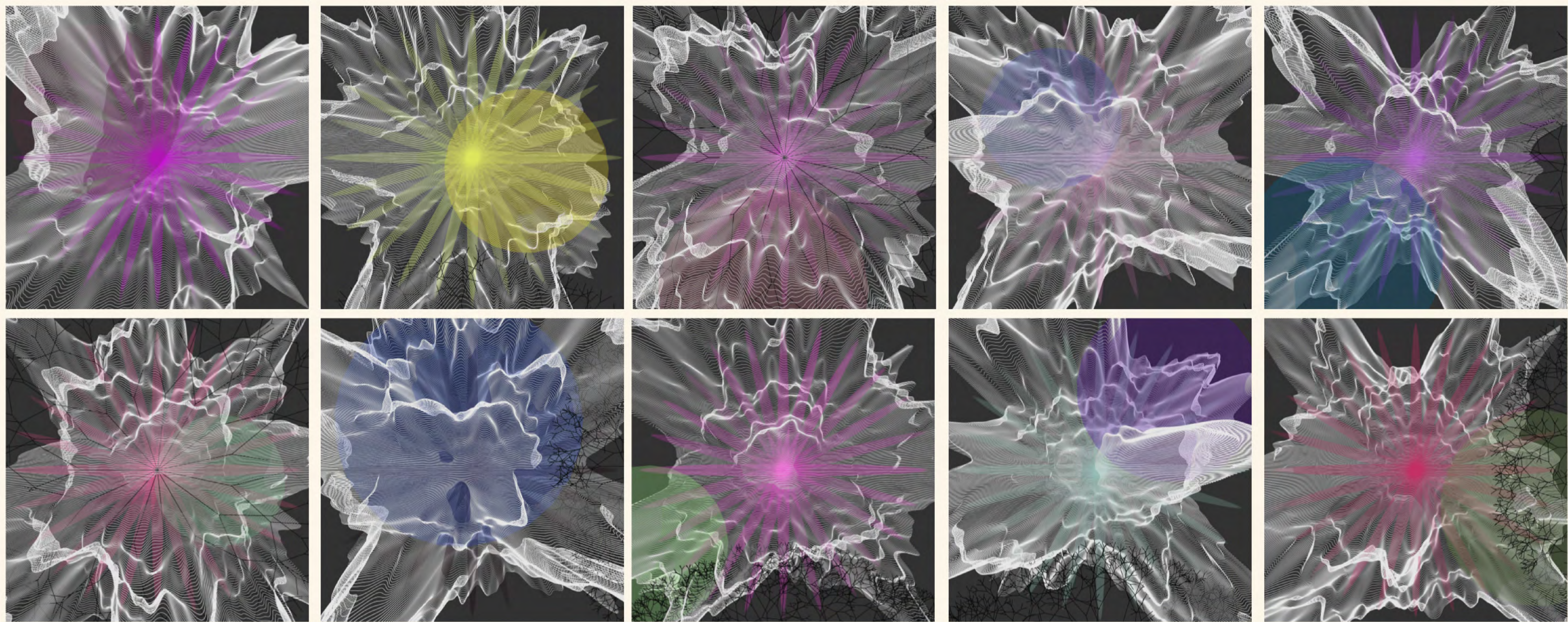




#09

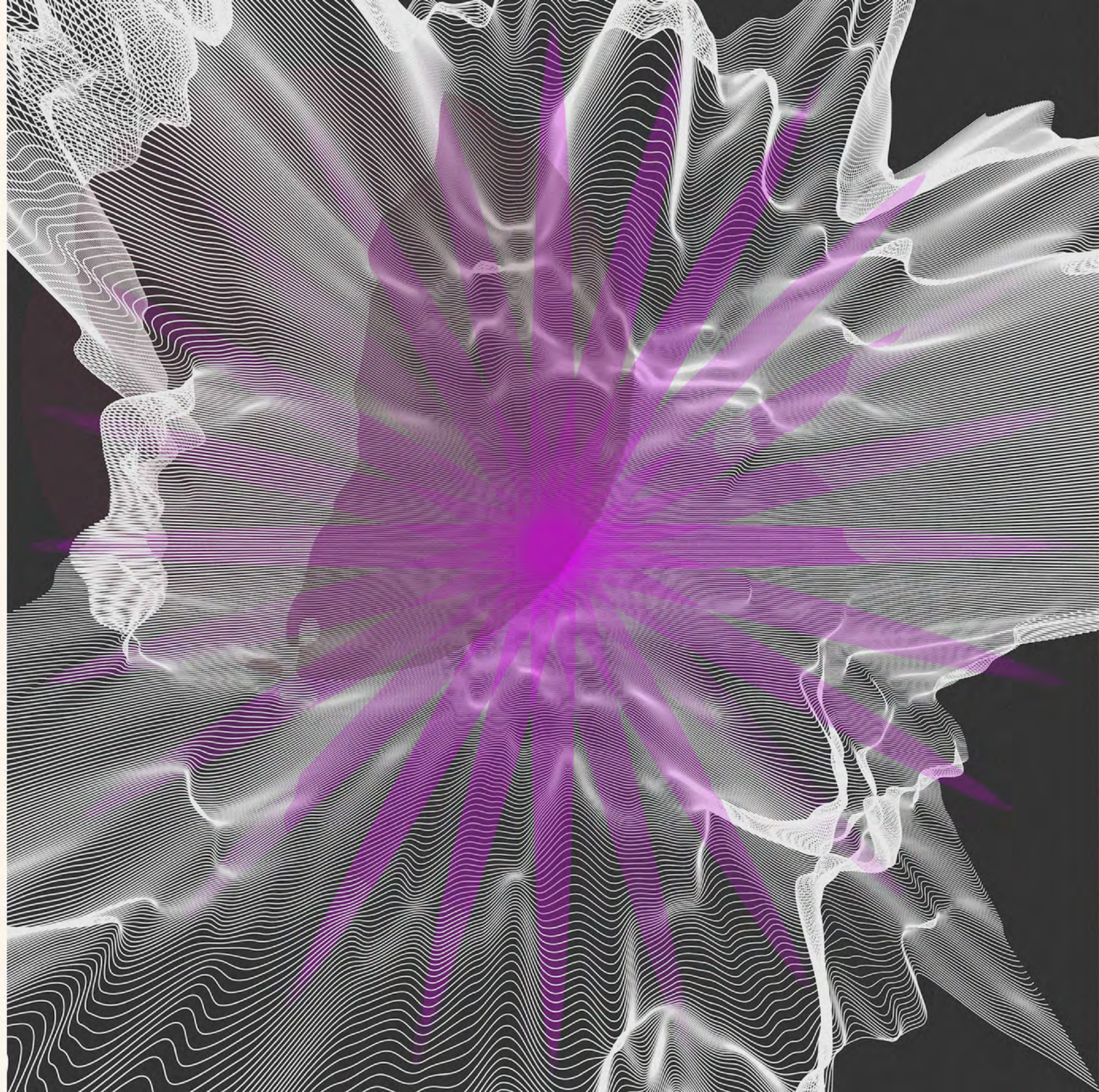


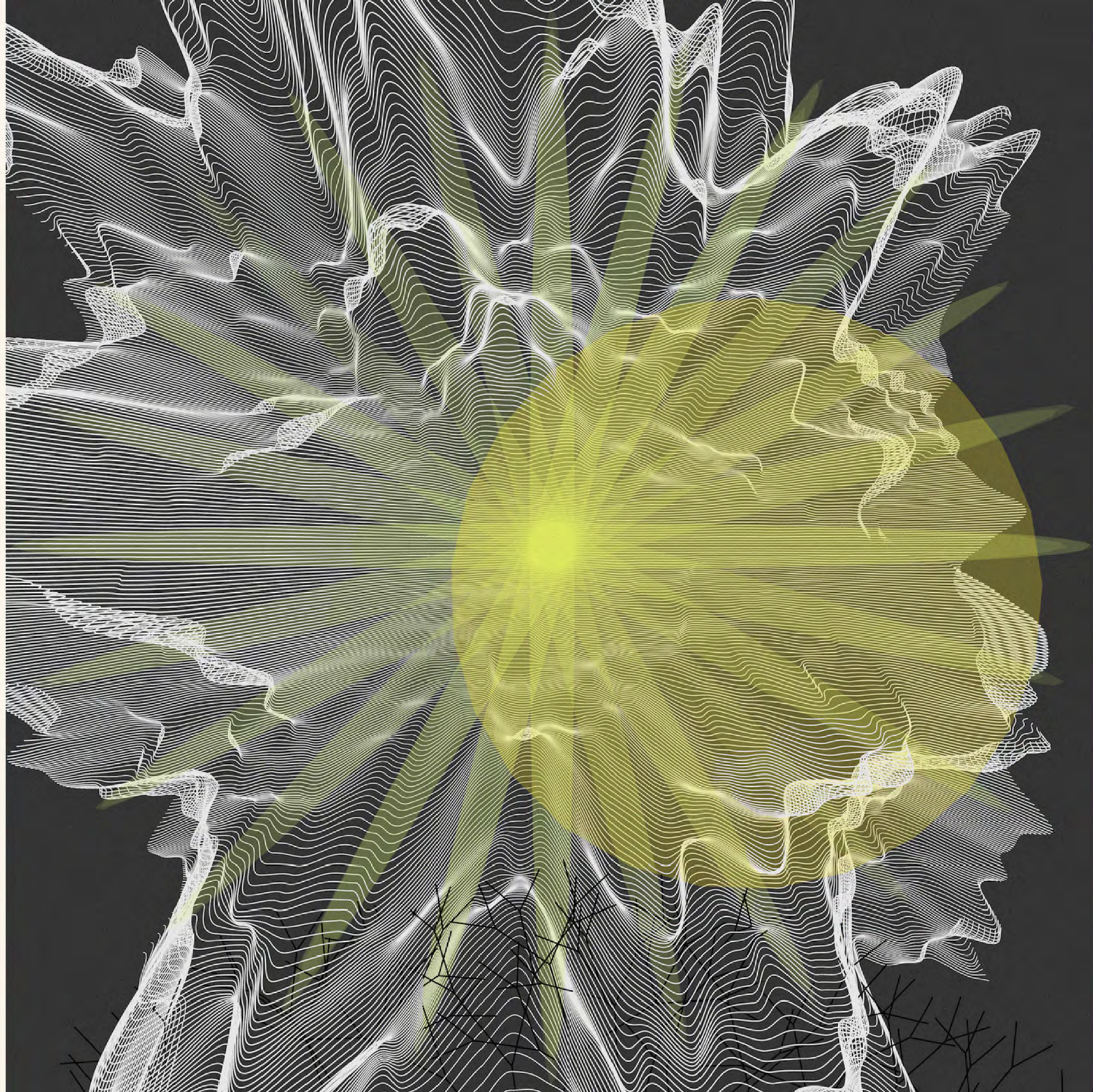
#10



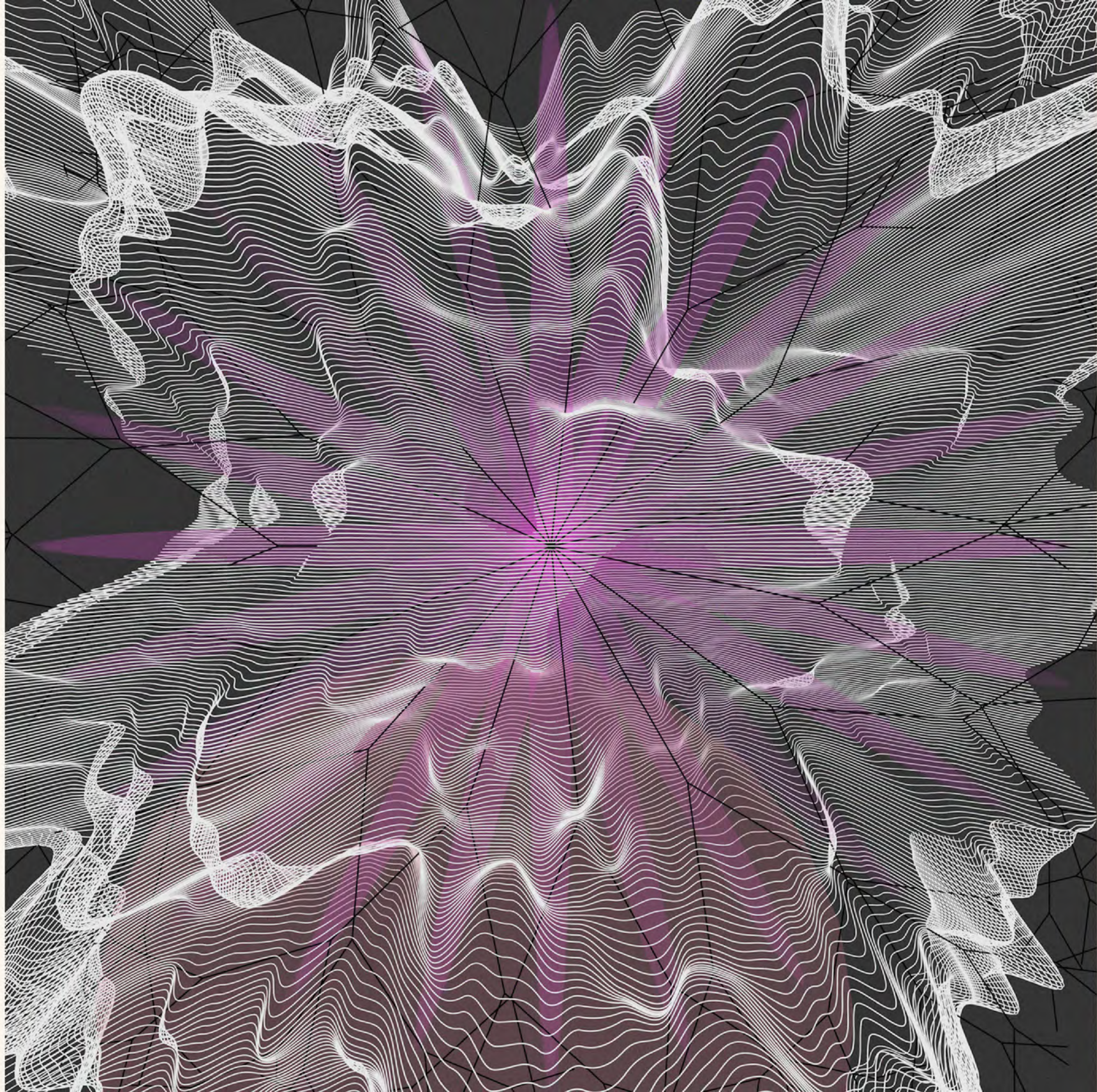
CERRADO

#01



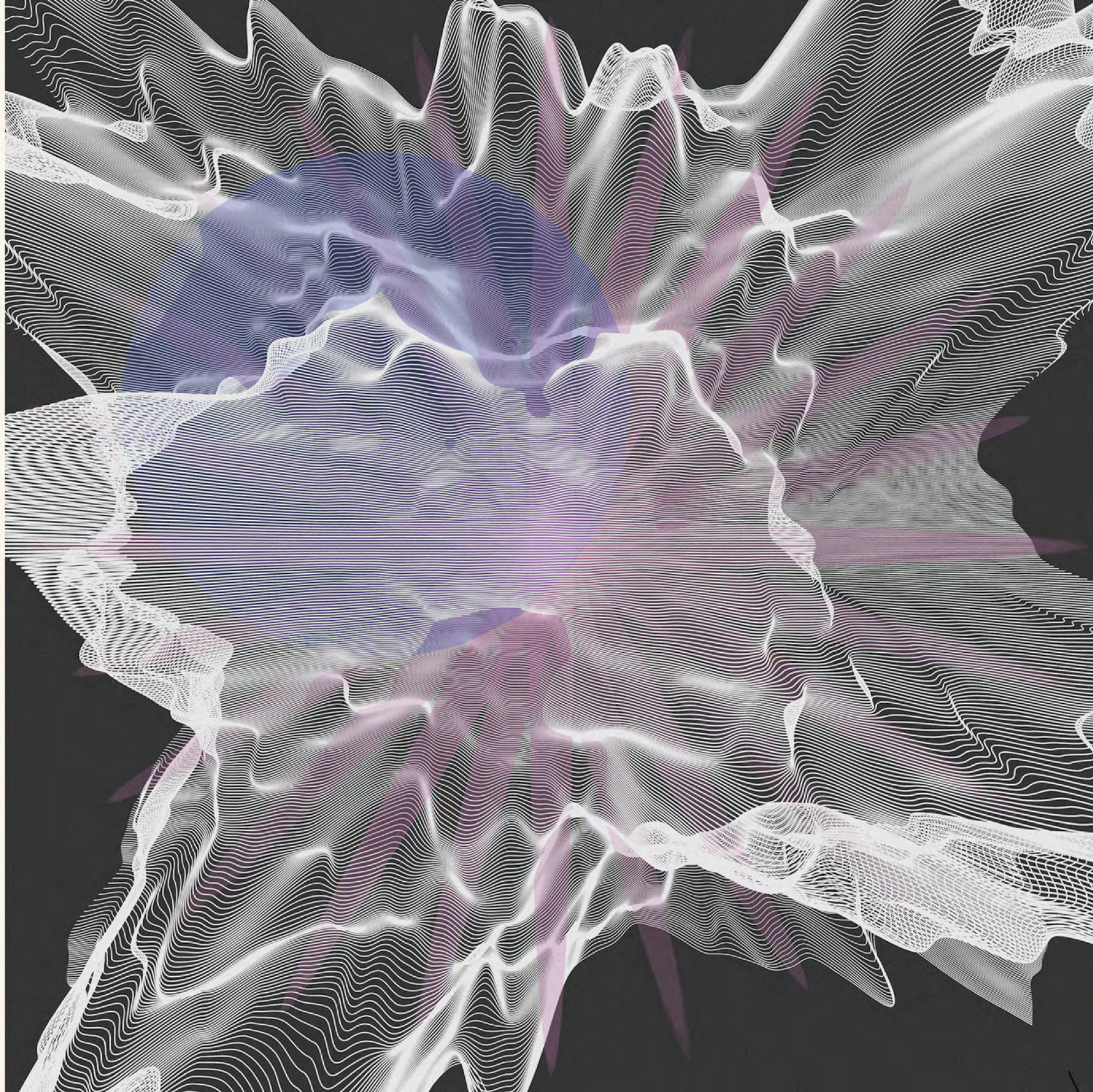


#02

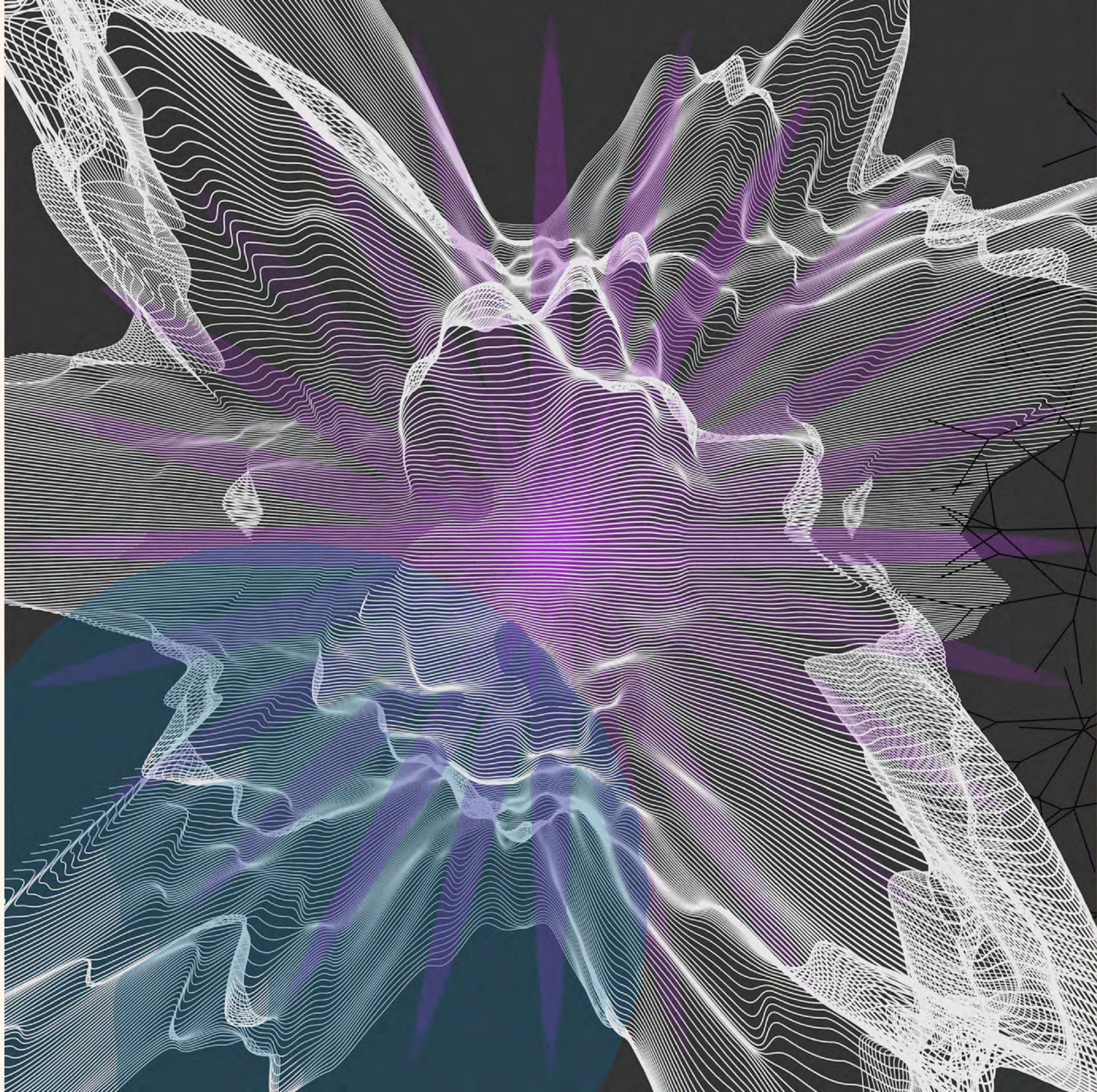


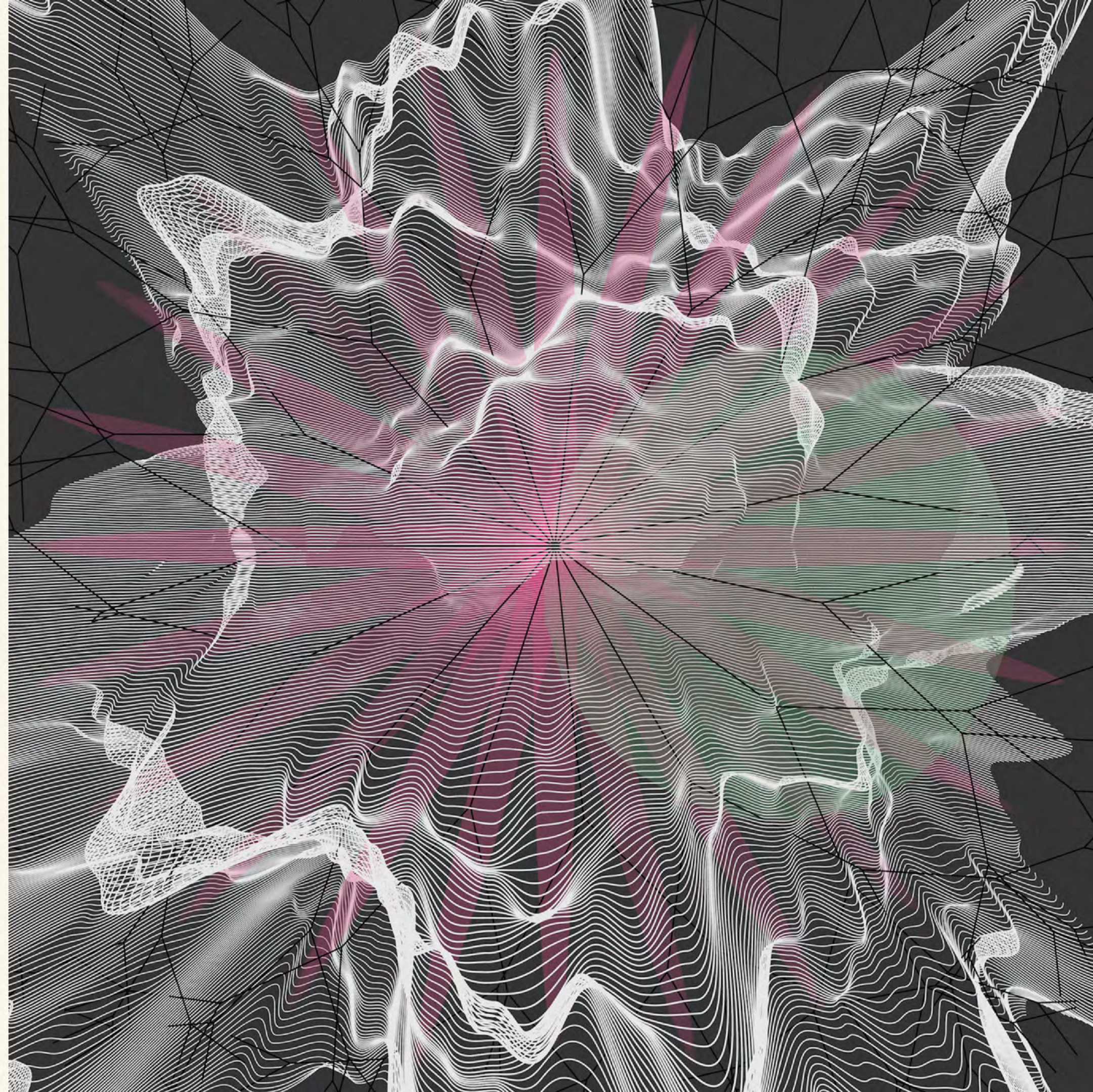
#03

#04

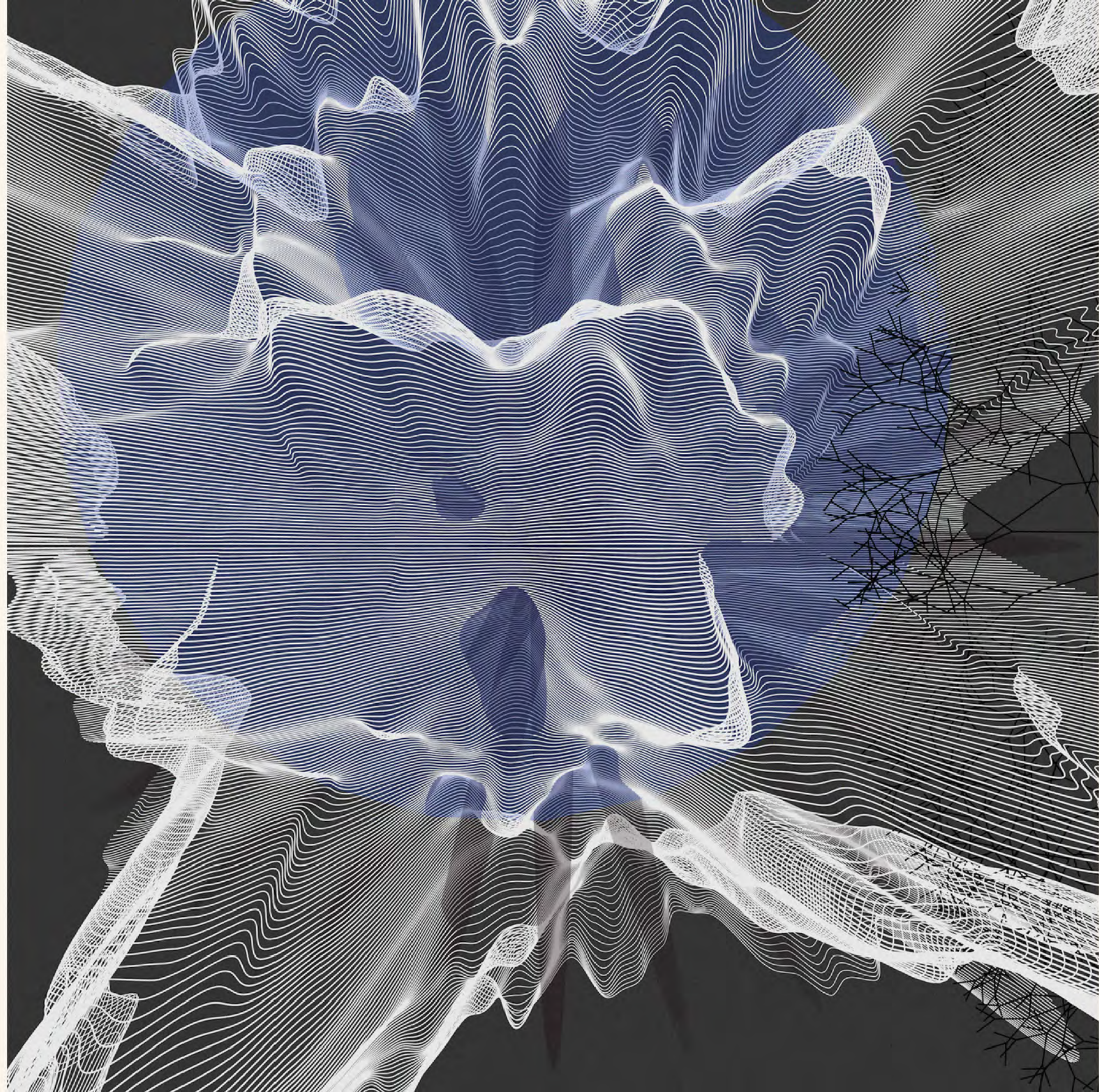


#05



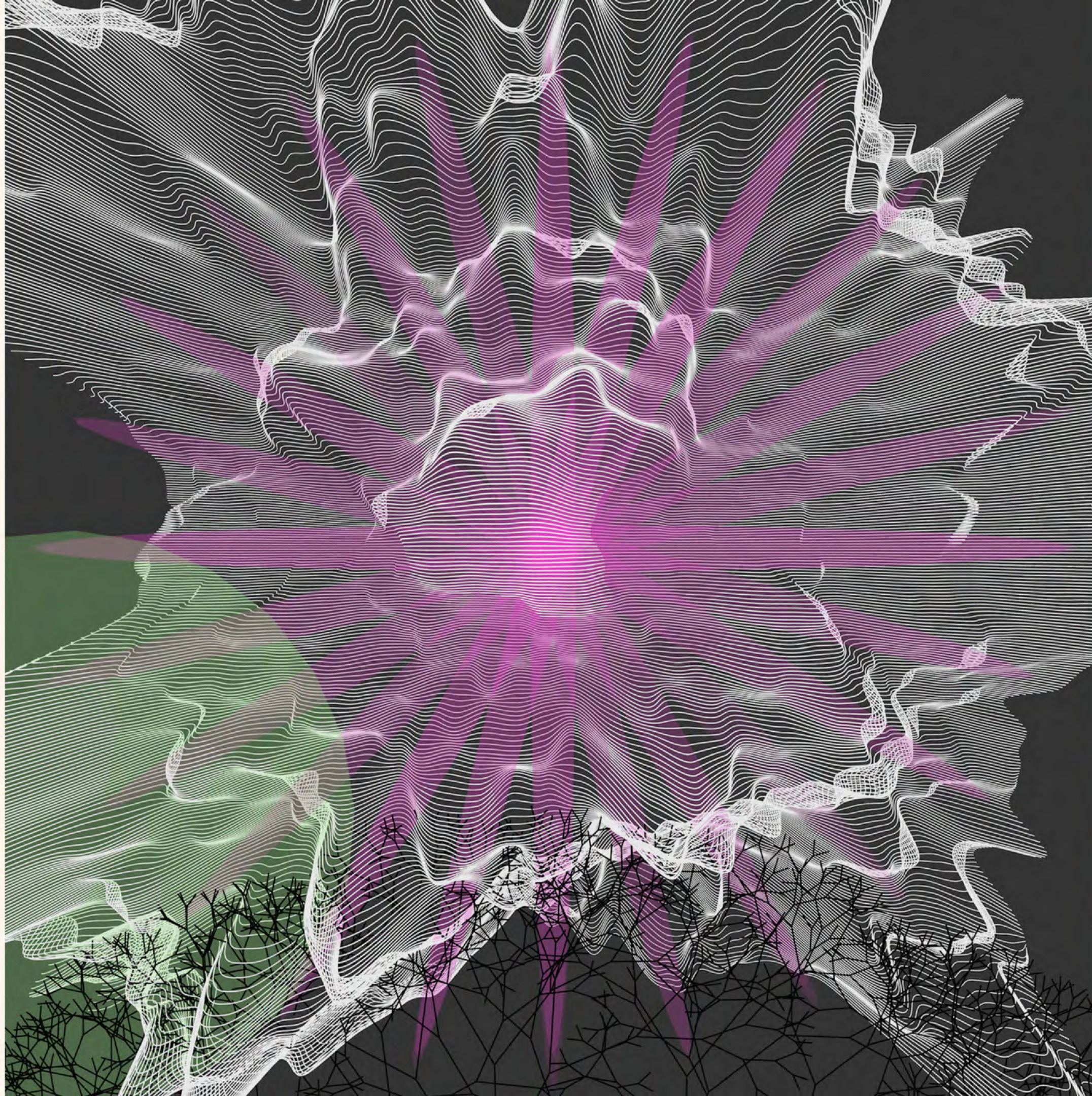


#06

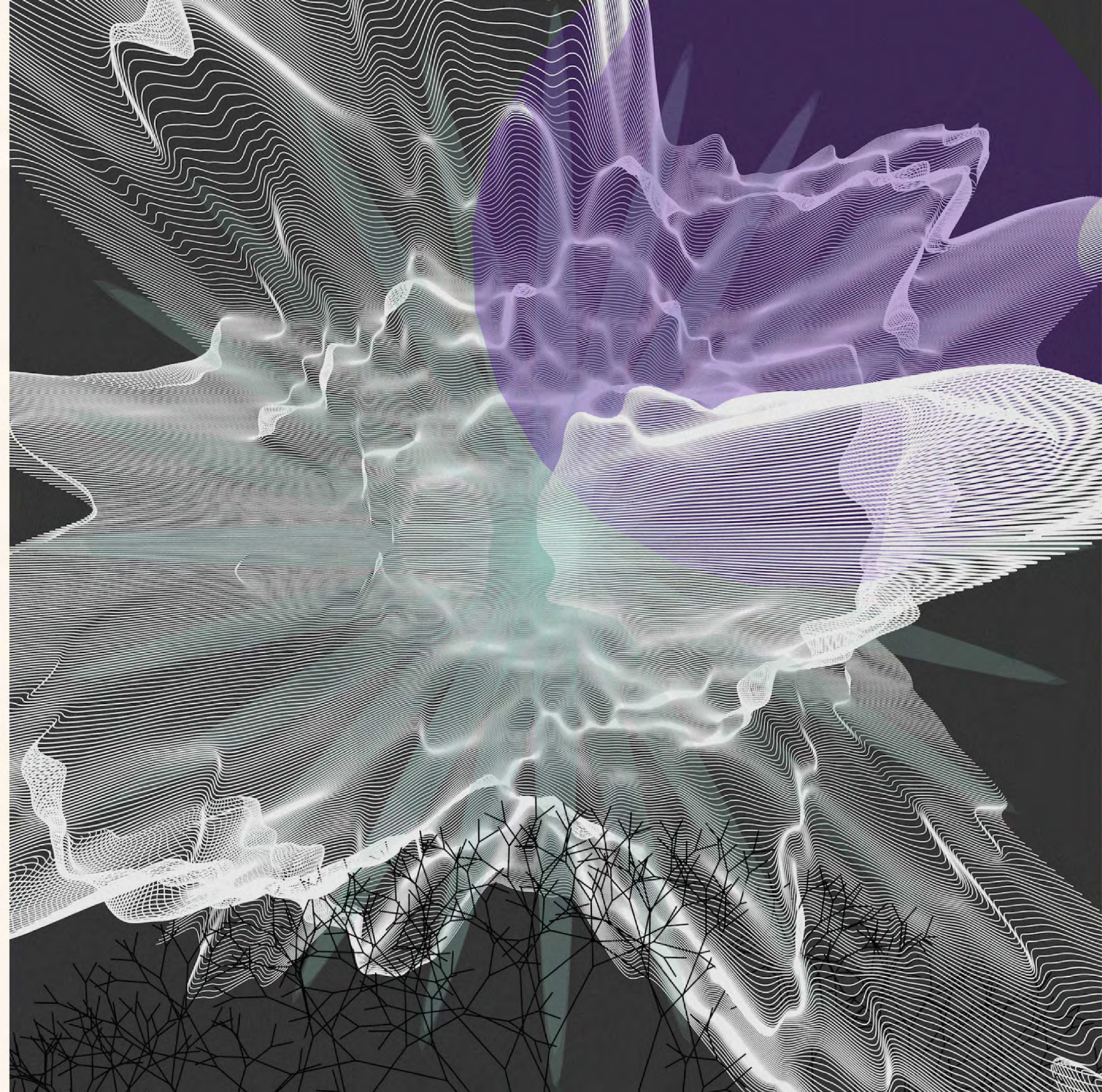


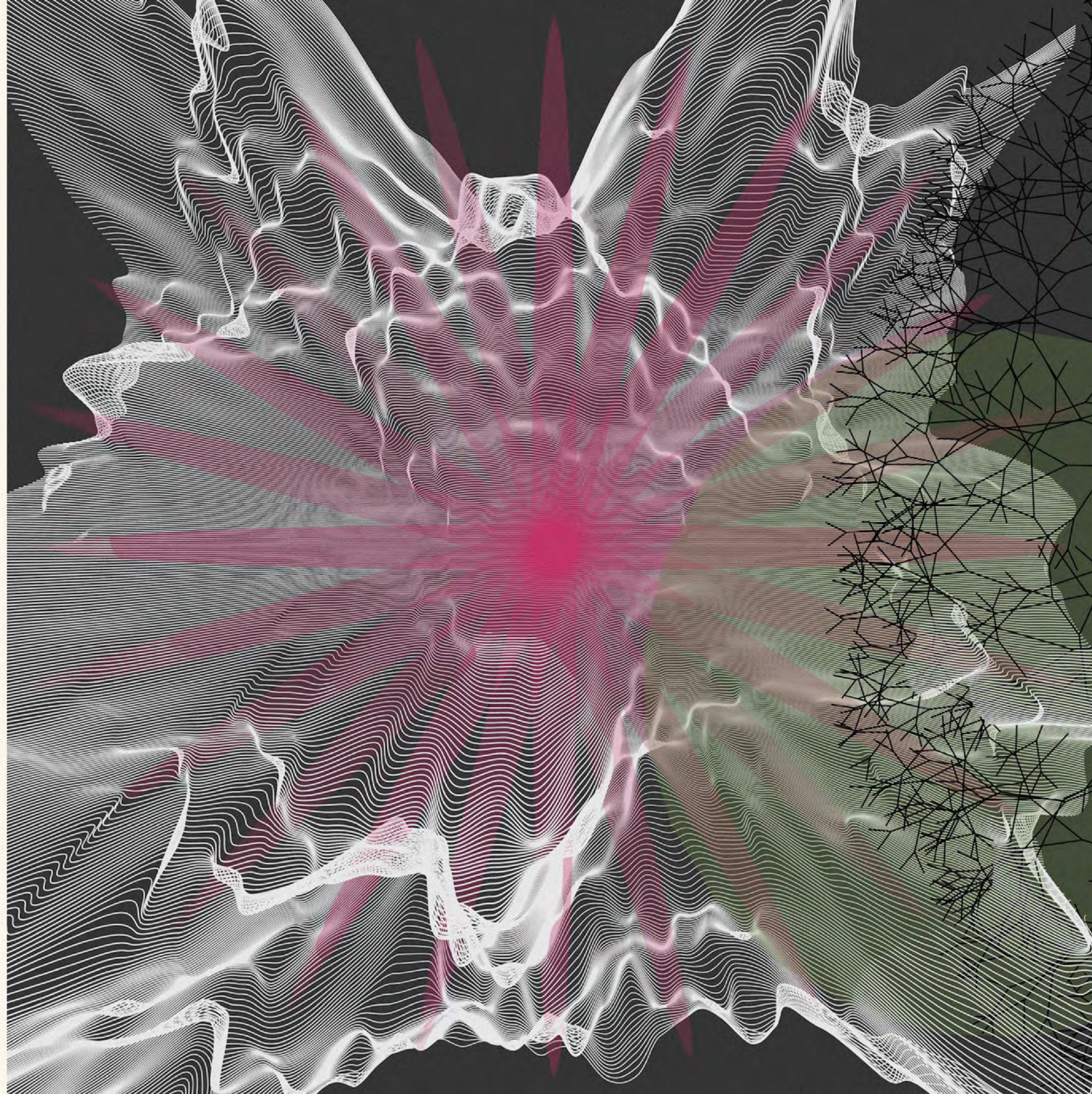
#07

#08



#09





#10

5. CONSIDERAÇÕES FINAIS



A arte generativa computacional pode ser definida como uma forma de expressão artística que utiliza algoritmos e processos computacionais para criar obras de arte com certa autonomia, em constante evolução e que emergem a partir da interação entre o artista e o código. Essa forma de criação desafia as noções tradicionais de autoria e controle artístico, permitindo a exploração de novas possibilidades criativas e questionando as fronteiras entre humano e algoritmo.

A ideia inicial deste trabalho envolveu a pesquisa sobre a definição de arte generativa, abordada no primeiro capítulo, com foco na arte computacional. Desde o início, a definição foi abrangente, permitindo diversas possibilidades. O trabalho experimental foi realizado com base em uma compreensão básica da arte computacional generativa, sem conhecimento prévio de programação, o que demonstra como a tecnologia moderna, além de fornecer ferramentas, facilita sua compreensão e aprendizado. Ao longo do processo, foram enfrentadas dificuldades pessoais e relacionadas à falta de experiência na área em questão, como mencionado anteriormente, bem como a necessidade de conciliar a graduação com o trabalho remunerado, reduzindo o tempo hábil disponível para o desenvolvimento desta pesquisa.

Em relação à proposta, foi estabelecido o objetivo de compreender: 1. como a redução ou aumento do controle do artista sobre o programa afeta o resultado estético e conceitual da obra; 2. de qual forma e por que a artista cede controle ao código e; 3. qual é o nível necessário de autonomia para o trabalho ser considerado generativo.

É perceptível que a artista estabeleceu e limitou as principais alternativas, o que permitiu direcionar o resultado estético do trabalho. Vale ressaltar que a escolha de certos parâmetros foi indispensável para esse tipo de criação, caso contrário, os resultados não estariam em conformidade com as referências iniciais. Por esse motivo, o trabalho foi estruturado para permitir a interferência da artista até a finalização do código, momento em que as possibilidades oferecidas pelo código foram consideradas 'definitivas' para a obra. Dessa forma, podemos concluir que o resultado estético é, em princípio, e nesta proposta, controlado pelo programador, estabelecendo um conjunto de possibilidades determinadas pelo código durante a formação da imagem. A mudança no resultado é inteiramente dependente desse equilíbrio entre a intervenção do artista-programador e as escolhas do código, sendo que, dentro das possibilidades, existem inúmeras formas de alcançar o resultado final. Foi esta maneira que a artista encontrou de ceder parte do controle ao código.

O resultado conceitual, neste contexto, leva em consideração o propósito inicial do trabalho. Ele pode ser modificado pelas ações tomadas após a geração das imagens pelo código, como no caso onde diversas imagens são criadas e apenas uma é selecionada como final. No entanto, o objetivo deste trabalho foi alcançado justamente ao abrir mão dessa seleção. Desta forma, intervir no resultado poderia não causar diferenças drásticas visuais, mas agindo assim, conceitualmente, o objetivo não seria alcançado.

Portanto, conclui-se que a não intervenção após a criação das imagens é tão importante quanto a própria escrita do código, sendo fundamental para a compreensão completa do trabalho. Caso contrário, renunciando à proposta conceitual, alterando os resultados finais das séries geradas, se estaria condicionando ainda mais o resultado a um gosto estético particular, em detrimento de se usufruir de modelos gerados pelo código.

O questionamento sobre o nível de autonomia necessário para que a obra seja considerada generativa permanece inconclusivo. Na pesquisa teórica, ficou claro que a quantidade de autonomia do sistema pode variar drasticamente, e não há como estabelecer, de forma definitiva, um valor mínimo de fatores fora do controle do artista. No entanto, é necessário considerar que esse valor mínimo deve ser relevante para o resultado da obra, e, no final das contas, a decisão de denominar um trabalho como generativo ou não cabe ao artista.

Neste momento, o que mais importa, é perceber que a obra é o processo em si, independente dos resultados gerados. E que a artista carrega valores estéticos, que orientam de alguma forma a obra, seja na elaboração dos parâmetros iniciais, nos ajustes dos algoritmos, nos testes e protótipos, até na identificação final dos modelos. Entretanto, esta ação conjunta criativa da arte generativa questiona modelos tradicionais de obras, autoria valores estéticos, mesmo que de forma superficial.

No futuro, pode-se pensar em aumentar a complexidade do código, adicionando outras possibilidades, para gerar resultados ainda mais diversos sem a intervenção direta do programador. Isso exigiria um estudo aprofundado da linguagem de programação para obter maior autonomia na alteração dos códigos e suas características, diminuindo a dependência das soluções desenvolvidas por terceiros.



REFERÊNCIAS

BODEN, M E EDMONDS A. **WHAT IS GENERATIVE ART?** [s.l: s.n.]. <https://www.researchgate.net/publication/233128802_What_is_generative_art/link/00463521693ec679c8000000/download> Acesso em julho de 2023.

GALANTER, P. **What is Generative Art? Complexity Theory as a Context for Art Theory.** 2003. <https://www.philipgalanter.com/downloads/ga2003_paper.pdf> Acesso em julho de 2023.

GARCIA, Chris. **HAROLD COHEN AND AARON: A 40-YEAR COLLABORATION.** Computer History Museum, 2016. <<https://computerhistory.org/blog/harold-cohen-and-aaron-a-40-year-collaboration/>> Acesso em julho de 2023.

DUNLOP, S. **Subject Guides: Coding: a Practical Guide: Processing and p5.js.** Disponível em: <<https://subjectguides.york.ac.uk/coding/p5js#:~:text=Processing.org%20is%20the%20official>>. Acesso em julho. 2023.

DRUID. AB 101: **Historical Figures in Generative Art — Georg Nees.** 2021. <<https://medium.com/the-link-art-blocks/ab-101-historical-figures-in-generative-art-georg-nees-6b006e62126a>> Acesso em julho de 2023.

MASON, Catherine. **Cybernetic Serendipity: History and Lasting Legacy.** Studio Internacional, 2018. <<https://www.studiointernational.com/cybernetic-serendipity-history-and-lasting-legacy>> Acesso em julho de 2023.

NORMAN, Jeremy. **"Cybernetic Serendipity": The First Widely-Attended International Exhibition of Computer Art.** 2012. <<https://www.historyofinformation.com/detail.php?entryid=1089>> Acesso em julho de 2023.

SIMÕES, Luiz Felipe. **Artista brasileira arrecada R\$ 28,4 milhões com coleção de NFTs.** Estadão, 2021. <<https://einvestidor.estadao.com.br/comportamento/artista-brasileira-nfts-arrecada-28-milhoes/>> Acesso em julho de 2023.



APÊNDICES

Código para "Lúcido" no p5.js:
<https://editor.p5js.org/beacoxta/sketches/uKnKMHpAm>

ou

SKETCH.JS

```
let noiseScale = 0.002; // nível de noise
let yStart = 0;
let yStep;
let randSeed;
let minNumOfLines = 300;
let maxNumOfLines = 500;
let myFilterShader;
let branchMin = 5;
let branchMax = 10;
let initialLines = 15;

function preload() { // filtro grain
  myFilterShader = loadShader("filter.vert",
"grain.frag");
}

function setup() {
  createCanvas(2160, 2160, WEBGL); //2x tela
  randSeed = random(100);
  angleMode(DEGREES);
  noiseSeed(randSeed);
  numOfLines = floor(random(minNumOfLines,
maxNumOfLines));
  yStep = height / numOfLines;
  generateLines();
  drawTree();

  myFilterShader.setUniform("grainAmp", 0.3);
  filterShader(myFilterShader);
}
```

```
function generateLines() { // ondas
  for (let i = 0; i < numOfLines; i++) {
    beginShape();
    let y = yStart + i * yStep;
    for (let x = 1; x < width; x++) {
      let noiseVal = noise(x * noiseScale, y *
noiseScale, randSeed);
      let zPos = map(noiseVal, 0, 1, -height, height);
      vertex(x - width / 2, y - height / 2, zPos);
    }
    endShape();
  }
}
```

```
function drawTree() { // árvore fractal
```

```
  let startCorner = floor(random(0, 4));

  let branchCount = floor(random(branchMin,
branchMax + 1));
```

```
  let startX, startY;
  if (startCorner === 0) {
    startX = 0;
    startY = height;
  } else if (startCorner === 1) {
    startX = width;
    startY = height;
  } else if (startCorner === 2) {
    startX = width;
    startY = 0;
  } else {
    startX = 0;
    startY = 0;
  }
}
```

```
for (let k = 0; k < initialLines; k++) {
  let angle = map(k, 0, initialLines, -180, 180);
  drawBranch(startX, startY, width / 4, angle,
branchCount);
}
}
```

```
function drawBranch(x, y, length, angle, branches) {
  if (branches === 0) {
    return;
  }
```

```
  let endX = x + length * cos(angle);
  let endY = y + length * sin(angle);
```

```
  line(x, y, endX, endY);
```

```
  drawBranch(endX, endY, length * 0.7, angle -
random(20, 40), branches - 1);
  drawBranch(endX, endY, length * 0.7, angle +
random(20, 40), branches - 1);
}
```

```
function draw() {
  background(230);
  //stroke (random(0, 255), random(0, 255),
random(0, 255), 100) // opção de linhas com cor
aleatória
  stroke(255)
  strokeWeight(3)
  //stroke(26, 36, 26, 100); // opção com linha verde
escura
  noFill();
  drawTree();
  stroke(50);
  strokeWeight(2)
  generateLines();
}
```

```
noStroke();
angleMode(RADIANS)
fill(random(0, 255), random(0, 255), random(0,
255), 30); // flores
for (let i = 0; i < 9; i++) {
  for (let f = 0; f < 600; f++) {
    let f = random(900);
  }
  ellipse(0, 0, 90, random((height-200), height));
  rotate(PI / 2);
  ellipse(0, 0, 90, random((height-200), height));
  rotate(PI / 3);
  ellipse(0, 0, 90, random((height-200), height));
  rotate(PI / 4);

}
```

```
noStroke();
fill(random(0, 255), random(0, 255), random(0,
255), 70); // círculo
let circled = random(width/3, width-200);
let circleX = random(-width/2, width/2);
let circleY = random(-height/2, height/2);
circle(circleX, circleY, circled);
```

```
myFilterShader.setUniform("grainAmp", 0.1);
filterShader(myFilterShader);
```

```
noLoop()
```

```
}
```

```
function keyTyped() { // clicar na imagem e digitar
"s" para salvar imagem
  if (key === "s") {
    save("lúcido.png");
  }
}
```


FILTER..VERT

```
lattribute vec3 aPosition;
attribute vec2 aTexCoord;

varying vec2 pos;

void main() {
    pos = aTexCoord;
    // flip the y axis
    pos.y = 1.0 - pos.y;

    vec4 positionVec4 = vec4(aPosition, 1.0);
    positionVec4.xy = positionVec4.xy * 2.0 - 1.0;

    gl_Position = positionVec4;
}
```

GRAIN.FRAG

```
#ifdef GL_ES
precision mediump float;
#endif

// Pixel position
varying vec2 pos;

// Uniforms set by filterShader
uniform sampler2D filter_background; // contains
the image being filtered
uniform vec2 filter_res; // contains the image
resolution in pixels

uniform float grainAmp; // how much variance in
grain texture

//uniform float millis; // time sketch has been
running

const float tau = 6.2831855; // 2 * PI

// The GPU has no random() function, so instead
// do some maths on the position of the pixel to
approximate a
// random number between 0 and 1
float rand(vec2 co, float salt){
    return fract(sin(dot(co, vec2(12.9898, 78.233) +
salt)) * 43758.5453);
}

void main() {
    // Read colour from image
    vec4 col = texture2D(filter_background, pos);
```

```
f // Use the rand function to create a random
number
    // with a different "salt" value for each colour
channel
    // pass that into sin() to get something between -1,
1
    // multiply by the grain variance to set how much
noise we want
    col.r += sin(rand(pos, 0.) * tau) * grainAmp;
    col.g += sin(rand(pos, 1.) * tau) * grainAmp;
    col.b += sin(rand(pos, 2.) * tau) * grainAmp;

    // Modified colour is our output
    gl_FragColor = col;
}
```


APÊNDICES

Código para "Cerrado" no p5.js:
<https://editor.p5js.org/beacoxta/sketches/uKnKMHpAm>

ou

SKETCH.JS

```
let noiseScale = 0.002; // nível de noise
let yStart = 0;
let yStep;
let randSeed;
let minNumOfLines = 300;
let maxNumOfLines = 500;
let myFilterShader;
let branchMin = 5;
let branchMax = 10;
let initialLines = 15;

function preload() { // filtro grain
  myFilterShader = loadShader("filter.vert",
"grain.frag");
}

function setup() {
  createCanvas(2160, 2160, WEBGL); //2x tela
  randSeed = random(100);
  angleMode(DEGREES);
  noiseSeed(randSeed);
  numOfLines = floor(random(minNumOfLines,
maxNumOfLines));
  yStep = height / numOfLines;
  generateLines();
  drawTree();

  myFilterShader.setUniform("grainAmp", 0.3);
  filterShader(myFilterShader);
}
```

```
function generateLines() { // ondas
  for (let i = 0; i < numOfLines; i++) {
    beginShape();
    let y = yStart + i * yStep;
    for (let x = 1; x < width; x++) {
      let noiseVal = noise(x * noiseScale, y *
noiseScale, randSeed);
      let zPos = map(noiseVal, 0, 1, -height, height);
      vertex(x - width / 2, y - height / 2, zPos);
    }
    endShape();
  }
}
```

```
function drawTree() { // árvore fractal
```

```
  let startCorner = floor(random(0, 4));

  let branchCount = floor(random(branchMin,
branchMax + 1));
```

```
  let startX, startY;
  if (startCorner === 0) {
    startX = 0;
    startY = height;
  } else if (startCorner === 1) {
    startX = width;
    startY = height;
  } else if (startCorner === 2) {
    startX = width;
    startY = 0;
  } else {
    startX = 0;
    startY = 0;
  }
}
```

```
for (let k = 0; k < initialLines; k++) {
  let angle = map(k, 0, initialLines, -180, 180);
  drawBranch(startX, startY, width / 4, angle,
branchCount);
}
}
```

```
function drawBranch(x, y, length, angle, branches) {
  if (branches === 0) {
    return;
  }
```

```
  let endX = x + length * cos(angle);
  let endY = y + length * sin(angle);
```

```
  line(x, y, endX, endY);
```

```
  drawBranch(endX, endY, length * 0.7, angle -
random(20, 40), branches - 1);
  drawBranch(endX, endY, length * 0.7, angle +
random(20, 40), branches - 1);
}
```

```
function draw() {
  background(50);
  //stroke (random(0, 255), random(0, 255),
random(0, 255), 100) // opção de linhas com cor
aleatória
  stroke(0)
  strokeWeight(3)
  //stroke(26, 36, 26, 100); // opção com linha verde
escura
  noFill();
  drawTree();
  stroke(230);
  strokeWeight(2)
  generateLines();
}
```

```
noStroke();
angleMode(RADIANS)
fill(random(0, 255), random(0, 255), random(0,
255), 30); // flores
for (let i = 0; i < 9; i++) {
  for (let f = 0; f < 600; f++) {
    let f = random(900);
  }
  ellipse(0, 0, 90, random((height-200), height));
  rotate(PI / 2);
  ellipse(0, 0, 90, random((height-200), height));
  rotate(PI / 3);
  ellipse(0, 0, 90, random((height-200), height));
  rotate(PI / 4);

}
```

```
noStroke();
fill(random(0, 255), random(0, 255), random(0,
255), 70); // círculo
let circled = random(width/3, width-200);
let circleX = random(-width/2, width/2);
let circleY = random(-height/2, height/2);
circle(circleX, circleY, circled);
```

```
myFilterShader.setUniform("grainAmp", 0.1);
filterShader(myFilterShader);
```

```
noLoop()
```

```
}
```

```
function keyTyped() { // clicar na imagem e digitar
"s" para salvar imagem
  if (key === "s") {
    save("lúcido.png");
  }
}
```


FILTER..VERT

```
lattribute vec3 aPosition;
attribute vec2 aTexCoord;

varying vec2 pos;

void main() {
    pos = aTexCoord;
    // flip the y axis
    pos.y = 1.0 - pos.y;

    vec4 positionVec4 = vec4(aPosition, 1.0);
    positionVec4.xy = positionVec4.xy * 2.0 - 1.0;

    gl_Position = positionVec4;
}
```

GRAIN.FRAG

```
#ifdef GL_ES
precision mediump float;
#endif

// Pixel position
varying vec2 pos;

// Uniforms set by filterShader
uniform sampler2D filter_background; // contains
the image being filtered
uniform vec2 filter_res; // contains the image
resolution in pixels

uniform float grainAmp; // how much variance in
grain texture

//uniform float millis; // time sketch has been
running

const float tau = 6.2831855; // 2 * PI

// The GPU has no random() function, so instead
// do some maths on the position of the pixel to
approximate a
// random number between 0 and 1
float rand(vec2 co, float salt){
    return fract(sin(dot(co, vec2(12.9898, 78.233) +
salt)) * 43758.5453);
}

void main() {
    // Read colour from image
    vec4 col = texture2D(filter_background, pos);
```

```
f // Use the rand function to create a random
number
    // with a different "salt" value for each colour
channel
    // pass that into sin() to get something between -1,
1
    // multiply by the grain variance to set how much
noise we want
    col.r += sin(rand(pos, 0.) * tau) * grainAmp;
    col.g += sin(rand(pos, 1.) * tau) * grainAmp;
    col.b += sin(rand(pos, 2.) * tau) * grainAmp;

    // Modified colour is our output
    gl_FragColor = col;
}
```