Behavior Research Methods (2023) 55:2423–2446
https://doi.org/10.3758/s13428-022-01900-w

The Order & Complexity Toolbox for Aesthetics (OCTA): A systematic approach to study the relations between order, complexity, and aesthetic appreciation

Eline Van Geert[1] · Christophe Bossens[1] · Johan Wagemans[1]

Accepted: 2 June 2022 / Published online: 28 September 2022

Abstract

Do individuals prefer stimuli that are ordered or disordered, simple or complex, or that strike the right balance of order and complexity? Earlier research mainly focused on the separate influence of order and complexity on aesthetic appreciation. When order and complexity were studied in combination, stimulus manipulations were often not parametrically controlled, only rather specific types of order (i.e., balance or symmetry) were usually studied, and/or the multidimensionality of order and complexity was largely ignored. Progress has also been limited by the lack of an easy way to create reproducible and expandible stimulus sets, including both order and complexity manipulations. The Order & Complexity Toolbox for Aesthetics (OCTA), a Python toolbox that is also available as a point-and-click Shiny application, aims to fill this gap. OCTA provides researchers with a free and easy way to create multi-element displays varying qualitatively (i.e., different types) and quantitatively (i.e., different levels) in order and complexity, based on regularity and variety along multiple element features (e.g., shape, size, color, orientation). The standard vector-based output is ideal for experiments on the web and the creation of dynamic interfaces and stimuli. OCTA will not only facilitate reproducible stimulus construction and experimental design in research on order, complexity, and aesthetics. In addition, OCTA can be a very useful tool in any type of research using visual stimuli, or even to create digital art. To illustrate OCTA's potential, we propose several possible applications and diverse questions that can be addressed using OCTA.

Keywords Order · Complexity · Aesthetics · Open-source software · Stimuli · Stimulus construction · Perceptual grouping · Perceptual organization · Reproducible stimuli · Python · Shiny app

Order, complexity, and the balance between order and complexity have long been considered important factors influencing aesthetic appreciation (for a review, see Van Geert & Wagemans, 2020). Although most empirical work has confirmed the importance of order and complexity, concrete results on the extent and direction of order and complexity influences, as well as the relationship between order and complexity, vary widely. Previous research has some limitations concerning the stimuli and manipulations used, however, and leaves some important research options uncovered (cf. The need for OCTA below). The Order & Complexity Toolbox for Aesthetics (OCTA) fills these gaps by providing researchers an easy, open, and reproducible way to create stimuli varying in order and complexity on multiple element features (see Fig. 1 for some OCTA example stimuli). Being available as both a Python package and a Shiny application, OCTA[1] is accessible to both researchers with and without programming experience.

We will first discuss how we define order and complexity, and explain how we view the relation between order, complexity, and appreciation. Next, we list existing stimulus sets and stimulus generation tools (to be) used in research on the perception and appreciation of order and complexity, point out the limitations to the existing research and tools, and indicate how OCTA can provide a fulfilling answer to the existing gaps. Then, we introduce the newly created stimulus generation toolbox and discuss its steps, concepts, and options in more detail. Afterwards, we treat more advanced uses of

✉ Eline Van Geert
eline.vangeert@kuleuven.be

[1] Laboratory of Experimental Psychology, Department of Brain and Cognition, KU Leuven, Tiensestraat 102 - box 3711, 3000 Leuven, Belgium

[1] We will use octa in small letters when referring specifically to the Python package and OCTA in capital letters when referring to the toolbox in general, regardless of whether it concerns the Python implementation or the Shiny app.

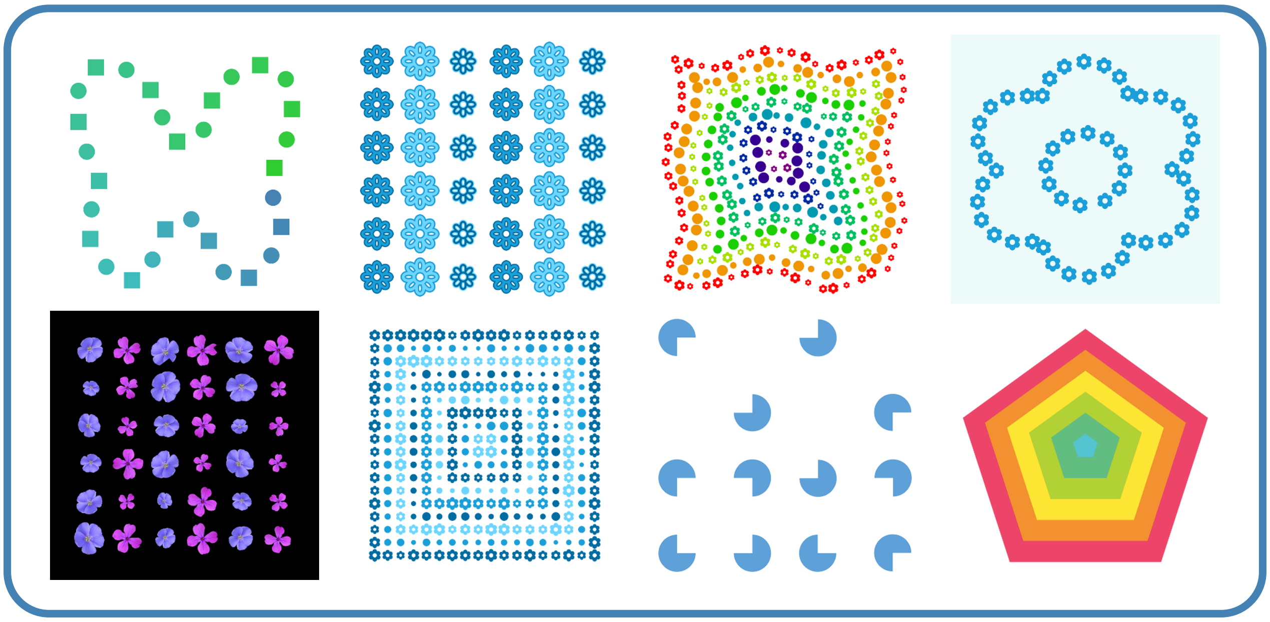


Fig. 1 Example stimuli created in OCTA. Clicking a stimulus leads to the octa code used to generate it. To view dynamic and interactive versions of some of these stimuli, visit this webpage. Figure licensed under CC BY 4.0 by the authors. Retrieved from https://doi.org/10.6084/m9.figshare.17708702. The bottom left example stimulus contains natural flower images from Hůla and Flegr (2016)

OCTA, provide a range of potential applications for OCTA, list some possible future extensions of OCTA, and give advice on how to start using OCTA.

Defining order and complexity

In this work, we define (stimulus) order as all aspects related to the *structure* and *organization* of information (in a stimulus) and (stimulus) complexity as all aspects related to the *quantity* and *variety* of information (in a stimulus; Van Geert & Wagemans, 2020). For example, Fig. 2a and b have the same number of distinguishable elements and the same dissimilarity between those elements (i.e., complexity; Berlyne, 1960) but differ in the way the elements are arranged (i.e., order). Similarly, also Fig. 2c and d share the same level of complexity, but differ in the way the elements are organized. In Fig. 2a and b, the difference in order is a *qualitative* difference, a difference in the *type* of order applied. In Fig. 2c and d, it concerns a *quantitative* difference, a difference in the *level* of order present. Figure 2a and c have the same overall organization (i.e., level and type of order), but differ in the quantity and variety of information present (i.e., different level of complexity).

In this example, we referred to the multidimensional concepts of *element* order and *element* complexity as referring to a combination of order and complexity on the color, shape, and size dimensions: we focused on the order and complexity of the distinguishable elements present in the stimulus on the different feature dimensions in combination. It is however also possible to investigate other forms of order and complexity in the stimulus, for example position order and complexity, specific feature order and complexity, or the complexity of individual feature values used. Qualitatively different position patterns can be used to place the elements in the stimulus (i.e., type of position order), and these positions can also show more or less diversity (e.g., equally spaced positions in rows and columns versus completely random positions; position complexity). Feature order and complexity are part of element order and complexity, but focus on one specific element feature at a time. For example, Fig. 2a has a higher level of size complexity than Fig. 2c, but b and d have a higher level of orientation complexity. Furthermore, the individual feature values of the elements present in the stimulus, or the features of the stimulus as a whole, may be more or less complex. For example, a safety pin may be a more complex shape than a circle, or placing the elements on a circular outline may form a less complex overall organization than placing them on a rectangular outline. Acknowledging the multidimensionality of order and complexity may be important in the search for more consistent relationships between order, complexity, and aesthetic appreciation (Nadal, Munar, Marty, & Cela-Conde, 2010; Van Geert & Wagemans, 2020).

Important to understand is that order and complexity are no direct opposites, and can be concurrently present, either on the same or on different feature dimensions. We distinguish order from simplicity (i.e., opposite of complexity) and complexity from disorder (i.e., opposite of order). Complexity on a feature dimension allows for a broader range of order levels and a larger set of order types to be present than is the case for

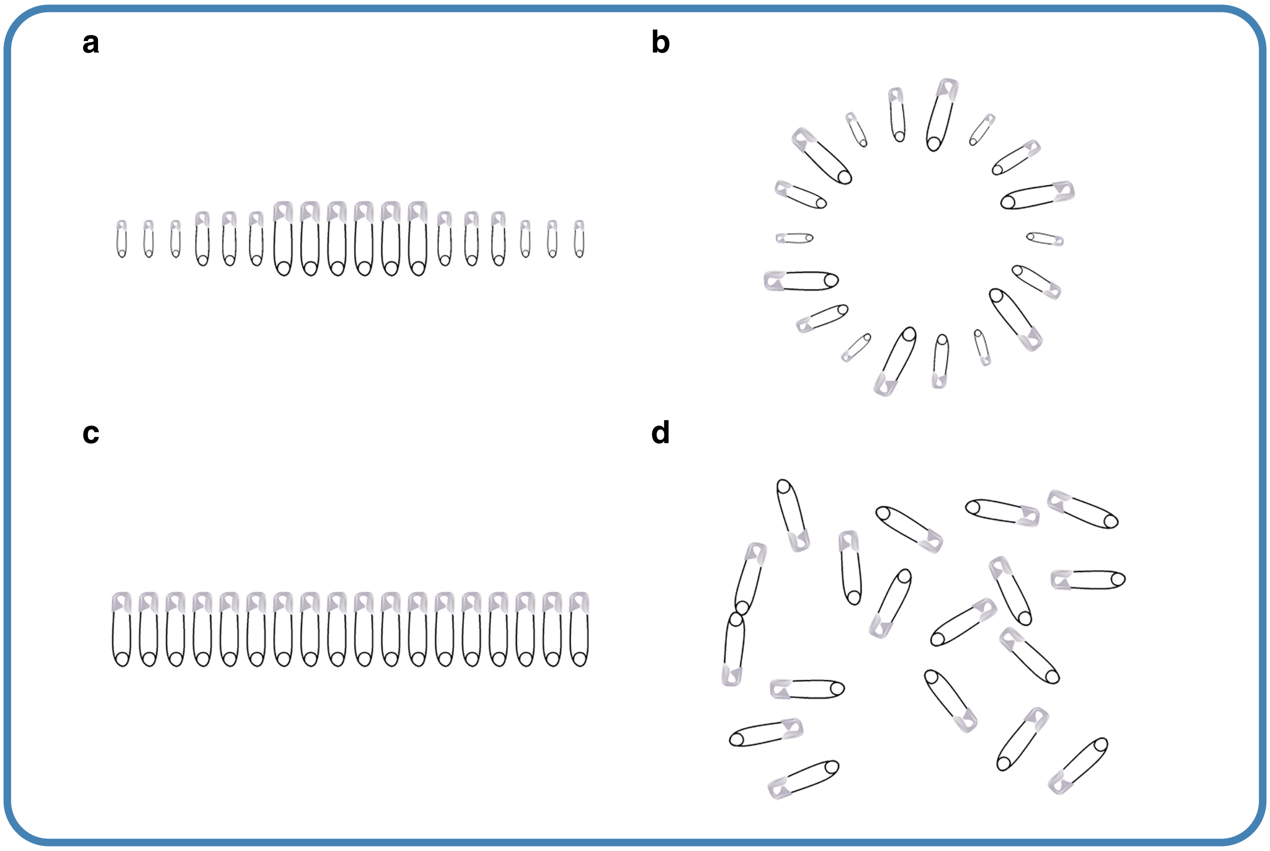


Fig. 2 Order and complexity on different stimulus dimensions, illustrated with stimuli created in OCTA. Clicking a stimulus leads to the octa code used to generate it. Figure licensed under CC BY 4.0 by the authors. Retrieved from https://doi.org/10.6084/m9.figshare.19634649. Vector image used to generate the OCTA stimuli: a safety pin created by Clker-Free-Vector-Images available on Pixabay

simplicity. For example, the safety pins in Fig. 2a can be arranged in more qualitatively and quantitatively different ways on the size dimension than the safety pins in Fig. 2c. However, simplicity on the size dimension does not prohibit disorder on other feature dimensions (e.g., orientation of the elements in Fig. 2d). As explained in Van Geert and Wagemans (2020), we view order and complexity as partial complements and partial opposites. On the one hand, complexity allows order to show its potential and order helps to make highly complex stimuli 'digestible' and aesthetically appreciated. Order and complexity thus need each other's presence to optimize appreciation. On the other hand, order and complexity can also relate negatively: uniformity for example indicates a low level of complexity, but also implies a high level of order on that feature dimension. Similarly, different types of symmetry may be seen as different types of order, but typically also reduce the range of complexity present in the display. Given this complex relationship between order and complexity, it is important to study their relation to aesthetic appreciation in combination, not separately (Van Geert & Wagemans, 2020).

One additional distinction that is relevant here, is the distinction between objective and subjective definitions and measures of order and complexity. Whereas *objective* order and complexity refer to the level and type of order or complexity that is physically present in the stimulus, *subjective* order and complexity entail the individuals' perception of the order or complexity present in the stimulus (Van Geert & Wagemans, 2020). It is important to make this distinction, as research results may differ depending on how order and complexity are defined and/or measured: the concepts of objective order and complexity may for example be more easily separable from each other than the concepts of subjective order and complexity. Although many objective measures for complexity exist (for overviews, see Donderi, 2006; Van Geert & Wagemans, 2020), the set of objective measures available for order is rather limited (Van Geert & Wagemans, 2020). Since Arnheim's (1971) seminal essay on "Entropy and art" (which was effectively about disorder and order in art), several scientists have proposed measures of order (and disorder) derived from information theory and dynamical systems theory. For instance, Redies and colleagues have defined and computed edge-orientation entropy or anisotropy in relation to preference for patterns and images of paintings (e.g., Grebenkina, Brachmann, Bertamini, Kaduhm, & Redies, 2018; Redies, Brachmann, & Wagemans, 2017).

However, a discussion of these would be beyond the scope of the present paper. The relative lack of objective order measures may be due to the difficulty of quantifying aspects of order other than perceptual balance or symmetry in an objective or automatized way. It is our intuition that the degree of order can be quantified for specific types of order (such as balance and symmetry) but that further research is needed to compare the degree of perceived order for different types of order. Our aim is to facilitate this research by providing a toolbox to systematically generate different types of order (e.g., different pattern types, different types of repetitions vs. changes) to compare their overall degree of perceived order (and how these interact with the number of elements, their degree of similarity and variety, etc.). Below we give an overview of different stimulus sets used in earlier research on the perception and appreciation of order and complexity that attempted to manipulate order or complexity in a parametric way.

Previous parametric stimulus sets in research on the perception and appreciation of order and complexity

Studies investigating the perception or appreciation of *order* often focused on different types of symmetry or on different measures of perceptual balance (see Fig. 3). Chipman and Mendelson (1979) chose black and white square patterns with different types of order: unstructured, horizontal and/or vertical symmetry, diagonal symmetry, checkerboard organization, and rotational organization. Locher, Stappers, and Overbeeke (1998) asked participants to create designs consisting of a set of either nine circles, squares, rectangles, or leaves varying in size (three large, three medium, three small forms per set). They investigated the changes in the center of balance as the design was completed. Wilson and Chatterjee (2005) created stimuli using the proprietary software Adobe Photoshop 7.0 and then selected a subset of these stimuli for their experiments based on a calculated balance score (i.e., the average score of eight measures of symmetry). These stimuli were later also used by Hübner and Fillinger (2016). Gollwitzer, Marshall, Wang, and Bargh (2017) generated unbroken and broken patterns to study pattern deviancy aversion in the context of social psychological research on prejudice. Although many studies in empirical aesthetics compared symmetric and asymmetric stimuli (Bertamini & Rampone, 2020), only some recent studies investigated symmetry perception or appreciation using the different systematic characterizations of symmetry in mathematics (e.g., Alp, Kohler, Kogo, Wagemans, & Norcia, 2018; Clarke, Green, Halley, & Chantler, 2011; Kohler, Clarke, Yakovleva, Liu, & Norcia, 2016; Martin, Uy, Kvapil, & Friedenberg, 2020). Four basic symmetrical transformations are defined for two-dimensional shapes (translation, rotation, reflection, glide reflection; Grünbaum & Shephard, 1989). With these transformations, seven distinct border patterns or frieze groups (i.e., two-dimensional designs with translations in only one direction) and seventeen distinct wallpaper pattern types (i.e., two-

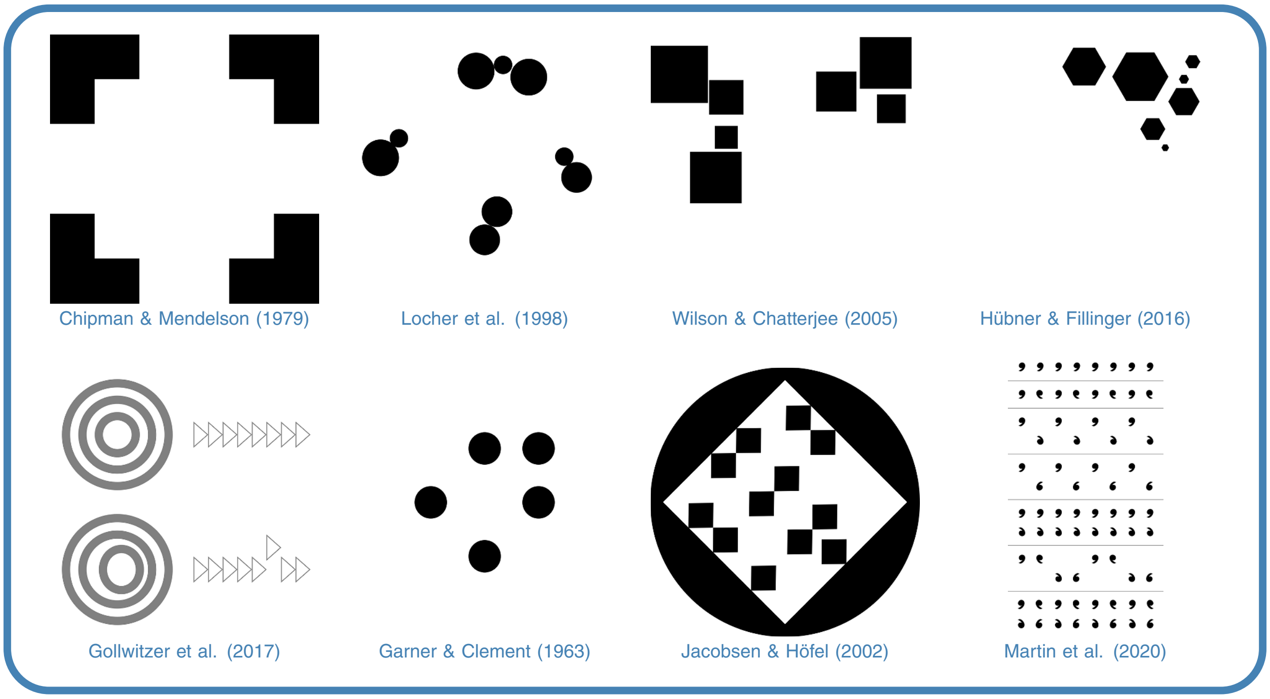


Fig. 3 Example stimuli from previous research on order and complexity recreated in OCTA. Clicking a stimulus leads to the octa code used to generate it. Figure licensed under CC BY 4.0 by the authors. Retrieved from https://doi.org/10.6084/m9.figshare.17739860

dimensional designs with translations in two independent directions) can be specified (Grünbaum & Shephard, 1989; Thomas, 2012). Martin et al. (2020) compared beauty ratings for each of the seven frieze patterns using a comma, flag, and a filled texture as elements. Clarke et al. (2011) studied the perceptual similarity of the seventeen wallpaper pattern types. They created exemplars starting from a white noise (triangular, rectangular, or square) fundamental region and combined them into a rectangular repeating tile to produce the patterns (Alp et al., 2018).

Studies focusing on the perception or appreciation of *complexity* have used a broad range of different stimulus types. Some studies used ecologically valid images and calculated statistical image properties to indicate complexity (e.g., Braun, Amirshahi, Denzler, & Redies, 2013). Other studies used different types of fractal images (e.g., Bies, Blanc-Goldhammer, Boydston, Taylor, & Sereno, 2016; Spehar, Walker, & Taylor, 2016). Sun and Firestone (2021) used a measure of structural surprisal (i.e., the level of surprise associated with a shape's skeletal structure) to categorize abstract shapes as simple and complex while controlling for the number of sides. The Matlab code to generate the shapes for that study was shared publicly on the Open Science Framework.

When both *order and complexity* were manipulated parametrically, researchers often used random polygons (e.g., Arnoult, 1960; Attneave, 1957), black and white square patterns (e.g., Chipman, 1977; Smets, 1973), or dot patterns (e.g., Garner & Clement, 1963; Hamada & Ishihara, 1988). Attneave (1957) constructed random polygon shapes based on a varying number of turns (i.e., complexity manipulation) and manipulated them to be symmetrical or asymmetrical (i.e., order manipulation). Attneave and Arnoult (1956) provided a series of methods for the construction of random shapes. This method was later used, for example by Vanderplas and Garvin (1959), to generate random shapes of six levels of complexity (i.e., varying number of points). Smets (1973) created patterns of black and white squares in which both subjective redundancy (i.e., the percentage of correctly selected black and white elements when reproducing the pattern, related to order) and maximal information (i.e., the number of independent elements, related to complexity) were varied. These patterns were also used in studies by Berlyne (1974) and Cupchik and Berlyne (1979). Chipman (1977) used similar but smaller patterns and varied structure and complexity by selecting different stimulus sets by hand. Garner and Clement (1963) produced 90 patterns by placing five dots in an imaginary 3-by-3 square matrix, with the restriction of at least one dot in each row and column, and investigated the influence of the number of patterns in the equivalence group for each pattern (i.e., the number of different patterns that can be created by rotation or reflection of the pattern) as well as symmetry. Hamada and Ishihara (1988) created rotation and reflection invariant dot patterns using imaginary rectangular and hexagonal frameworks. The dot patterns varied in the number of dots they consisted of, the order of the symmetry groups as well as the symmetry group they belonged to (i.e., cyclic and dihedral groups). In more recent years, new stimulus sets focused on symmetry and complexity manipulations. Jacobsen and Höfel (2002) constructed stimuli consisting of a solid black circle showing a centered, quadratic, rhombic cut-out and an arrangement of 86 to 88 small black triangles. Half of the patterns created were symmetric (four possible symmetry axes), the other half were not symmetric. Gartus and Leder (2013) created a new set of abstract black and white patterns using a simulated annealing stochastic optimization algorithm. They varied the number of objects (i.e., complexity manipulation) as well as the type of symmetry: no symmetry, and full and broken symmetry patterns with one, two, and four symmetry axes.

Existing software to generate aesthetic stimuli

FlexTiles

Westphal-Fitch and colleagues (2012) created FlexTiles, a custom-written image manipulation program, in which each of 36 tiles can be rotated to four possible orientations (0°, 90°, 180°, 270°). Participants were asked to rotate the tiles as much or as little as they liked. Patterns that emerged showed rotation, translational symmetry, bilateral symmetry along vertical axis, rotational symmetry, symmetry along diagonal axis, or local linear groupings with no overall symmetry. In later experiments, they created flawed and unflawed versions of patterns, with violations determined by color and/or orientation. Muth, Westphal-Fitch, and Carbon (2019) also made use of the FlexTiles software and asked participants to produce patterns that would be liked or would be interesting. A separate group of participants rated the produced stimuli on liking, interestingness, complexity, and order. Although Muth, Westphal-Fitch, and Carbon (2019) shared the stimuli used in their experiments, the FlexTiles software to recreate or adapt the stimuli has not been made openly available for use by other researchers. For OCTA stimuli created in the style of FlexTiles, see Fig. 4.

Statistical geometry sampler

Güçlütürk, Jacobs, and van Lier (2016) generated statistical geometric patterns using a space filling algorithm that places non-overlapping geometric shapes that monotonically decrease in size in a random fashion on a canvas (Shier, 2011; Shier & Bourke, 2013). A parameter c determining the size of the first shape element and the speed with which the size

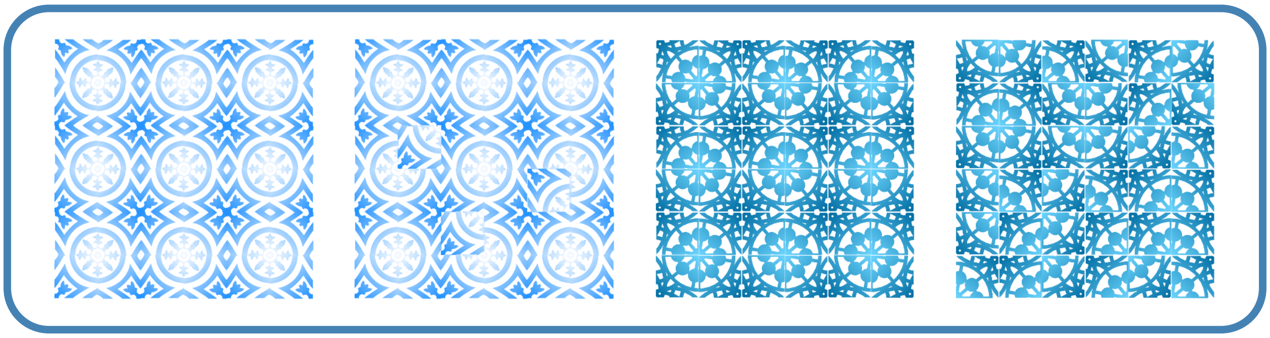


Fig. 4 Example stimuli in the style of FlexTiles recreated in OCTA. Clicking a stimulus leads to the octa code used to generate it. Figure licensed under CC BY 4.0 by the authors. Retrieved from https://doi.org/10.6084/m9.figshare.17740640. Vector images used to generate the OCTA stimuli: a tile created by Jorge Carrillo and a tile created by Lluisa Iborra available on the Noun Project

decreased was manipulated as well as the geometric shape used (i.e., circle, hexagon, square, and triangle). Although the algorithm, documented C code, and a lot of example stimuli are openly available on the website of the authors of the algorithm, and the authors mention to be open to support people interested to get started on their own images, no easy interface to create new stimuli is provided.

Aesthetic abstract textures generator

One tool that does generate reproducible stimuli for use in aesthetics is the random abstract texture generator of Alvarez, Monzón, and Morel (2021). Jean-Michel Morel, Luis Alvarez, and colleagues did design a point-and-click online tool to generate aesthetic stimuli based on composition principles and random sampling (Alvarez, Gousseau, Morel, & Salgado, 2015; Alvarez, Monzón, & Morel, 2021).[2] This tool is meant for artists and designers to explore and test new styles, and does not allow users to manually specify the location, color, or shape of any element in the display. The shape elements are placed automatically based on the chosen composition principles and random sampling. Although this tool is very useful for the generation of reproducible aesthetic stimuli in general, it does not allow for individually controlled parametric order and complexity manipulations on different element features.

The need for OCTA (see Fig. 5)

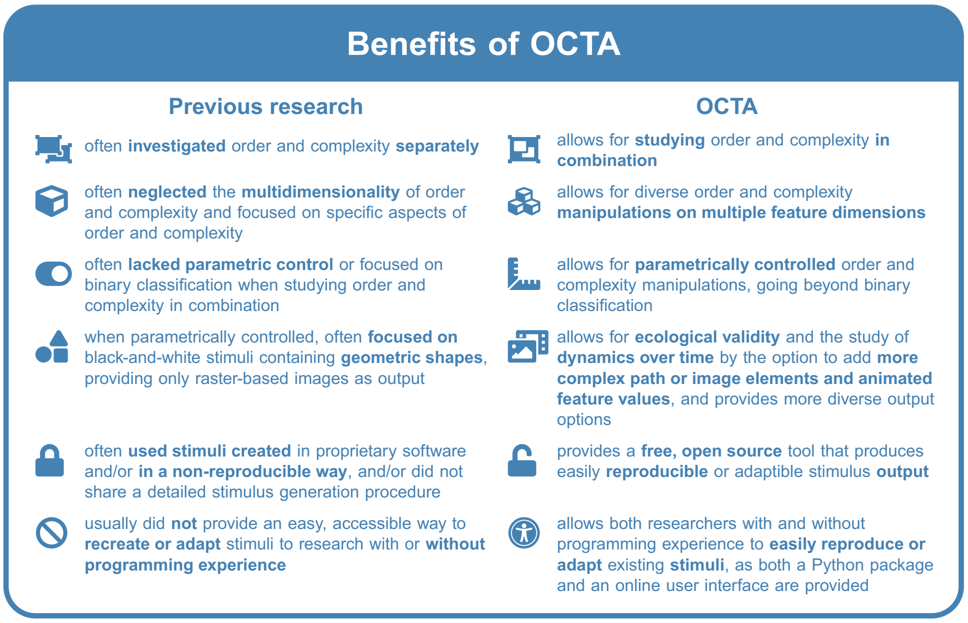
As may be clear from the examples given, the existing research on the perception and appreciation of order and complexity leaves important gaps. First, most empirical work investigated the relation of order and complexity to aesthetic appreciation separately rather than in combination. Second, the relation of appreciation with complexity has been studied much more frequently than the relation with order and studies usually focused on specific aspects of order (i.e., symmetry and perceptual balance). However, many other types of organization exist (e.g., similarity grouping, proximity grouping, alternation/iteration, systematic alteration/gradient) and their influence on aesthetic appreciation has not yet been systematically studied (Van Geert & Wagemans, 2020). Furthermore, the multidimensionality (e.g., order and complexity of element colors, shapes, and sizes) of both order and complexity has often been neglected (Nadal et al., 2010; Van Geert & Wagemans, 2020). To our knowledge, no single existing stimulus set combines systematic manipulations of order and complexity on different element features (e.g., shape, size, color, orientation). Findings based on earlier research not distinguishing between different element features may be tied to the specific aspect of order or complexity that was investigated, and may not generalize to other order or complexity dimensions.[3] Third, when order and complexity were studied in combination, researchers often focused on a binary classification in high and low order and/or complexity rather than more fine-grained order or complexity manipulations,[4] or more ecologically valid stimuli were used but then parametric control over both order and complexity dimensions was lost (e.g., Nadal et al., 2010; Van Geert & Wagemans, 2021). Findings based on this last type of research, using less controlled stimuli, require replication with a more parametrically controlled stimulus set where both order and complexity of the stimulus can be manipulated as independently as possible.[5] Fourth, when parametrically varied stimulus sets were used, the focus

[2] The same authors also created an abstract shape generator which may be of interest to the readers.

[3] This will always be the case to some extent but experimental control and generalization are made much easier in OCTA because it provides easy handles to create and test systematically many more variations.

[4] Whether a more continuous manipulation of order levels also leads to a continuous change in perceived order is an empirical question that can be addressed using the OCTA toolbox.

[5] Although it is important to study order and complexity *in combination* (i.e., in a factorial design taking into account both order and complexity), we do believe it is important to study order and complexity *as independently as possible* (i.e., in a non-confounded way, separating both factors as much as possible within the same experiment).



Benefits of OCTA

Previous research	OCTA
often **investigated** order and complexity **separately**	allows for **studying** order and complexity **in combination**
often **neglected** the **multidimensionality** of order and complexity and focused on specific aspects of order and complexity	allows for diverse order and complexity **manipulations on multiple feature dimensions**
often **lacked parametric control** or focused on binary classification when studying order and complexity in combination	allows for **parametrically controlled** order and complexity manipulations, going beyond binary classification
when parametrically controlled, often **focused on** black-and-white stimuli containing **geometric shapes**, providing only raster-based images as output	allows for **ecological validity** and the study of **dynamics over time** by the option to add **more complex path or image elements and animated feature values**, and provides more diverse output options
often **used stimuli created** in proprietary software and/or **in a non-reproducible way**, and/or did not share a detailed stimulus generation procedure	provides a **free, open source** tool that produces easily **reproducible** or adaptible stimulus **output**
usually did **not** provide an easy, accessible way to **recreate or adapt** stimuli to research with or **without programming experience**	allows both researchers with and without programming experience to **easily reproduce or adapt** existing **stimuli**, as both a Python package and an online user interface are provided

Fig. 5 Benefits of OCTA. Figure licensed under CC BY 4.0 by the authors. Retrieved from https://doi.org/10.6084/m9.figshare.17740904

was often on black-and-white stimuli containing geometric shapes. Fifth, the stimuli used in aesthetics research were often created using proprietary software, in a non-reproducible and non-adaptable way, or not openly available to the research community, or the researchers did not share enough details concerning how the stimuli were generated. Sixth, in case a detailed stimulus generation procedure was provided, stimuli could not be recreated or adapted easily without programming experience. Therefore, research on order and complexity would benefit from an easy way to create standardized stimuli varying qualitatively and quantitatively in many different aspects of order and complexity, with options for increased ecological validity but without losing parametric control.

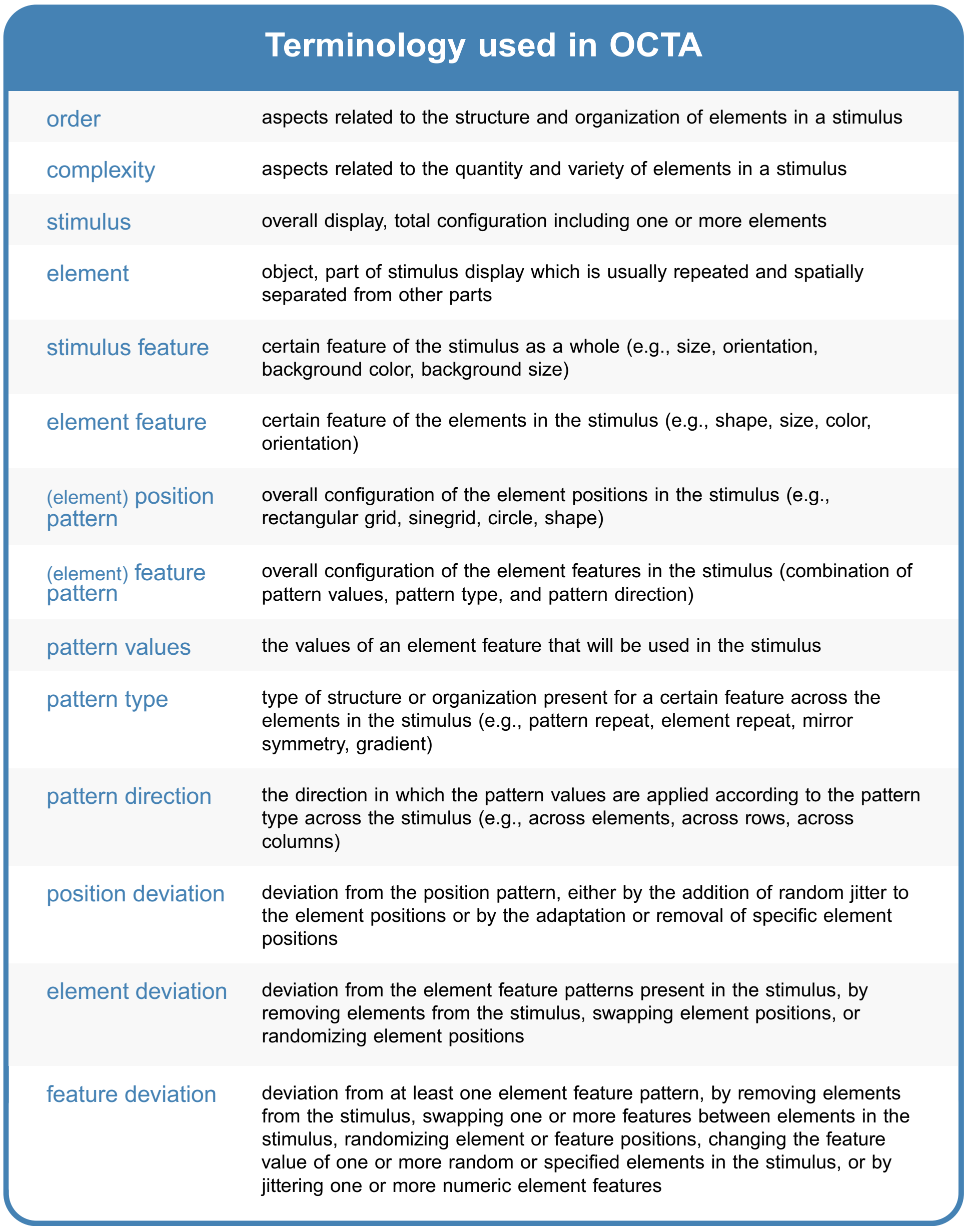
The Order & Complexity Toolbox for Aesthetics (OCTA) is such a free, openly available tool and creates many opportunities to investigate order and complexity in a standardized and multi-dimensional manner, with a focus on multi-element displays. With OCTA, one can manipulate and measure order and complexity in a systematic way, and study their effects in combination. OCTA acknowledges the multidimensionality of order and complexity by allowing separate manipulations of different element features (e.g., shape, size, color, orientation). It allows for parametric control on many different stimulus and element features. It also acknowledges the potential of research on the dynamics of order and complexity by including animation options for several element features including color, size, and orientation. To provide more ecological validity, OCTA makes it possible to use of images or path elements[6] in a stimulus. OCTA is freely available, open source, and enables users to create reproducible, code-based stimulus sets that are easily adaptable or extendible. The standard vector-based output is ideal for experiments on the web and the creation of dynamic interfaces and stimuli, while raster-based output is possible as well. Furthermore, OCTA is accessible to both researchers with and without programming experience, as both a Python package and an online user interface are provided. With OCTA being fast, flexible, and transparent, we strongly believe that OCTA will facilitate reproducible stimulus construction and experimental design in research on order, complexity, and aesthetics. In addition, OCTA can be a very useful tool to investigate visual perceptual organization, to create visual stimuli for any type of experimental or cognitive task, or even to create digital art.

The OCTA toolbox as a Python package and a Shiny application

The octa package is a Python package developed in Python 3.8 (Van Rossum & Drake, 2009) and is available (on the Python Package Index or on GitHub) as open-source software under the GNU Lesser General Public License (Version 3) as published by the Free Software Foundation.[7] We chose to create this toolbox

[6] Path SVG elements can be used to draw any type of lines, curves, or custom shapes based on a combination of straight or curved lines. For more information on how to define paths, have a look at these online tutorials by W3Schools and Mozilla.

[7] To make full use of octa, Python 3.6 or higher is required, and the toolbox depends on the Python packages svgwrite (Moitzi, 2021), svg.path (Regebro, 2021), svgpathtools (Port, 2021), svgutils (Telenczuk, 2021), jsonpickle (Aguilar, 2021), html2image (vgalin, 2021), svglib (Gherman, 2021), reportlab (Robinson, Becker, the ReportLab team, & the community, 2021), colour (Lab, 2021), and IPython (Pérez & Granger, 2007). In addition, Google Chrome (Windows, MacOS) or Chromium Browser (Linux) needs to be installed to be able to generate PNG, JPG, or TIFF raster images based on the stimulus.



Terminology used in OCTA

order	aspects related to the structure and organization of elements in a stimulus
complexity	aspects related to the quantity and variety of elements in a stimulus
stimulus	overall display, total configuration including one or more elements
element	object, part of stimulus display which is usually repeated and spatially separated from other parts
stimulus feature	certain feature of the stimulus as a whole (e.g., size, orientation, background color, background size)
element feature	certain feature of the elements in the stimulus (e.g., shape, size, color, orientation)
(element) position pattern	overall configuration of the element positions in the stimulus (e.g., rectangular grid, sinegrid, circle, shape)
(element) feature pattern	overall configuration of the element features in the stimulus (combination of pattern values, pattern type, and pattern direction)
pattern values	the values of an element feature that will be used in the stimulus
pattern type	type of structure or organization present for a certain feature across the elements in the stimulus (e.g., pattern repeat, element repeat, mirror symmetry, gradient)
pattern direction	the direction in which the pattern values are applied according to the pattern type across the stimulus (e.g., across elements, across rows, across columns)
position deviation	deviation from the position pattern, either by the addition of random jitter to the element positions or by the adaptation or removal of specific element positions
element deviation	deviation from the element feature patterns present in the stimulus, by removing elements from the stimulus, swapping element positions, or randomizing element positions
feature deviation	deviation from at least one element feature pattern, by removing elements from the stimulus, swapping one or more features between elements in the stimulus, randomizing element or feature positions, changing the feature value of one or more random or specified elements in the stimulus, or by jittering one or more numeric element features

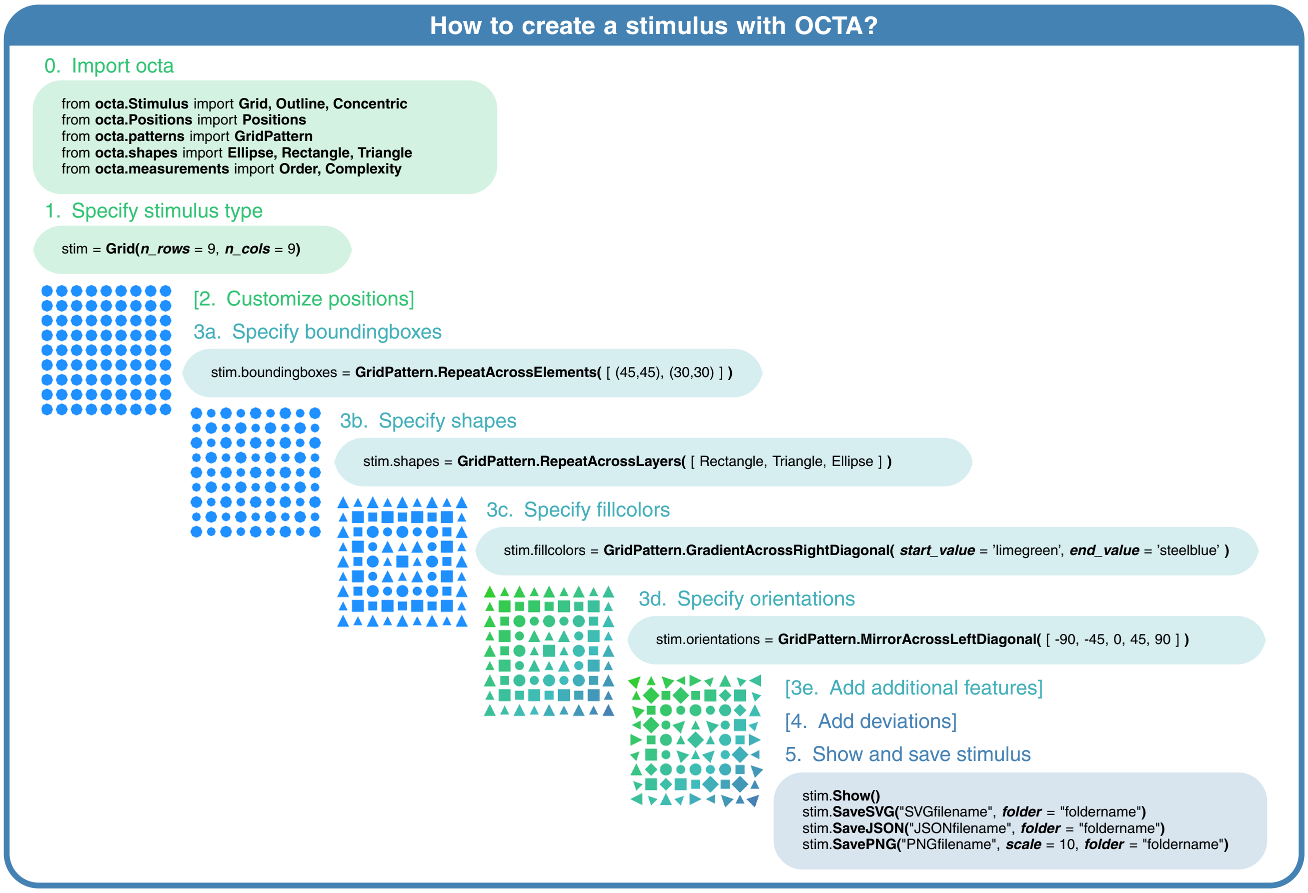
Fig. 6 Terminology used in OCTA. Figure licensed under CC BY 4.0 by the authors. Retrieved from https://doi.org/10.6084/m9.figshare.18102443

in Python as Python is a powerful and popular open-source programming language for which a large online community is available. Elaborate documentation on the octa Python package is available, offering a step-by-step introduction to the full functionality of the OCTA toolbox. To get users started, a website with example stimuli (see dynamic stimuli here) and the corresponding code is provided. For users who want to dig further into the toolbox, detailed function documentation is available. Besides developing the Python package, we also developed a Graphical User Interface for the OCTA toolbox, in the form of a Shiny application.[8] The procedure described below will be for using the Python package, but most aspects are similar in the Shiny application (we will indicate the most important differences in text). Additional instructions for the use of the Shiny application are provided in the app itself.

Figure 6 gives an overview of the terminology used in OCTA.[9] In OCTA, you can create *stimuli* consisting of multiple *elements*. First, the user specifies the *type* of stimulus they want

[8] When using either the octa Python package or the OCTA Shiny app in your (academic) work, please cite this paper to acknowledge the authors.

[9] We realize that it is quite difficult to understand and remember this terminology. This difficulty follows from the wide range of stimulus variations we want OCTA to be able to generate. First-time users might be put off by this difficulty. However, we suggest you try working with the toolbox to familiarize yourself with the options, and then Fig. 6 and the descriptions in the main text become a handy reference in case you want to develop a more refined understanding all the available options or gain more experience with them.



```
from octa.Stimulus import Grid, Outline, Concentric
from octa.Positions import Positions
from octa.patterns import GridPattern
from octa.shapes import Ellipse, Rectangle, Triangle
from octa.measurements import Order, Complexity
```

```
stim = Grid(n_rows = 9, n_cols = 9)
```

```
stim.boundingboxes = GridPattern.RepeatAcrossElements( [ (45,45), (30,30) ] )
```

```
stim.shapes = GridPattern.RepeatAcrossLayers( [ Rectangle, Triangle, Ellipse ] )
```

```
stim.fillcolors = GridPattern.GradientAcrossRightDiagonal( start_value = 'limegreen', end_value = 'steelblue' )
```

```
stim.orientations = GridPattern.MirrorAcrossLeftDiagonal( [ -90, -45, 0, 45, 90 ] )
```

```
stim.Show()
stim.SaveSVG("SVGfilename", folder = "foldername")
stim.SaveJSON("JSONfilename", folder = "foldername")
stim.SavePNG("PNGfilename", scale = 10, folder = "foldername")
```

Fig. 7 Tutorial on how to create a stimulus in OCTA. Clicking a stimulus leads to the octa code used to generate it. Figure licensed under CC BY 4.0 by the authors. Retrieved from https://doi.org/10.6084/m9.figshare.17743082

to create (i.e., Grid, Outline, Concentric; see Fig. 8) as well as the general *stimulus features* they want to adapt (e.g., stimulus size, background color or shape, stimulus orientation). Second, they can replace the default *position pattern* for the stimulus type by a custom one (e.g., sine grid, custom shape, custom positions). Third, the *element feature patterns* can be specified by defining *pattern type*, *pattern direction*, and *pattern values* per element feature. Fourth, *position, element, and feature deviations* can be added. Finally, the user can save the resulting stimulus in the preferred formats: as a vector-based image (SVG), as a raster-based image (PNG, JPG, PDF, or TIFF), and/or as a computer-readable file (JSON). The SVG format is recommended for online use, as it has the same quality at all viewing sizes and keeps the possibility for animated element and stimulus features. The JSON output can be used to recreate the stimulus in Python using the OCTA toolbox without the original code (using the LoadFromJSON function).[10] In the Shiny app, the user can also view or download the Python code needed to reproduce the current stimulus with the octa package in Python.

Step-by-step guide to creating a stimulus using OCTA (see Fig. 7)

The following section gives a more detailed overview of the steps of creating a stimulus (set) in OCTA.[11] For a visual summary of the different steps needed to create a first stimulus in OCTA, consult Fig. 7.

Step 0. Load the octa package in Python

To start using the octa Python package, install octa as well as its dependencies (svgwrite, svg.path, svgpathtools, svgutils, jsonpickle, html2image, svglib, reportlab, colour, and IPython). Once everything is installed, it can be helpful to

[10] Keep in mind that to reproduce the exact same stimulus, one needs to set a seed before running (each line of) the OCTA Python code and also keep this seed (these seeds) along with the JSON file in case any randomization procedures are used in the stimulus generation. In addition, the safest way to ensure reproducibility is by keeping track of the code, seed, and OCTA toolbox version you used to create your stimuli.

[11] In line with footnote 10, the following description of the available options (especially in steps 2–4) will probably be overwhelming for most first-time users. The best way to come to grips with this is to first check the figures (especially Figs. 7-15) for a quick visual impression of the available options and then to read the accompanying text for more details about specific possibilities. Again, this description will become more understandable and useful after having gained some initial experience with OCTA.

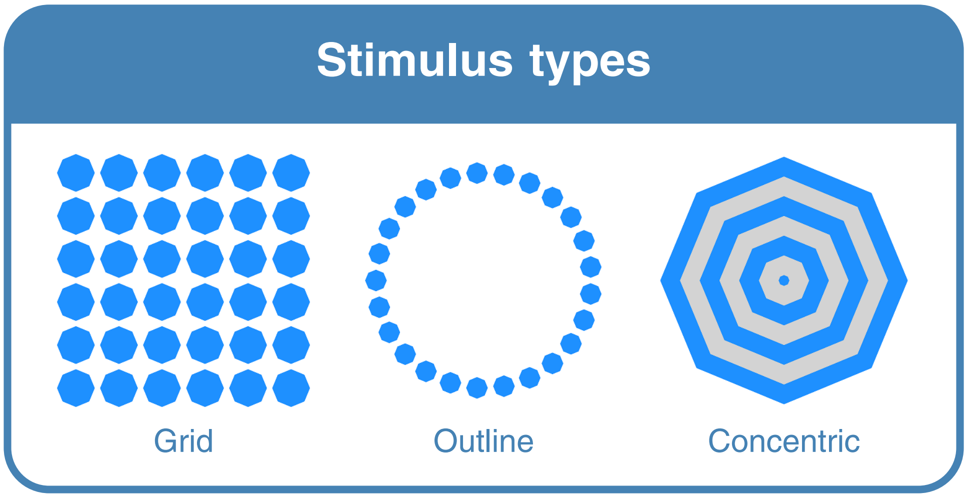


Fig. 8 Stimulus types available in OCTA. Clicking a stimulus leads to the octa code used to generate it. Figure licensed under CC BY 4.0 by the authors. Retrieved from https://doi.org/10.6084/m9.figshare.17748872

import specific functions from the octa package. In the Shiny app, this step is unnecessary as all required packages are loaded automatically. Furthermore, the user can download the Python code from the Shiny app and copy it to Python to import the functions from the octa package necessary for a specific stimulus.

Step 1. Specify a stimulus type

The first step when creating a stimulus is to define the type of stimulus one wants to create: a Grid stimulus, an Outline stimulus, or a Concentric stimulus (see Fig. 8).[12] These stimulus types correspond to frequently used stimulus types in the literature. For instance, Grid stimuli have been used by Garner and Clement (1963) and Chipman and Mendelson (1979), and have formed the basis for FlexTiles and many stimuli in visual search tasks. Outline stimuli could form the basis for hierarchical stimuli like those used by Navon (1977), Poirel, Pineau, and Mellet (2006), and Krakowski et al. (2016). For Concentric stimuli, see for instance Gollwitzer et al. (2017). Whereas a Grid stimulus requires a specified number of rows and columns, Outline or Concentric stimuli require a specified number of elements.[13] If desired, the user can specify additional arguments to customize the stimulus. The following features of the stimulus as a whole can be specified: spacing between rows and columns (in case of a Grid stimulus), shape and shape bounding box (in case of an Outline stimulus), horizontal and vertical margin (when automatic sizing method is used), stimulus size (when fixed size is needed), background color, background shape, stimulus orientation, mirror value for the stimulus as a whole, mask to apply for the stimulus as a whole, and link, class label, and id label for the stimulus. By default, stimuli will be autocentered with a horizontal and vertical margin of 20 units in the current user coordinate system (i.e., user units), have a white background color, and an orientation of 0°. The Grid category has a default row and column spacing of 50 user units, and the Outline category by default has a circular shape with a bounding box of 150 by 150 user units. For more information on all stimulus features, please consult the online documentation. In the Shiny app, additional stimulus features can be specified under the tab '0. Add stimulus features.' The general stimulus type has been merged with the other position pattern options however, and can be changed under the tab '1. Add position pattern.'

Step 2. Specify element positions

Having created the stimulus, the resulting x and y coordinates for the element positions can be requested via `stimulus.positions.GetPositions()`. The position of an element is determined by the center of the element's bounding box (i.e., the rectangle 'bounding' the size of the element shape). The default position patterns used for Grid, Outline, and Concentric stimuli are a rectangular grid with a row and column spacing of 50 user units, a circle outline with radius 150, and identical (0,0) positions, respectively. The user can replace the default position pattern for the stimulus type by a custom one (e.g., a sinewave-shaped grid, a custom shape outline, random positions within a specified rectangle, manually specified positions). Figure 9 gives an overview of

[12] These options were chosen to allow the user to create a broad range of stimuli with only a limited number of specified stimulus types. The set of specified stimulus types is partially arbitrary and necessarily incomplete, and can be expanded in later generations of the toolbox based on user feedback or requests.

[13] The stimulus types Outline and Concentric are defined as Grid stimuli with the number of rows equal to one and the number of columns equal to the number of elements.

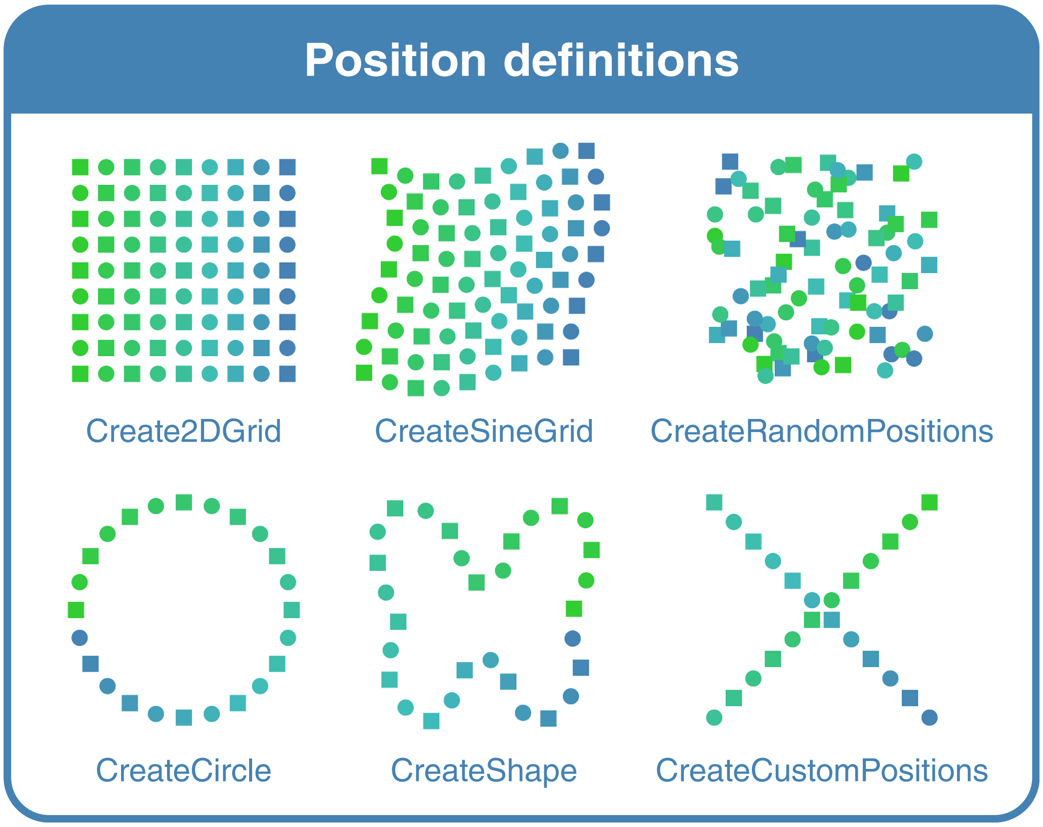


Fig. 9 Position definition options available in OCTA. Clicking a stimulus leads to the octa code used to generate it. Figure licensed under CC BY 4.0 by the authors. Retrieved from https://doi.org/10.6084/m9.figshare.17749139

all currently available position pattern functions.[14] Important to note is that any custom set of positions can be defined using the CreateCustomPositions function. For more information on all position pattern definitions, please consult the online documentation. In the Shiny app, the element positions can be specified under the tab '1. Add position pattern.'

Step 3. Specify patterns and pattern values for different element features

Once the stimulus features and element positions are specified, the pattern types, pattern directions, and pattern values for the different element features can be adjusted. The patterns can be applied to the following element features: shapes, boundingboxes, fillcolors, orientations, borderwidths, bordercolors, opacities, mirrorvalues, links, classlabels, and idlabels (see Element features and Fig. 12). In Grid and Outline stimuli, one value per element feature is repeated across all elements in the stimulus by default. In Concentric stimuli, a pattern with two fillcolors is repeated across all elements in the stimulus by default, and the boundingbox sizes follows a decreasing gradient across elements. Available *pattern types* (see Fig. 10)[15] include pattern repetition (Repeat), element repetition (ElementRepeat), mirror symmetry (Mirror), and a gradient from a start value to an end value (Gradient). In Grid stimuli, these patterns can be applied according to the following *pattern directions*: across elements, across rows, across columns, across the left diagonal, across the right diagonal, and across layers (see Fig. 11). In addition, more complex patterns can be constructed using the TiledGrid and TiledElementGrid options. Whereas a TiledGrid copies the feature values in a source grid a specified number of times in the row and column directions, a TiledElementGrid copies each element in the source grid a specified number of times in the row and column directions. Finally, a random application of the values for the element feature across the elements is possible too (RandomPattern). By default, the pattern values in the RandomPattern are repeated until the length is equal to the number of elements in the stimulus. Optionally, a list of frequencies can be provided to determine how many times each pattern value has to be present. In Outline and Concentric

[14] These options were chosen to allow the user to create a broad range of stimuli with only a limited number of specified position definitions. The set of specified position definition functions is partially arbitrary and necessarily incomplete, but can be expanded in later generations of the toolbox based on user feedback or requests.

[15] These options were chosen based on the currently available literature using different pattern types, e.g., Chipman and Mendelson (1979), and to allow the user to create a broad range of stimuli with only a limited number of specified pattern types. The set of specified pattern types is necessarily incomplete, but can be expanded in later generations of the toolbox based on user feedback or requests.

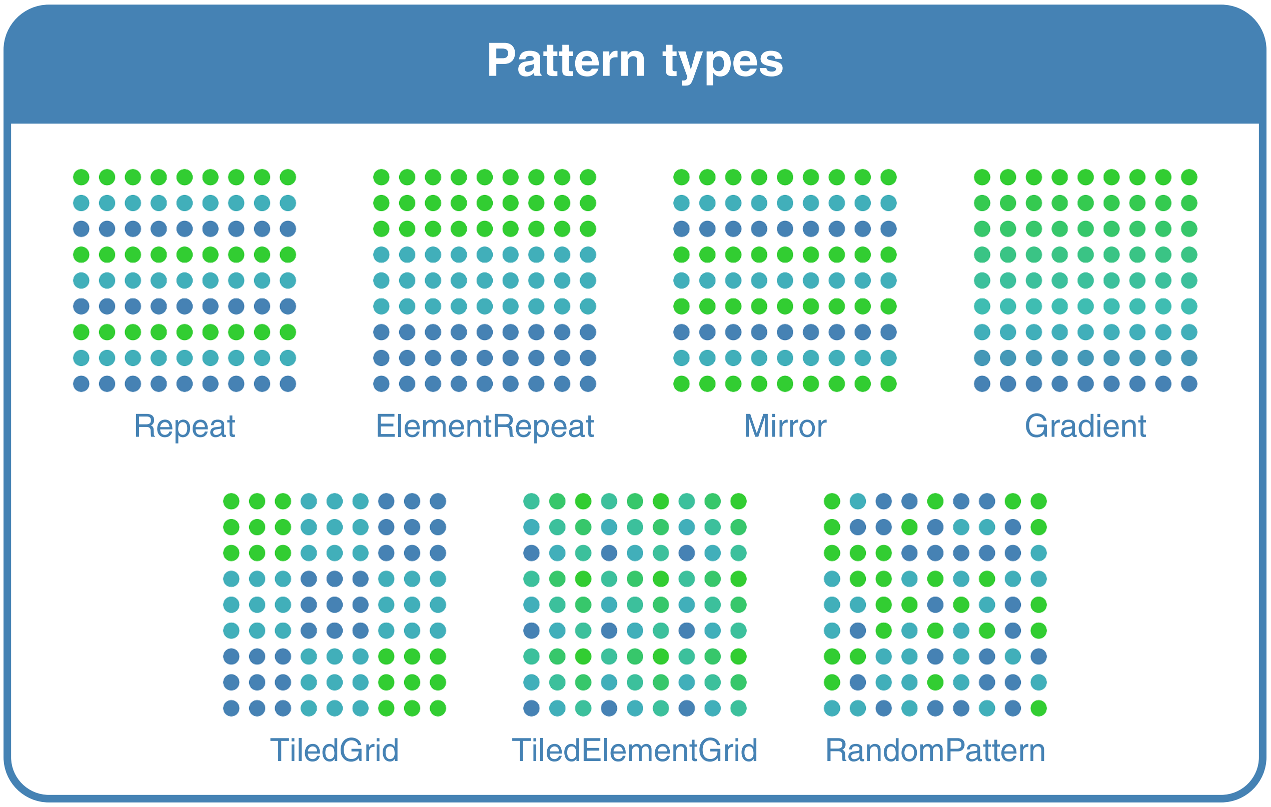


Fig. 10 Pattern types available in OCTA. Clicking a stimulus leads to the octa code used to generate it. Figure licensed under CC BY 4.0 by the authors. Retrieved from https://doi.org/10.6084/m9.figshare.17749340

stimuli, it is strongly advised to apply patterns across elements (as other pattern directions will not be distinguishable in Outline and Concentric stimuli). In the Shiny app, the feature patterns can be specified using the tabs under '2. Add feature patterns.' Some predefined pattern values are provided for each element feature, but custom pattern values can be specified too.

Element features (see Fig. 12)

Shapes Available element shape types are geometric shapes, i.e., Ellipse, Rectangle, Triangle, Polygon(n_sides), RegularPolygon(n_sides); more complex path elements[16] based on a provided path string or an existing SVG file, i.e., Path(path, xsize, ysize), PathSvg(source); images based on a provided filename or source url, i.e., Image(source), FitImage(source); and text elements,[17] i.e., Text(text). Keep in mind that for the order and complexity measurements, the shapes argument only takes the shape *type* into account and does not distinguish between different polygons, different paths, different images, or different texts.[18] Currently no built-in animation options are available for the shape feature, but it is possible to include dynamic image files (e.g., dynamic SVG or gif file) as shape values in the stimulus. Also in the Shiny app, it is possible to specify custom shapes to use in the stimulus, including Image and PathSvg shapes that are publicly accessible online. It is however not possible to use local files when working with the online Shiny app.

Boundingboxes Boundingboxes are always rectangular but do not have to be squared. Boundingbox values are defined in user units and can be provided as follows: (xsize, ysize). Although currently no built-in animation options are available for the boundingbox feature, it is possible to either generate separate stimuli and combine them in time afterwards, or to include dynamic image files (e.g., dynamic SVG or GIF file with shape changing in size) as shape values in the stimulus. Shapes will take their maximal size possible within the specified boundingbox, taking into account shape definitions (e.g., the

[16] When providing a path definition or SVG file, keep in mind that the path or file may contain margins, making the visible result smaller than the defined boundingbox size. Correctly fetching a path from an existing SVG file may fail in case the SVG file contains multiple paths, no paths, or paths are specified differently than expected (e.g., including margins).

[17] Text elements are an experimental feature of the OCTA toolbox. The sizing of the text elements is optimized for a single capital letter to fit in the boundingbox provided.

[18] For example, in an OCTA stimulus containing two different images as image elements, the number of shape *types* used will be one. To take into account the number of different images, polygons, paths, or texts used, one can add the 'data' argument to the element features used in the order and complexity measurements.

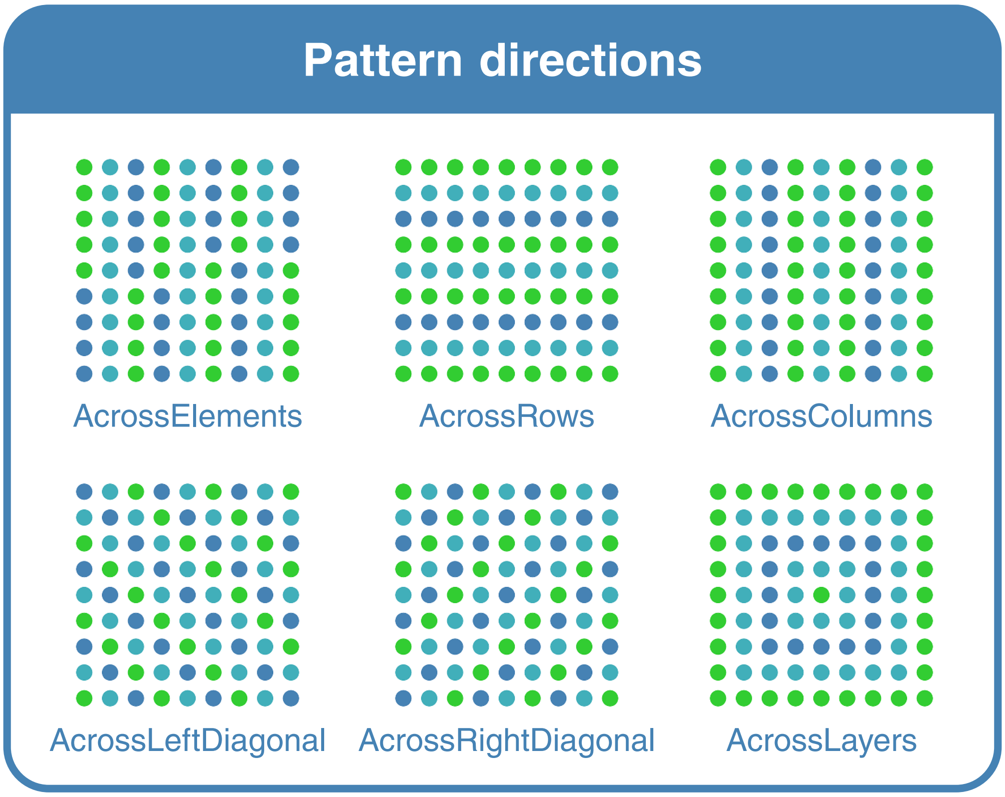


Fig. 11 Pattern directions available in OCTA. Clicking a stimulus leads to the octa code used to generate it. Figure licensed under CC BY 4.0 by the authors. Retrieved from https://doi.org/10.6084/m9.figshare.17749634

Image shape type will retain original aspect ratio of the image, whereas the FitImage shape type will fit the image to the boundingbox without taking original aspect ratio into account).[19]

Fillcolors Besides static uniform colors defined based on their hexadecimal color code or color name, the user can specify a radial or linear color gradient with multiple color values (i.e., radial, horizontal, vertical, diagonal) or dynamic fillcolors by setting a new color (i.e., 'set') when a certain action occurs (e.g., when the element is clicked) or by animating the color value (i.e., 'animate'). Fillcolors are not visible for image elements.

Orientations Orientations are defined in degrees. Besides static orientation values, dynamic definition of orientations is possible (i.e., 'set' or 'animate').

Borderwidths Borderwidths are defined in user units. Besides static borderwidth values, dynamic definition of borderwidths is possible (i.e., 'set' or 'animate'). Borderwidths are not taken into account when a shape is fit to a particular boundingbox value, which means that half of the borderwidth will fall outside of the specified shape and thus potentially outside of the specified boundingbox (as the borderwidth is centered around the shape border). Keep in mind that the default bordercolor is transparent, so bordercolor needs to be set for the borderwidth value to have a visible effect. In addition, borderwidths are not visible for image elements and may behave unexpectedly for path elements.[20]

Bordercolors Besides static uniform bordercolors defined based on their hexadecimal color code or color name, the user can specify a radial or linear color gradient with multiple bordercolor values (i.e., radial, horizontal, vertical, diagonal) or dynamic bordercolors by setting a new color (i.e., 'set') when a certain action occurs (e.g., when the element is clicked) or by animating the color value (i.e., 'animate'). Bordercolors are not visible for image elements. Keep in mind that the default borderwidth is

[19] As specified in footnote 12, when providing a path definition or SVG file, keep in mind that the path or file may contain margins, making the visible result smaller than the defined boundingbox size. Furthermore, the sizing of the text elements is optimized for a single capital letter to fit in the boundingbox provided.

[20] Related to footnote 12, when providing a path definition or an SVG file containing a path definition, the path definition may contain margins, making the visible result smaller than the defined boundingbox size. In addition, the path may be defined on another size than it is scaled to, and the borderwidth will be applied to the original size, making the visible borderwidth often smaller than intended. A correct path definition and a borderwidth value custom to the original path boundingbox is thus key to get a correct borderwidth for path elements.

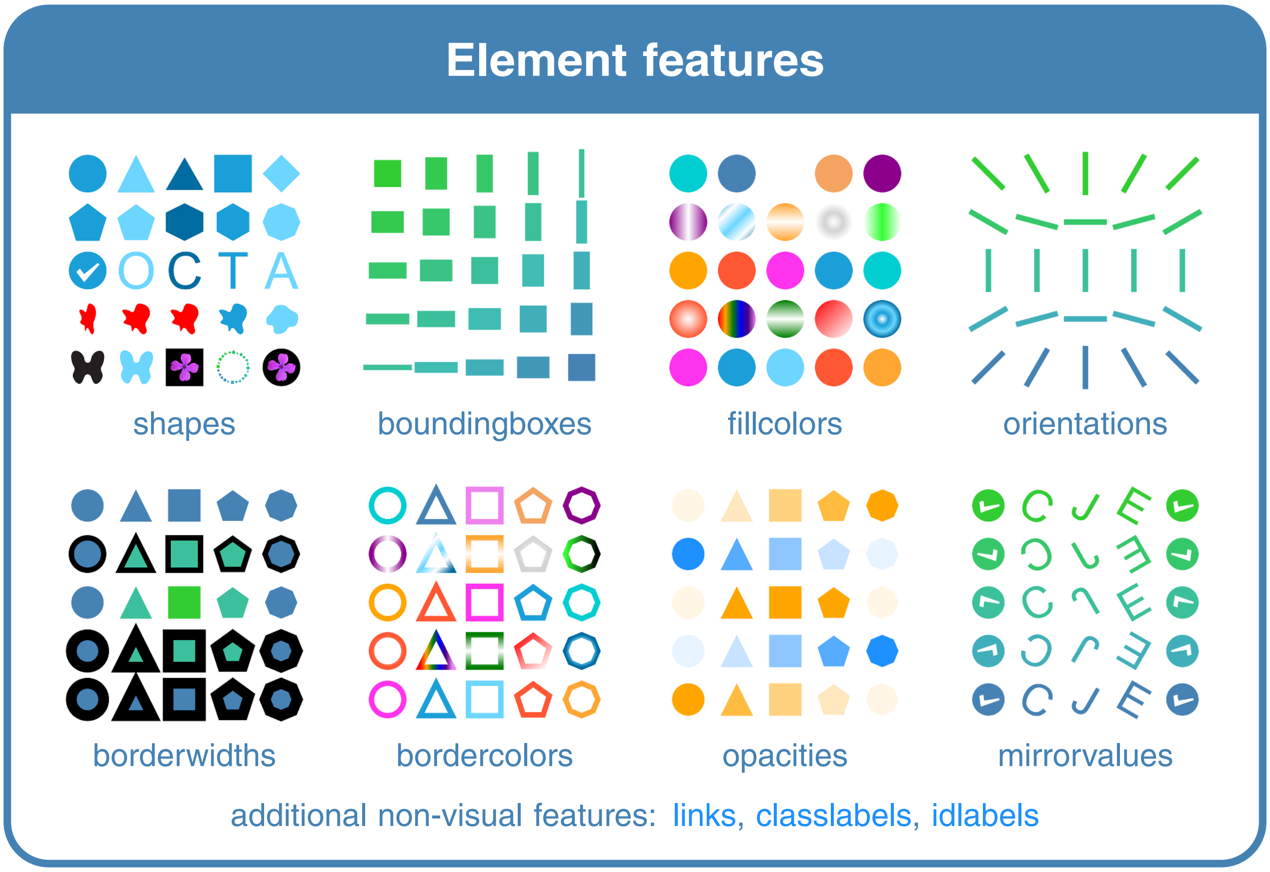


Fig. 12 Element features available in OCTA. Clicking a stimulus leads to the octa code used to generate it. To view dynamic and interactive versions of some of these stimuli, visit this webpage. Figure licensed under CC BY 4.0 by the authors. Retrieved from https://doi.org/10.6084/m9.figshare.17749229. The shapes example stimulus contains a butterfly image from the Auckland Optotypes (Hamm, Yeoman, Anstice, and Dakin, 2018) and a flower image from Hůla and Flegr (2016)

zero, so borderwidth needs to be set for the bordercolor values to have a visible effect.

Opacities Opacities are defined between 0 (no opacity) and 1 (full opacity). Besides static values, dynamic definition of opacities is possible (i.e., 'set' or 'animate').

Mirrorvalues As element shapes can be mirrored along the horizontal and/or vertical axis, or not be mirrored, this element feature can take any of four different values: 'none', 'horizontal', 'vertical', or 'horizontalvertical'. Currently no built-in animation options are available for the mirrorvalue feature, but it is possible to either generate separate stimuli and combine them in time afterwards, or to include dynamic image files (e.g., dynamic SVG or GIF file with shape changing in mirrorvalue) as shape values in the stimulus.

Links A custom hyperlink can be added to every element in the stimulus. This entails a hand cursor shown when the viewer hovers over that specific element, and the opening of a hyperlink when the element is clicked.

Classlabels and idlabels Although classlabels and idlabels do not have a directly visible effect on the resulting element, they can be used to add additional JavaScript actions or CSS style changes to individual elements (using the idlabel) or to a group of elements (i.e., all elements with the same classlabel). This method can, for example, be used to add sounds when hovering over an element or when an element is clicked.

Data Data is a hidden element feature not meant for direct user interaction. It stores the additional arguments given to any of the shapes (e.g., the number of sides for Polygon objects, the source argument for Image objects). This element feature may become important when calculating order and complexity measures within the OCTA toolbox, to distinguish between polygons with a different number of sides, different path definitions, different text elements, or images with different sources.

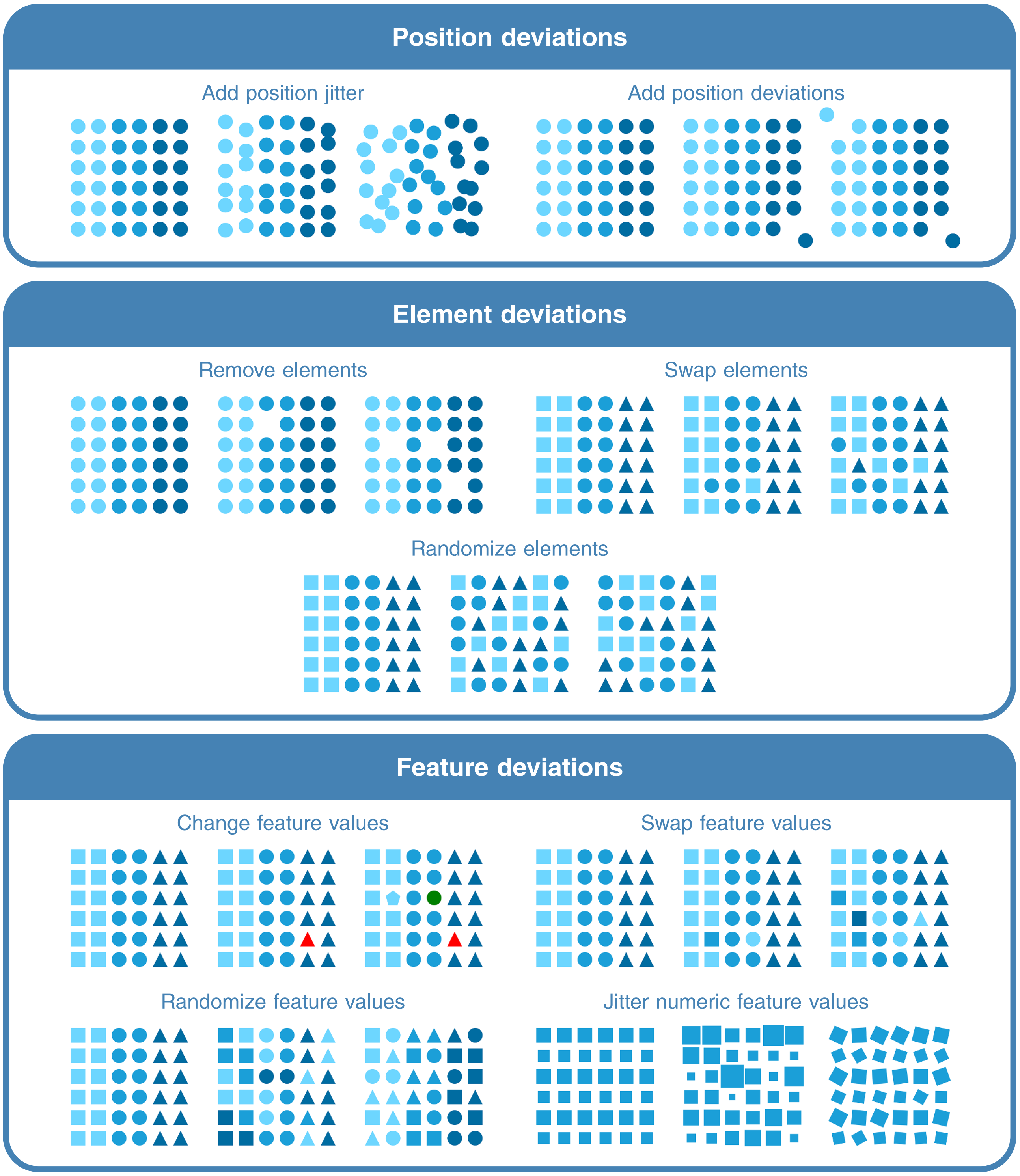


Fig. 13 Position, element, and feature deviation options in OCTA. Clicking a stimulus leads to the octa code used to generate it. Figure licensed under CC BY 4.0 by the authors. Retrieved from https://doi.org/10.6084/m9.figshare.17749763

Step 4. Add deviations and calculate measures

Once all element feature patterns are specified as desired, the user can add position, element, or feature deviations to decrease the order and/or increase the complexity of the stimulus. Figure 13 gives examples of the position, element, and feature deviations that can be added. Although most of these deviation types affect both objective order and objective complexity, some manipulations specifically target either order or complexity. Position, element, and feature deviations are saved separately from the original patterns used to create the stimulus. If multiple deviations are added, later deviations could overwrite earlier deviations if they concern the same position, element, or element feature. In the Shiny app, deviations can be added under the tab '3. Add deviations.' Within this tab, position and element deviations are bundled under 'Position deviations' and element feature deviations are displayed under 'Feature deviations.'

Position deviations Random jitter or specified deviations can be added to the element positions. Although the impact of position deviations is dependent on the specific deviations added as well as the position pattern used as starting point, they generally increase objective complexity in the positions used in the stimulus (i.e., they increase the variety of positions present in the stimulus). Element or feature complexity stays unchanged when deviating position only. Whether the position deviations influence objective order in positions, elements, or features depends on the specific deviation.[21]

For normally distributed or uniformly distributed ***position jitter***, one can specify whether the jitter needs to be applied to the x coordinates, y coordinates, both axes equally, or both axes independently using the axis argument. In the case of uniformly distributed jitter, the user specifies a minimum and a maximum value (min_val and max_val). In the case of normally distributed jitter, the user specifies mean (mu) and standard deviation (std). Default values for axis, distribution, min_val, max_val, mu, and std are 'xy', 'normal', -1, 1, 0, and 1, respectively. To add ***specific position deviations***, the user specifies the element ids (starting from 0 until n_elements - 1), x offsets and/or y offsets for each of the elements to which a deviation relative to the predetermined position needs to be added. Element id needs to be given an integer value or a list of integer values. X and y offsets can be numeric values or a list of numeric values.

Element deviations To add element deviations, it is possible to ***remove*** a set of random or specified ***elements*** from the display, to ***swap*** the positions of distinct ***elements***, or to ***randomize*** the order of all ***elements*** in a particular direction. Bringing an additional element into the pattern is possible as well, but requires the user to add an additional element in the stimulus definition and in the definition of the stimulus positions. Swapping or randomizing the position of distinct elements in the display decreases objective element order, but leaves objective position, element, or feature complexity unchanged. If also non-distinct elements would be swapped or randomized, the objective element order may stay unchanged. Removing elements does complicate the position pattern and potentially reduces element or feature complexity, but more generally also decreases element and feature order.

Feature deviations To add feature deviations in the stimulus, it is possible to ***change*** a ***feature value*** for a number of random or specified elements, to ***swap*** the ***feature values*** for a number of random pairs of (distinct) elements in the display, to ***randomize*** the order of all ***feature values*** in a particular direction, or to ***jitter*** any of the ***numeric feature values*** across all elements. Swapping or randomizing the position of distinct feature values in the display decreases objective order for the feature dimensions involved, but also increases objective element complexity. Changing a feature value to a value that is not yet in the pattern values for that feature or jittering numeric feature values will additionally increase objective feature complexity. An advantage of adding feature deviations is that order can be distorted on one feature dimension specifically but preserved for other feature dimensions (contrary to what is the case with element deviations).

Order and complexity measures and manipulations Although deviations are one way to increase or decrease different types of order and complexity in the stimulus, other approaches are possible too. Figures 14 and 15 give an overview of some order and complexity manipulations that are possible in OCTA. Figure 16 lists the order and complexity measures available in OCTA.

Manipulating order Position order can be changed qualitatively by changing the type of position pattern. Element (and feature) order can be changed qualitatively by changing the different element feature pattern types and directions.[22] To induce quantitative changes in order, the user can swap the positions of distinct elements or randomize the elements in the stimulus (which keep element complexity level constant), or add any other element or feature deviations (but these other deviations may influence element or feature complexity as well; cf. Step 4. Add deviations and calculate measures). The user can also make stimulus features and element features (in)congruent to impact the order level of the stimulus. In addition, the congruency of patterns, pattern types, or pattern directions across feature dimensions can be adapted.[23]

Manipulating complexity Qualitative changes in complexity can be achieved by changing the feature dimension on which

[21] One may argue that position deviations decrease the objective order in the positions, but this may not always be the case. For example, when adding symmetric position deviations, these positions deviations may only alter—or in some cases even increase—the structure and organization of the positions present in the stimulus.

[22] Be aware that some feature pattern changes may also impact element complexity because of emerging (non-)congruency between different feature patterns.

[23] Keep in mind that changes in pattern congruency may simultaneously influence element complexity.

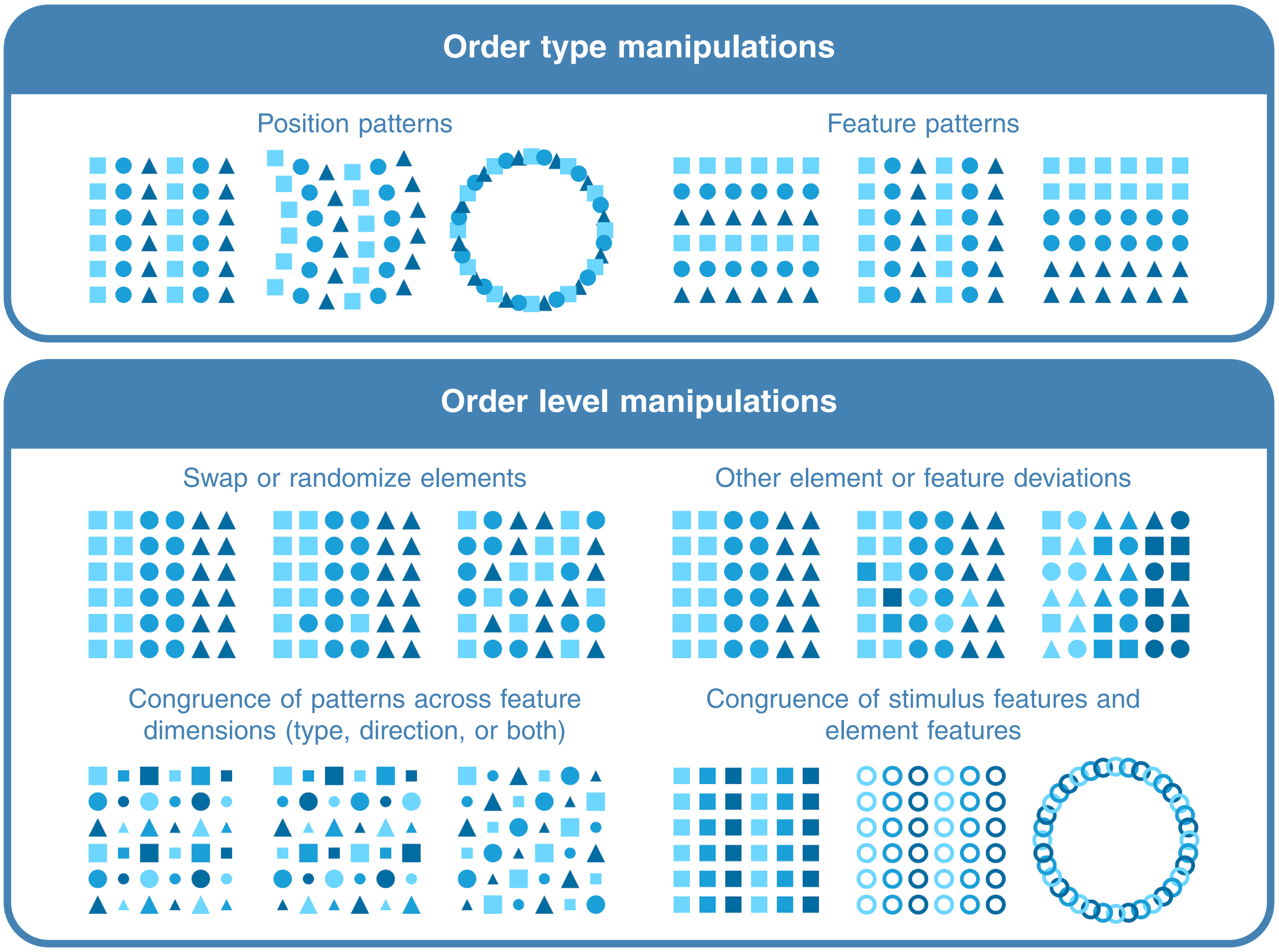

Order type manipulations
Position patterns
Feature patterns
Order level manipulations
Swap or randomize elements
Other element or feature deviations
Congruence of patterns across feature dimensions (type, direction, or both)
Congruence of stimulus features and element features

Fig. 14 Order manipulations in OCTA. Clicking a stimulus leads to the octa code used to generate it. Figure licensed under CC BY 4.0 by the authors. Retrieved from https://doi.org/10.6084/m9.figshare.17749853

the complexity is present (e.g., shape, color, or size complexity). Quantitative element complexity changes can include (a) changing the number of visible elements present in the stimulus (i.e., by removing elements or by changing the position pattern of the stimulus), (b) manipulating the variety of elements (i.e., by including more pattern values on a feature dimension, by choosing more diverse pattern values, by adding feature deviations, or by changing the congruency of patterns across feature dimensions), (c) changing the complexity (familiarity, unintelligibility, etc.) of individual feature pattern values (e.g., use complex path shape instead of rectangles), or (d) changing the complexity of individual stimulus features. Position complexity can be changed quantitatively by adding random position jitter or structured position deviations (cf. Position deviations).

Measuring order and complexity OCTA provides some basic functionality to measure aspects of order and complexity in the created stimulus. When it comes to complexity measures, it is possible to calculate (a) the ***number of elements*** present in the display (Number or N), (b) ***how many different types of elements*** are present in the display based on the feature dimensions specified (Level of Complexity based on Elements or LOCE), (c) ***how many different features*** are present across all feature dimensions (Level of Complexity or LOC), and (d) ***how many different feature dimensions*** have ***more than one feature value*** (i.e., have non-identical values; Level of Complexity based on Identity or LOCI). For order, the user can ***request*** the applied ***patterns***, pattern types, and pattern directions across all feature dimensions; ***check*** whether all specified feature dimensions have ***congruent patterns***, pattern types, or pattern directions; ***calculate how many*** specified feature dimensions have ***congruent patterns***, pattern types, or pattern directions; calculate the ***number of deviant elements*** that are present given the specified feature dimensions (e.g., by added element or feature deviations); and calculate the ***number of deviant positions*** that are present in the stimulus (i.e., by added position jitter or specified position deviations). In the Shiny app, order and complexity measures can be calculated under the tab ‘4. Calculate measures.’ Also in the Shiny app the user has the option to specify which element features to take into account when calculating the measures.

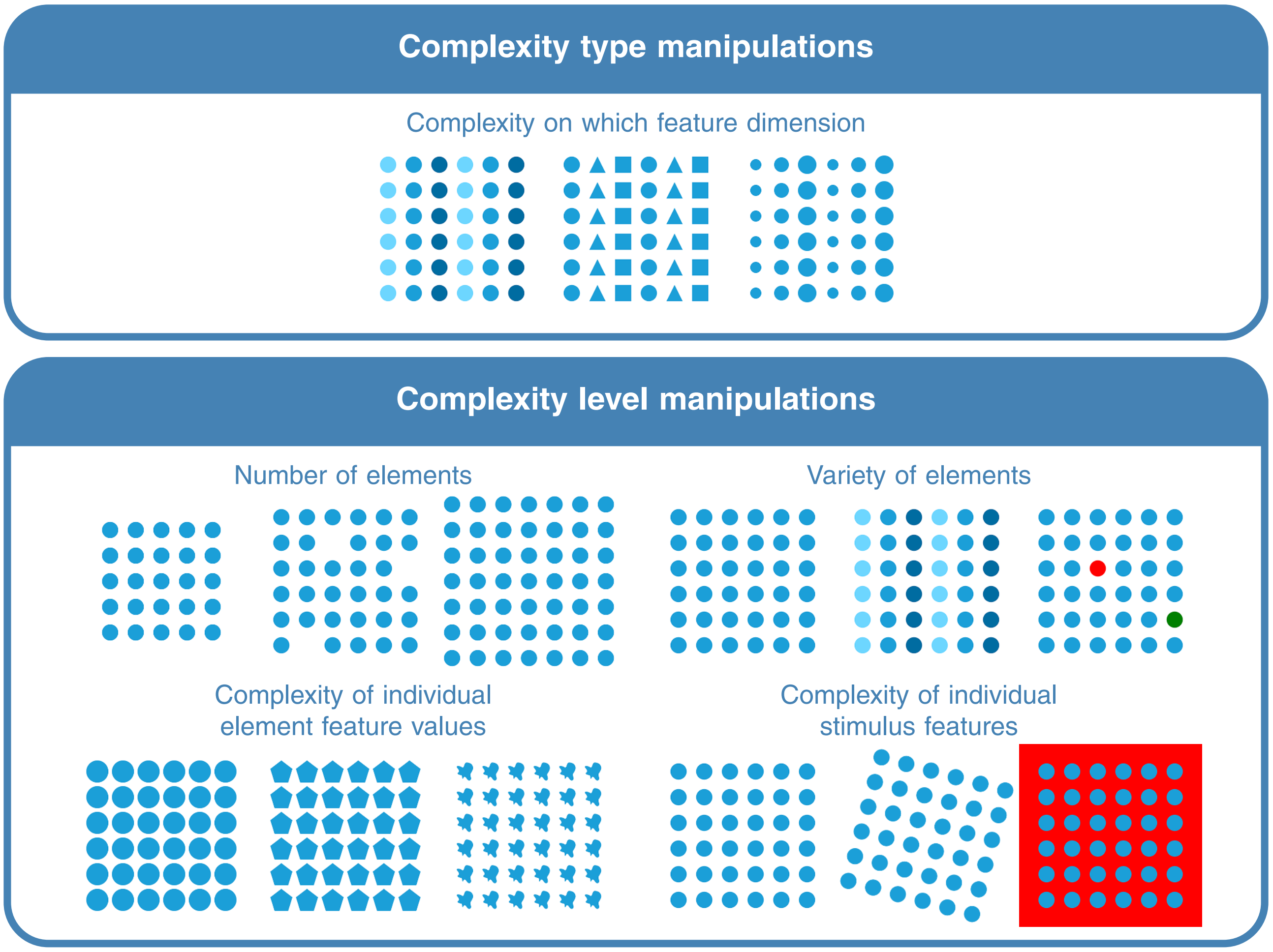

Complexity type manipulations
Complexity on which feature dimension
Complexity level manipulations
Number of elements
Variety of elements
Complexity of individual element feature values
Complexity of individual stimulus features

Fig. 15 Complexity manipulations in OCTA. Clicking a stimulus leads to the octa code used to generate it. Figure licensed under CC BY 4.0 by the authors. Retrieved from https://doi.org/10.6084/m9.figshare.17749958

Step 5. Generate the stimulus, show it on screen, and save to the desired output format

Finally, the user can save the resulting stimulus in the preferred format: as a vector-based image (SVG), as a raster-based image (PNG, JPG, PDF, or TIFF), or as a computer-readable file (JSON). The SVG format is recommended for online use, as it has the same quality at all viewing sizes and keeps the possibility for animated element and stimulus features. For raster-based image output, a scale value can be added to increase the quality when starting from a very small stimulus that would otherwise have pixel artefacts.[24] The JSON output can be used to recreate the stimulus in Python using the octa package without the original code (using the `LoadFromJSON` function). In the Shiny app, the user can download the stimulus as a vector-based SVG image or as a raster-based PNG or PDF image, the JSON output, or the Python code needed to reproduce the current stimulus with the octa package in Python. Besides, the Shiny app user can view the Python code (without the octa import statements), the raw SVG code, and the JSON code in the app itself.

[24] It is also possible to add this scale value to the SaveSVG function, to scale the vector-based image to the preferred size.

Discussion and conclusion

With the OCTA toolbox, it is possible to study order and complexity in combination. It acknowledges the multidimensionality of order and complexity and provides the tools to manipulate and measure order and complexity in several ways and on different feature dimensions.[25] The possibility to add more complex

[25] It is a conscious choice to not provide a full set of commonly used order and complexity measures within OCTA and only keep it to order and complexity measures directly deriving from the stimulus construction procedure. Calculating additional complexity measures can be useful, for which other tools already exist, e.g., imagefluency, Mayer (2021), described in Mayer and Landwehr (2018b) and Mayer and Landwehr (2018a); image spectral slope, fractal dimension and Shannon entropy as described in Mather (2018) and Mather (2020); PHOG measures described in Braun et al. (2013); and edge-orientation entropy as described in Redies, Brachmann, and Wagemans (2017). The user can use the vector- or raster-based image output to calculate these additional measures.

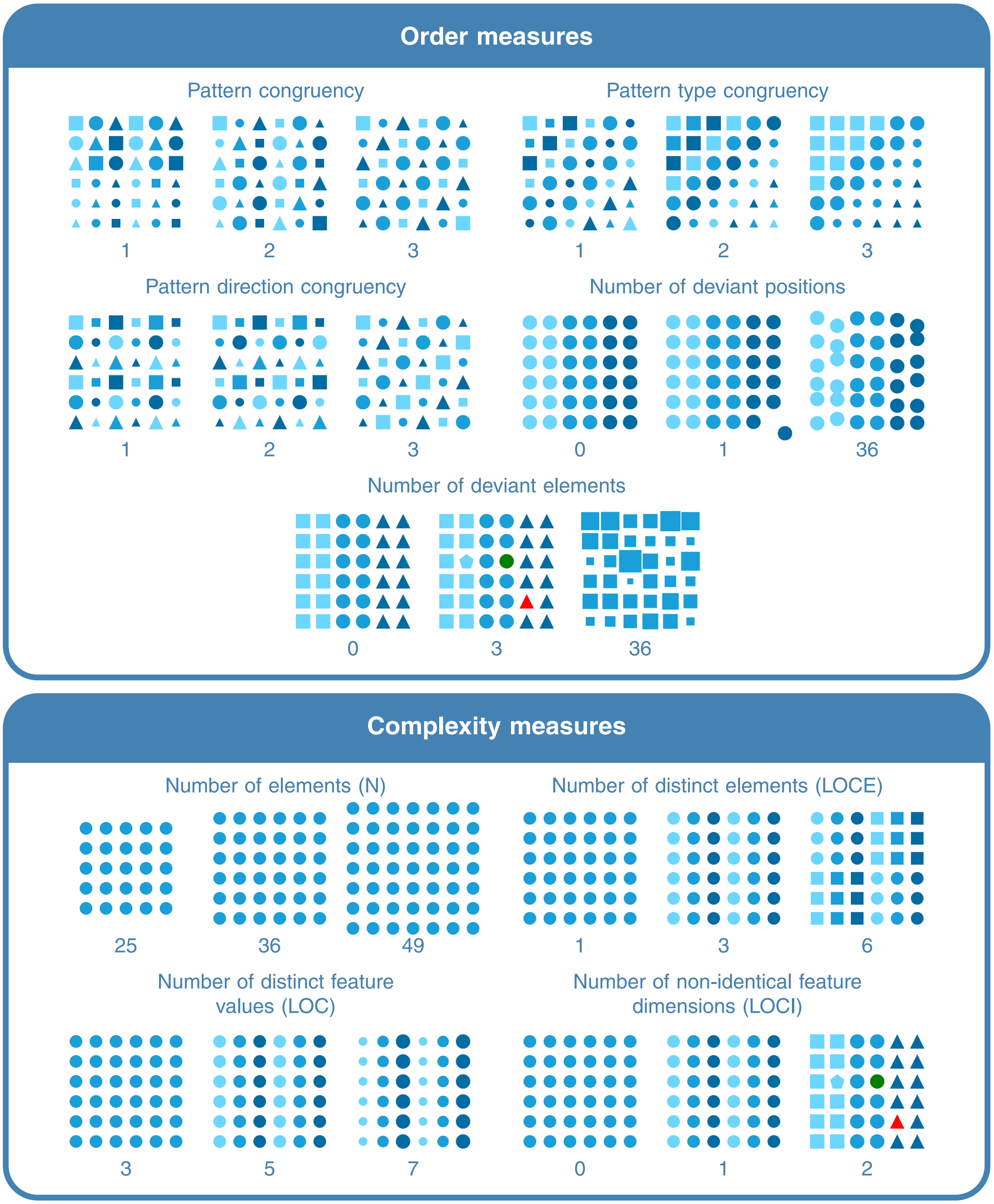

Order measures
Pattern congruency
1
2
3
Pattern type congruency
1
2
3
Pattern direction congruency
1
2
3
Number of deviant positions
0
1
36
Number of deviant elements
0
3
36
Complexity measures
Number of elements (N)
25
36
49
Number of distinct elements (LOCE)
1
3
6
Number of distinct feature values (LOC)
3
5
7
Number of non-identical feature dimensions (LOCI)
0
1
2

Fig. 16 Order and complexity measures in OCTA. Examples include boundingboxes, fillcolors, and shapes as (distinction) features. Clicking a stimulus leads to the octa code used to generate it. Figure licensed under CC BY 4.0 by the authors. Retrieved from https://doi.org/10.6084/m9.figshare.17750069

shapes and images in the stimulus enables the user to find their own balance between ecological validity and experimental control. As online testing becomes ever more common, vector-based OCTA stimuli are ideal for online use and allow room for dynamical features. The OCTA toolbox however is not constrained to vector-based output as it still allows for saving the stimuli as raster-based image formats. Importantly, OCTA provides a reproducible way of creating stimuli, as all OCTA code is open source, and tools are made available to reproduce the stimuli, either based on the original seed used and the original code or with the original seed used and a JSON file. Furthermore, as both a Python package and an online app are provided and a multitude of documentation and example stimuli are provided, using OCTA is possible for both researchers with and without programming experience, making it a widely accessible tool. Below we discuss some more advanced uses of OCTA, potential applications, and give advice on how to start using OCTA.

Using animated stimulus and element feature values

One of the big advantages of the OCTA toolbox being vector-based, is the option to animate a diverse set of stimulus and element features either directly within the OCTA toolbox (i.e., using SVG animation to animate stimulus orientation and element fillcolors, orientations, borderwidths, bordercolors, and opacities) or after the OCTA stimulus has been created (i.e., using some additional CSS or JavaScript code, making use of the class and id labels that can be added to specific elements within OCTA stimuli and to OCTA stimuli as a whole).

Within the OCTA toolbox, a wide range of animation options is available: for example, animations initiated by clicking or by a specified starting time, animations including discrete steps or continuous change in feature values, one-time or indefinitely repeating animations. For use of animated feature values in OCTA, have a look at some dynamic stimulus examples or try out some of the animated example feature values in the OCTA Shiny app. For a more detailed reference on animation options in the SVG language, consult the SVG documentation on animation by The World Wide Web Consortium or Mozilla, or the documentation of the svgwrite Python package that is used within the octa Python package.

For more information on how to add CSS or JavaScript animations to SVG images based on class or id labels once the OCTA stimulus has been generated, consult the general documentation on CSS or JavaScript animations by W3Schools or search the internet for more specific tutorials on SVG animation using CSS or JavaScript. For a very simple demo using OCTA stimuli, consult the part on adding stimulus class and id labels in the OCTA documentation (or go directly to the example).

Creating sets of stimuli

Depending on the research question at hand, one can create differently controlled stimulus sets using OCTA. For example, a researcher interested in investigating the influence of the level of order on perception and appreciation of an image under different levels of complexity may create stimuli varying in order level but keeping order type, complexity level and complexity type constant at a smaller number of values. A researcher interested in investigating the generalizability of particular findings concerning order, complexity, and aesthetic appreciation may create stimuli keeping order and complexity levels and types constant but varying the pattern values used in each of the feature dimensions (e.g., fillcolors, boundingboxes, shapes). Furthermore, a researcher interested in studying to what extent and in which way order and complexity on different feature dimensions influence the perception and appreciation of an image, may create stimuli varying the type of complexity present (e.g., number and variety of fillcolors, boundingboxes, or shapes) and keeping order type, order level, and complexity level constant.[26] Researchers investigating perceptual grouping principles including proximity and different types of similarity may use the OCTA toolbox to create congruent and incongruent stimuli, vary row and column spacing in Grid stimuli, or vary the absolute feature values used to investigate the generalizability of the grouping strengths beyond typically used feature values (e.g., black-colored circular elements in the commonly used dot lattices). Moreover, researchers concerned with ecological validity of earlier findings may create equivalent stimulus sets with more abstract and more concrete shapes relevant to daily life (e.g., comparing a standard circle and a circle looking like a button, or comparing a standard triangle and a triangle looking like a tent). These are only examples of stimulus sets that could be created, as the OCTA toolbox gives researchers a very elaborate range of options that can be combined in any way preferred.

When creating sets of stimuli, it may be advisable to use the octa Python package rather than the app, as using Python directly will give you more opportunities to use loops and create multiple stimuli at once. This does not hold users back to first create one of the stimuli in the Shiny app, and then copy the code to Python for creating the complete set of stimuli. For researchers more familiar with R than with Python, the reticulate package (Allaire, Ushey, Tang, & Eddelbuettel, 2017) can be very useful to interact with the octa Python code when creating and saving the combinations of parameter values for the stimuli in R rather than in Python directly.

[26] One prerequisite for studying the multidimensionality of order and complexity may be finding equivalent levels of change on different feature dimensions.

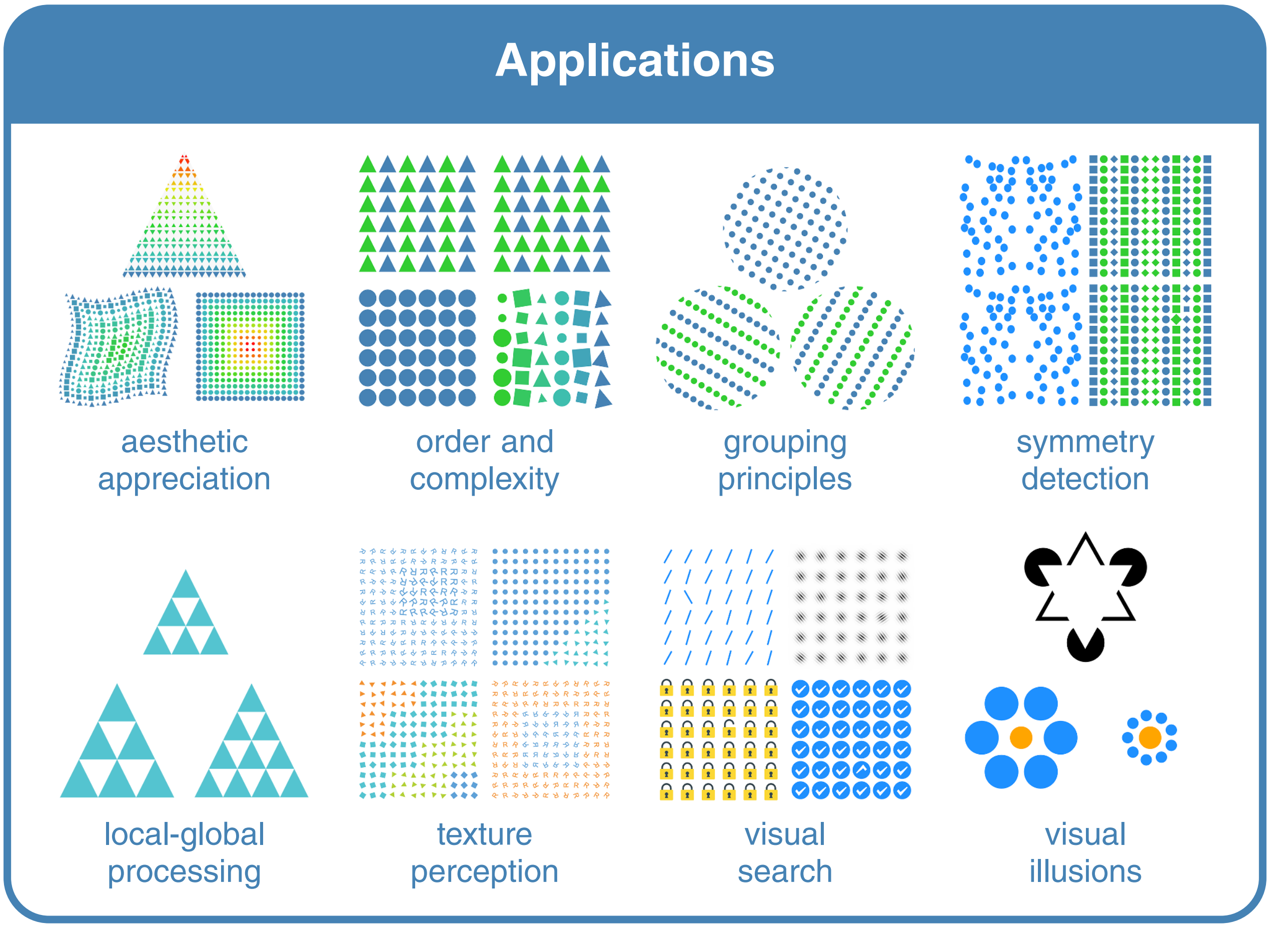


Fig. 17 Applications of OCTA. Clicking a stimulus leads to the octa code used to generate it. Figure licensed under CC BY 4.0 by the authors. Retrieved from https://doi.org/10.6084/m9.figshare.17750123. The OCTA stimuli related to local-global processing are based on stimuli used in Kimchi and Palmer (1982, Exp. 1). The OCTA stimuli related to texture perception are inspired by Julesz (1981). The OCTA stimuli related to visual illusions represent a Kanizsa triangle and the Ebbinghaus illusion

Applications

Although OCTA has originally been created to study order and complexity in the context of empirical aesthetics, the toolbox can be used for generation of static or dynamic stimuli in a much broader field of research using visual stimuli as well as in non-academic use contexts. Examples of some research fields that could benefit from the use of OCTA are perceptual organization (e.g., grouping principles), symmetry detection, local-global processing and part-whole relationships, texture perception, visual search, visual illusions, perceptual averaging, ensemble perception, other types of visual perception research (for a review of many of these topics, see Wagemans, 2018), creativity, joy of ordering, aesthetic appreciation in general, and many more (see Fig. 17). For example, although OCTA is not built with a focus on different types of symmetry specifically, all four basic two-dimensional symmetry operations (rotation, translation, reflection, and glide reflection) are at least in some form automatically available in OCTA,[27] and the seven frieze patterns as well as the 17 wallpaper patterns can be created using OCTA.[28] In sum, OCTA can be a useful tool in any type of research using visual stimuli, and even to generate digital art (for some more examples, see Fig. 1). In addition, OCTA stimuli can be used in many different kinds of tasks (rating, pairwise comparison, ranking, sorting,

[27] Rotation can be achieved using the orientation feature dimension, translation in horizontal and vertical directions is possible using the col_spacing and row_spacing parameters in the Grid stimulus type and translation in any direction is possible using custom position definitions; reflection can be achieved using the mirrorvalue feature dimension; and glide reflection can be achieved using a combination of translation and reflection manipulations.

[28] As OCTA currently builds mostly from rectangular grids, generating wallpaper patterns starting from these requires less customization (e.g., custom position definitions) than creating wallpaper patterns starting from other grid types. One potential avenue for further development of the OCTA toolbox is adding additional grids and related pattern types, to make the creation of some of these wallpaper patterns more straightforward in OCTA.

detection, construction, adjustment, etc., for a review of many of these tasks, see Palmer, Schloss, and Sammartino, 2013). For some experiment demos in jsPsych (de Leeuw, 2015), consult the linked webpages (demos, code). Preliminary versions of OCTA are currently already being used both in research on the perception and appreciation of order and complexity (including order and complexity variations in shapes, boundingboxes, fillcolors, and number of elements, Van Geert, Hofmann, & Wagemans, in preparation; Van Geert, Warny, & Wagemans, in preparation) and in research on the proximity principle in perceptual organization (including manipulations of row spacing and column spacing, Van der Hulst, Van Geert, & Wagemans, in preparation).

Advice on starting to use OCTA

Although learning to work with a new tool can lead to some cold feet, the online application as well as all the additional resources (e.g., manual, example stimuli, detailed function documentation; for an overview, visit elinevg.github.io/OCTA/) help users along the way. As OCTA is a new tool, feedback is welcomed and there are many opportunities for further development (e.g., additional position patterns and deviation options, additional animation options, option to create multiple stimuli at once). We do believe however that the current functionality already provides an immense array of options unexplored in aesthetics and visual perception research so far, and invite readers to explore the options[29] in the app (for starting users) or the Python package (for researchers with some prior Python programming experience; on the Python Package Index or on GitHub).[30]

Open practices statement

The octa Python package is available (on the Python Package Index or on GitHub) as open-source software under the GNU Lesser General Public License (Version 3) as published by the Free Software Foundation. The following webpage collects all additional resources concerning the OCTA toolbox: https://elinevg.github.io/OCTA/, including the Shiny app, the OCTA manual, the octa function documentation, example stimuli created in OCTA, demo experiments using OCTA stimuli, and more. When using either the octa Python package or the OCTA Shiny app in your (academic) work, please cite this paper to acknowledge the authors.

[29] With some additional JavaScript, it is even possible to add sounds to specific elements or stimuli as a whole and thus to create multimodal stimuli (cf. class labels and id labels in the online documentation).

[30] To start working with the octa Python package, it can be helpful to copy Python code from the Shiny app, the manual or the example stimuli in Python and start from there.

Funding This work has been supported by a PhD fellowship from the Research Foundation – Flanders (FWO) awarded to Eline Van Geert (Grant 11D3619N) and by long-term structural funding from the Flemish Government awarded to Johan Wagemans (METH/14/02 and METH/21/02).

References

Aguilar, D. (2021). Jsonpickle (Version 2.0.0). Retrieved from https://github.com/jsonpickle/jsonpickle

Allaire, J., Ushey, K., Tang, Y., & Eddelbuettel, D. (2017). Reticulate: R interface to Python. Retrieved from https://github.com/rstudio/reticulate

Alp, N., Kohler, P.J., Kogo, N., Wagemans, J., & Norcia, A.M. (2018). Measuring integration processes in visual symmetry with frequency-tagged EEG. *Scientific Reports*, *8*(1), 6969. https://doi.org/10.1038/s41598-018-24513-w

Alvarez, L., Gousseau, Y., Morel, J.-M., & Salgado, A. (2015). Exploring the space of abstract textures by principles and random sampling. *Journal of Mathematical Imaging and Vision*, *53*(3), 332–345. https://doi.org/10.1007/s10851-015-0582-z

Alvarez, L., Monzón, N., & Morel, J.-M. (2021). Interactive design of random aesthetic abstract textures by composition principles. *Leonardo*, *54*(2), 179–184. https://doi.org/10.1162/leon_a_01768

Arnheim, R. (1971). *Entropy and art: An essay on disorder and order*. University of California Press.

Arnoult, M.D. (1960). Prediction of perceptual responses from structural characteristics of the stimulus. *Perceptual and Motor Skills*, *11*(3), 261–268. https://doi.org/10.2466/pms.1960.11.3.261

Attneave, F. (1957). Physical determinants of the judged complexity of shapes. *Journal of Experimental Psychology*, *53*(4), 221–227. https://doi.org/10.1037/h0043921

Attneave, F., & Arnoult, M.D. (1956). The quantitative study of shape and pattern perception. *Psychological Bulletin*, *53*(6), 452–471. https://doi.org/10.1037/h0044049

Berlyne, D.E. (Ed.) (1960). *Conflict, arousal and curiosity*. New York, NY: McGraw-Hill. https://doi.org/10.1037/11164-000

Berlyne, D.E. (Ed.) (1974). *Studies in the new experimental aesthetics: Steps toward an objective psychology of aesthetic appreciation*. Oxford, England: Hemisphere.

Bertamini, M., & Rampone, G. (2020). The study of symmetry in empirical aesthetics. In M. Nadal, & O Vartanian (Eds.) *The Oxford handbook of empirical aesthetics*. Oxford University Press. https://doi.org/10.1093/oxfordhb/9780198824350.013.23

Bies, A.J., Blanc-Goldhammer, D.R., Boydston, C.R., Taylor, R.P., & Sereno, M.E. (2016). Aesthetic responses to exact fractals driven by physical complexity. *Frontiers in Human Neuroscience, 10*. https://doi.org/10.3389/fnhum.2016.00210

Braun, J., Amirshahi, S.A., Denzler, J., & Redies, C. (2013). Statistical image properties of print advertisements, visual artworks and images of architecture. *Frontiers in Psychology, 4*. https://doi.org/10.3389/fpsyg.2013.00808

Chipman, S.F. (1977). Complexity and structure in visual patterns. *Journal of Experimental Psychology: General*, *106*(3), 296–301.

Chipman, S.F., & Mendelson, M.J. (1979). Influence of six types of visual structure on complexity judgments in children and adults. *Journal of Experimental Psychology: Human Perception and Performance*, *5*(2), 365–378.

Clarke, A.D.F., Green, P.R., Halley, F., & Chantler, M.J. (2011). Similar symmetries: The role of wallpaper groups in perceptual texture similarity. *Symmetry*, *3*(2), 246–264. https://doi.org/10.3390/sym3020246

Cupchik, G.C., & Berlyne, D.E. (1979). The perception of collative properties in visual stimuli. *Scandinavian Journal of Psychology*, *20*(1), 93–104. https://doi.org/10.1111/j.1467-9450.1979.tb00688.x

de Leeuw, J.R. (2015). jsPsych: A JavaScript library for creating behavioral experiments in a web browser. *Behavior Research Methods*, *47*(1), 1–12. https://doi.org/10.3758/s13428-014-0458-y

Donderi, D.C. (2006). Visual complexity: A review. *Psychological Bulletin*, *132*(1), 73–97. https://doi.org/10.1037/0033-2909.132.1.73

Garner, W.R., & Clement, D.E. (1963). Goodness of pattern and pattern uncertainty. *Journal of Verbal Learning and Verbal Behavior*, *2*(5-6), 446–452. https://doi.org/10.1016/S0022-5371(63)80046-8

Gartus, A., & Leder, H. (2013). The small step toward asymmetry: Aesthetic judgment of broken symmetries. *I-Perception*, *4*(5), 361–364. https://doi.org/10.1068/i0588sas

Gherman, D. (2021). Svglib (Version 1.1.0). Retrieved from https://github.com/deeplook/svglib

Gollwitzer, A., Marshall, J., Wang, Y., & Bargh, J.A. (2017). Relating pattern deviancy aversion to stigma and prejudice. *Nature Human Behaviour*, *1*(12), 920–927. https://doi.org/10.1038/s41562-017-0243-x

Grebenkina, M., Brachmann, A., Bertamini, M., Kaduhm, A., & Redies, C. (2018). Edge-orientation entropy predicts preference for diverse types of man-made images. *Frontiers in Neuroscience, 12*. https://doi.org/10.3389/fnins.2018.00678

Grünbaum, B., & Shephard, G.C. (1989) *Tilings and patterns*. New York: W. H. Freeman and Company.

Güçlütürk, Y., Jacobs, R. H. A. H., & van Lier, R. (2016). Liking versus complexity: Decomposing the inverted U-curve. *Frontiers in Human Neuroscience, 10*. https://doi.org/10.3389/fnhum.2016.00112

Hamada, J., & Ishihara, T. (1988). Complexity and goodness of dot patterns varying in symmetry. *Psychological Research*, *50*(3), 155–161. https://doi.org/10.1007/BF00310176

Hamm, L.M., Yeoman, J.P., Anstice, N., & Dakin, S.C. (2018). The Auckland Optotypes: An open-access pictogram set for measuring recognition acuity. *Journal of Vision*, *18*(3), 13–13. https://doi.org/10.1167/18.3.13

Hübner, R., & Fillinger, M.G. (2016). Comparison of objective measures for predicting perceptual balance and visual aesthetic preference. *Frontiers in Psychology, 7*. https://doi.org/10.3389/fpsyg.2016.00335

Hůla, M., & Flegr, J. (2016). What flowers do we like? The influence of shape and color on the rating of flower beauty. *PeerJ*, *4*, e2106. https://doi.org/10.7717/peerj.2106

Jacobsen, T., & Höfel, L. (2002). Aesthetic judgments of novel graphic patterns: Analyses of individual judgments. *Perceptual and Motor Skills*, *95*(3), 755–766. https://doi.org/10.2466/pms.2002.95.3.755

Julesz, B. (1981). Textons, the elements of texture perception, and their interactions. *Nature*, *290*(5802), 91–97. https://doi.org/10.1038/290091a0

Kimchi, R., & Palmer, S.E. (1982). Form and texture in hierarchically constructed patterns. *Journal of Experimental Psychology: Human Perception and Performance*, *8*(4), 521–535. https://doi.org/10.1037/0096-1523.8.4.521

Kohler, P.J., Clarke, A., Yakovleva, A., Liu, Y., & Norcia, A.M. (2016). Representation of maximally regular textures in human visual cortex. *The Journal of Neuroscience*, *36*(3), 714–729. https://doi.org/10.1523/JNEUROSCI.2962-15.2016

Krakowski, C.-S., Poirel, N., Vidal, J., Roëll, M., Pineau, A., Borst, G., & Houdé, O. (2016). The forest, the trees, and the leaves: Differences of processing across development. *Developmental Psychology*, *52*(8), 1262–1272. https://doi.org/10.1037/dev0000138

Lab, V. (2021). Colour (Version 0.1.5). Retrieved from http://github.com/vaab/colour

Locher, P.J., Stappers, P.J., & Overbeeke, K. (1998). The role of balance as an organizing design principle underlying adults' compositional strategies for creating visual displays. *Acta Psychologica*, *99*(2), 141–161. https://doi.org/10.1016/S0001-6918(98)00008-0

Martin, P., Uy, N., Kvapil, M, & Friedenberg, J. (2020). The aesthetics of frieze of patterns: A preference for emergent features [Poster retrieved from https://doi.org/10.13140/RG.2.2.34413.74721]

Mather, G. (2018). Visual image statistics in the history of Western art. *Art and Perception*, *6* (2-3), 97–115. https://doi.org/10.1163/22134913-20181092

Mather, G. (2020). Aesthetic image statistics vary with artistic genre. *Vision*, *4*(1), 10. https://doi.org/10.3390/vision4010010

Mayer, S. (2021). Imagefluency: Image statistics based on processing fluency. Zenodo. https://doi.org/10.5281/zenodo.5614666

Mayer, S., & Landwehr, J.R. (2018a). Objective measures of design typicality. *Design Studies*, *54*, 146–161. https://doi.org/10.1016/j.destud.2017.09.004

Mayer, S., & Landwehr, J.R. (2018b). Quantifying visual aesthetics based on processing fluency theory: Four algorithmic measures for antecedents of aesthetic preferences. *Psychology of Aesthetics, Creativity, and the Arts*, *12*(4), 399–431. https://doi.org/10.1037/aca0000187

Moitzi, M. (2021). Svgwrite (Version 1.4.1). Retrieved from http://github.com/mozman/svgwrite.git

Muth, C., Westphal-Fitch, G., & Carbon, C.-C. (2019). Seeking (dis)order: Ordering appeals but slight disorder and complex order trigger interest. *Psychology of Aesthetics, Creativity, and the Arts*. https://doi.org/10.1037/aca0000284

Nadal, M., Munar, E., Marty, G., & Cela-Conde, C.J. (2010). Visual complexity and beauty appreciation: Explaining the divergence of results. *Empirical Studies of the Arts*, *28*(2), 173–191. https://doi.org/10.2190/EM.28.2.d

Navon, D. (1977). Forest before trees: The precedence of global features in visual perception. *Cognitive Psychology*, *9*(3), 353–383. https://doi.org/10.1016/0010-0285(77)90012-3

Palmer, S.E., Schloss, K.B., & Sammartino, J. (2013). Visual aesthetics and human preference. *Annual Review of Psychology*, *64*(1), 77–107. https://doi.org/10.1146/annurev-psych-120710-100504

Pérez, F., & Granger, B.E. (2007). IPython: A system for interactive scientific computing. *Computing in Science & Engineering, 9*(3). https://doi.org/10.1109/MCSE.2007.53

Poirel, N., Pineau, A., & Mellet, E. (2006). Implicit identification of irrelevant local objects interacts with global/local processing of hierarchical stimuli. *Acta Psychologica*, *122*(3), 321–336. https://doi.org/10.1016/j.actpsy.2005.12.010

Port, A. (2021). Svgpathtools (Version 1.4.1). Retrieved from https://github.com/mathandy/svgpathtools

Redies, C., Brachmann, A., & Wagemans, J. (2017). High entropy of edge orientations characterizes visual artworks from diverse cultural backgrounds. *Vision Research*, *133*, 130–144. https://doi.org/10.1016/j.visres.2017.02.004

Regebro, L. (2021). Svg.path (Version 4.1). Retrieved from https://github.com/regebro/svg.path

Robinson, A., Becker, R., the ReportLab team, & the community (2021). Reportlab: The Reportlab Toolkit (Version 3.6.1). Retrieved from http://www.reportlab.com/

Shier, J. (2011). Filling space with random fractal non-overlapping simple shapes. Hyperseeing, summer 2011 issue, 131–140, published by ISAMA (International Society of the Arts, Mathematics, and Architecture). Retrieved from http://www.isama.org/hyperseeing/11/11b.pdf

Shier, J., & Bourke, P. (2013). An algorithm for random fractal filling of space: An algorithm for random fractal filling of space. *Computer Graphics Forum*, *32*(8), 89–97. https://doi.org/10.1111/cgf.12163

Smets, G. (1973) *Aesthetic judgment and arousal: An experimental contribution to psycho-aesthetics*. Leuven, Belgium: Leuven University Press.

Spehar, B., Walker, N., & Taylor, R.P. (2016). Taxonomy of individual variations in aesthetic responses to fractal patterns. *Frontiers in Human Neuroscience, 10*. https://doi.org/10.3389/fnhum.2016.00350

Sun, Z., & Firestone, C. (2021). Curious objects: How visual complexity guides attention and engagement. *Cognitive Science, 45*(4). https://doi.org/10.1111/cogs.12933

Telenczuk, B. (2021). Svgutils (Version 0.3.4). Retrieved from https://svgutils.readthedocs.io

Thomas, B.G. (2012). 15 - Colour symmetry: The systematic coloration of patterns and tilings. In J. Best (Ed.) *Colour design*. https://doi.org/10.1533/9780857095534.3.381 (pp. 381–432): Woodhead Publishing.

Van der Hulst, E., Van Geert, E., & Wagemans, J. (in preparation). Shape variation in proximity grouping: An individual differences approach.

Van Geert, E., Hofmann, D., & Wagemans, J. (in preparation). The perception and appreciation of order and complexity.

Van Geert, E., & Wagemans, J. (2020). Order, complexity, and aesthetic appreciation. *Psychology of Aesthetics, Creativity, and the Arts*, *14*(2), 135–154. https://doi.org/10.1037/aca0000224

Van Geert, E., & Wagemans, J. (2021). Order, complexity, and aesthetic preferences for neatly organized compositions. *Psychology of Aesthetics, Creativity, and the Arts*, *15* (3), 484–504. https://doi.org/10.1037/aca0000276

Van Geert, E., Warny, A., & Wagemans, J. (in preparation). A systematic approach to study preferences for complexity.

Van Rossum, G., & Drake, F.L. (2009) *Python 3 reference manual*. CreateSpace: Scotts Valley, CA.

Vanderplas, J.M., & Garvin, E.A. (1959). Complexity, association value, and practice as factors in shape recognition following paired-associates training. *Journal of Experimental Psychology*, *57*(3), 155–163. https://doi.org/10.1037/h0042010

vgalin (2021). html2image (Version 1.1.2). Retrieved from https://github.com/vgalin/html2image

Wagemans, J. (2018). Perceptual organization. In J. T. Wixted, & J. Serences (Eds.), *The Stevens' Handbook of Experimental Psychology and Cognitive Neuroscience: Vol. 2. Sensation, Perception & Attention*. (pp. 803–872). Hoboken, NJ: John Wiley & Sons, Inc. https://doi.org/10.1002/9781119170174.epcn218

Westphal-Fitch, G., Huber, L., Gómez, J.C., & Fitch, W. T. (2012). Production and perception rules underlying visual patterns: Effects of symmetry and hierarchy. *Philosophical Transactions of the Royal Society B: Biological Sciences*, *367*(1598), 2007–2022. https://doi.org/10.1098/rstb.2012.0098

Wilson, A., & Chatterjee, A. (2005). The assessment of preference for balance: Introducing a new test. *Empirical Studies of the Arts*, *23*(2), 165–180. https://doi.org/10.2190/B1LR-MVF3-F36X-XR64