Interactive By-example Design of Artistic Packing Layouts

Bernhard Reinert[1] Tobias Ritschel[1,2] Hans-Peter Seidel[1]
MPI Informatik[1] MMCI / Saarland University[2]

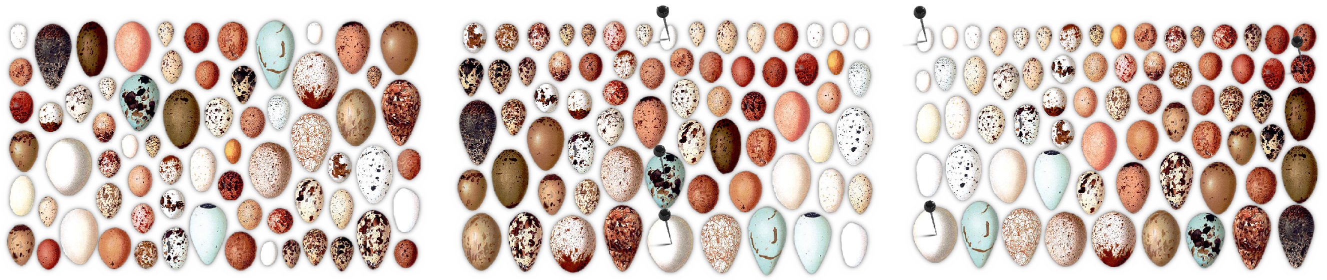


Figure 1: *Starting from a common layout* (Left)*, the user's objective is inferred from placement of three primitives (push pins), leading to a layout organized vertically by size* (Middle) *and after a different placement additionally by brightness horizontally* (Right).

Abstract

We propose an approach to "pack" a set of two-dimensional graphical primitives into a spatial layout that follows artistic goals. We formalize this process as projecting from a high-dimensional feature space into a 2D layout. Our system does not expose the control of this projection to the user in form of sliders or similar interfaces. Instead, we infer the desired layout of all primitives from interactive placement of a small subset of example primitives. To produce a pleasant distribution of primitives with spatial extend, we propose a novel generalization of Centroidal Voronoi Tesselation which equalizes the distances between boundaries of nearby primitives. Compared to previous primitive distribution approaches our GPU implementation achieves both better fidelity and asymptotically higher speed. A user study evaluates the system's usability.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation—Display algorithms I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction techniques;

Keywords: layout inference, packing, distribution, user interface

Links: DL PDF WEB VIDEO DATA

1 Introduction

Arranging sets of primitives into a pleasing spatial packing layout that tightly fills in 2D is tedious and requires expert skills (Fig. 2). While arrangements can serve for recreation and aesthetic purposes, they often seek to convey an underlying message concerning the relation between primitives and serve a didactical purpose. In this work, we propose a system to automate artistic layouts, by inferring the user's high-level intentions from the interaction performed. The interactive exploration of different artistic layouts and primitive relations enabled by our system goes beyond static print or display layouts and helps to improve general layouts, such required for Mind maps [Buzan 1976], tag clouds [Bateman et al. 2008] or any arrangement of graphical 2D primitives.

Fig. 1 shows three steps of a typical interaction using our system: After loading a set of primitives, our system presents a general-purpose layout (Fig. 1, *left*). To change this layout, a simple solution would be to expose many sliders that control what importance to what weight quality would be given. Such high-dimensional parameter spaces are hard to navigate for colloquial users and hamper creative exploration. Our system takes a different approach: We offer the user to move primitives to new positions (Fig. 1, *middle*) and by that to infer the user's intention, leading to a new layout, in this case, where primitives are organized vertically by size. After a second manipulation (Fig. 1, *right*) the layout is organized by brightness horizontally and by size vertically.

To allow such operations we make the following contributions:

- An interactive inverse layout approach to infer a user's packing layout intention from a small number of examples.
- A layout algorithm to evenly distribute primitives with spatial extend in real-time using a GPU.
- A study of packing layout task performance of novice users.

2 Previous Work

Properly distributing primitives in a domain has been a challenge in computer graphics for both, technical and aesthetical reasons. Seeking to place samples that evaluate a function such that aliasing is minimized, Mitchell [1987] argued that samples should have "blue noise" characteristics, that is: the distance to the neighbors should not be smaller than a threshold. Placing primitives for artistic purposes in 2D is widely used for non-photorealistic rendering, e. g., for stippling [Deussen et al. 2000; Hiller et al. 2003], mosaics [Hausner 2001; Kim and Pellacini 2002] or texture synthesis [Lagae and Dutré 2005]. In particular Hiller et al. [2003] who distributes primitives in the plane such that they follow a prescribed density, is a similar case of our system that produces distributions that follow rules inferred from the users' interaction with the distribution itself. Placing primitives in the plane, the usage of Voronoi tessellation is popular to avoid collision [Dalal et al. 2006] and achieve pleasant (temporal) distributions.

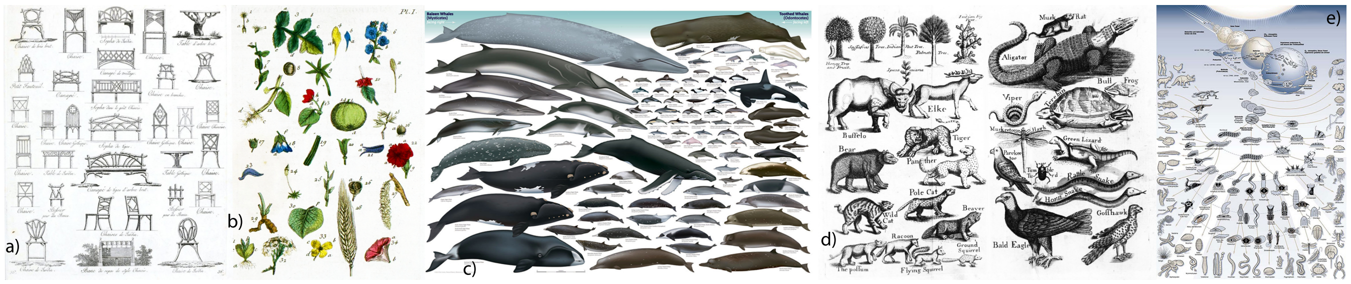


Figure 2: *Examples:* a): *G. Grohmann: "Recueil de dessins" (1805).* b): *Bulliard: "La Flore Des Environs de Paris" (1776).* c): *U. Gorter: "Whales of the World" (2003)* d): *J. Brickwil: "Natural history of North-Carolina" (1712).* e): *Schweizerbart: "Evolution der Tiere" (2001).*

The parameters for primitive placement can be difficult to control as noted by Hurtut et al. [2009] and Öztireli et al. [2012], who proposed to transfer the statistics from a source to a target distribution of primitives. Our approach is not based on distribution statistics. Instead, we infer high-level rules that describe the intended embedding of a high-dimensional feature space into a low dimensional medium from the user input instead of spatially-invariant statistics of items that cannot express all of the user's intention. Exploration of high-dimensional spaces of visual features were described by Lasram et al. [2012]. Beyond distribution statistics, grouping for stylization was described by Bezerra et al. [2008]: A layout of a scene is given, and the style of items is made coherent according to an observed grouping. A subset of our approach performs the inverse: We are given primitive features (e. g., brightness, shape, size) and want to find a layout.

Optimally placing a set of spatially extended objects into a constraining container (bin packing), such as 3D shapes into another 3D shape [Gal et al. 2007] or text into a 2D contour [Xu and Kaplan 2007; Maharik et al. 2011] is an NP-hard problem, but can be solved with sufficient approximate solutions in practice. Packing into a container can be one constraint among many in our system. Packing UV charts [Lévy et al. 2002] is a common technical challenge for surface parametrization. Closest to our objective is the approach of Yu et al. [2011] that arranges furniture in a room according to rules learned from exemplars in a forward procedure, without assistance for the user to change the layout or to learn from his feedback.

One of the most classic layout problems is desktop publishing and user interfaces [Lok and Feiner 2001; Jacobs et al. 2003]. Here, the state of the art is based on systems that exploit the regular, grid-structure of text layout, which does not generalize to arbitrary items. In information visualization, graph drawing [Harel and Koren 2002] and specifically word clouds [Bateman et al. 2008; Strobelt et al. 2012] share challenges such as collision avoidance with our approach. Placement of textual labels by example was considered by Vollick et al. [2007]. For the word clouds, user input has only been included in a forward manner in the ManiWordle system [Koh et al. 2010], where a user can fixate individual word primitives to specific locations. Different from such off-line systems, we account for visual features of the primitives themselves (and not only abstract word frequency), learn layout from user feedback and present new layouts, all on-line, with interactive performance.

Inferring a layout from sparse user constraints is an instance of semi-supervised learning [Chapelle et al. 2006], in particular semi-supervised dimensionality reduction [Zhang et al. 2007] where some primitives are labeled (i. e., placed by the user) and most are not. In the most general setting, our problem can be regarded as (inverse) procedural modeling, that can be solved with amazing results by approaches such as Metropolis [Talton et al. 2011], which is likely too costly to deliver timely response to the users interaction.

3 Overview

Conceptually, our system consists of an infinite loop: First, a *forward* layout step places primitives according to some rules (Sec. 4). If user interaction occurs, an *inverse* layout step (Sec. 5) refines the rules for the forward layout and the loop repeats.

A typical use case of the system is as follows (Fig. 1): Initially, the user is presented some generic layout of graphical primitives in 2D. This layout maps primitives with certain similar features (e. g., brightness, shape, etc.) to similar locations (Sec. 4.1). Primitives are placed in such a way, that the average distance of the boundary of nearby primitives (the "gap" between them) has a similar value everywhere (Sec. 4.2). Next, users interactively manipulate this layout by constraining a small number of primitives to particular locations (Sec. 5). This is depicted by a push-pin icon shown next to the constrained primitive. The system infers what features are to be used in the forward layout from these constraints.

4 Forward layout

Primitives are n objects represented as images with a possibly concave boundary Ω_i. A typical number of primitives is between 10 and 200. The next input to our approach is a per-primitive feature vector $\mathbf{f} \in \mathbb{R}^m$ representing $m \in \mathbb{N}$ different features. Features capture visual properties, such as size, shape, brightness, texture, etc. (automatically extracted from the input primitive by image processing) as well as semantic quantities like age, strength, etc. (acquired from a database). The sequence of the feature vectors of all primitives is denoted as $F = \{\mathbf{f}_1, \ldots, \mathbf{f}_n \in \mathbb{R}^m\}$. A typical number of features is 10. Output of our system is a sequence $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^2\}$ which contains locations at which primitives are to be placed in the two-dimensional layout space $\mathbb{R}^2$ (Fig. 3).

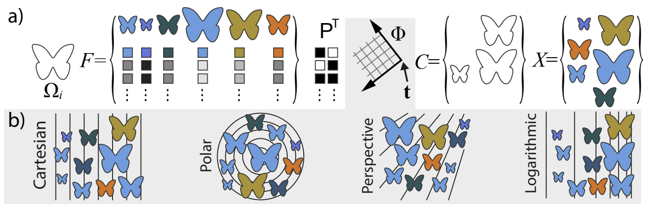


Figure 3: a): *Notations of our formalization.* b): *Isolines of first parameter dimension for different layout functions ϕ.*

Forward layout is performed in two main steps to be explained next: *feature mapping* (Sec. 4.1) and *distributing primitives with spatial extent* (Sec. 4.2).

4.1 Feature mapping

Feature mapping reduces high-dimensional features $\mathbf{f} \in \mathbb{R}^m$ to their low-dimensional 2D layout coordinates $\mathbf{x} \in \mathbb{R}^2$ as

$$\mathbf{x} = \phi(\mathsf{P}\mathbf{f} + \mathbf{t}), \tag{1}$$

where P is a *feature projection* matrix, $\mathbf{t}$ is a *parameter translation* and ϕ is a *layout function*, all explained in the next paragraphs.

Feature projection and parameter translation is performed by a tuple $(\mathsf{P}, \mathbf{t})$ as a projection matrix $\mathsf{P} \in \mathbb{R}^{2\times m}$ and a translation vector $\mathbf{t} \in \mathbb{R}^2$ that map feature vectors $\mathbf{f} \in \mathbb{R}^m$ to parameter vectors $\mathbf{p} \in \mathbb{R}^2$ as $\mathbf{p} = \mathsf{P}\mathbf{f} + \mathbf{t}$. P is non-zero only at position k, l if feature l is mapped to dimension k. The value at $\mathsf{P}_{k,l}$ gives the factor by which the feature is scaled to create the parameter vector dimension. Per dimension/row only one non-zero element/feature scaling-factor is present, i. e., only one feature is used per parameter vector dimension. As an example given three features (size, brightness, anisotropy)

$$\mathsf{P} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0.2 & 0 \end{bmatrix}$$

would select the third feature (anisotropy) with unit scaling as the first dimension and the second feature (brightness) as the second dimension, scaled by 0.2. $\mathbf{t}$ is used to shift the parameter vector along the parameter axis, and can be used for example to move the primitives along the axes in layout space in a Cartesian layout.

Layout function The parameter vector $\mathbf{p}$ serves as input to different layout functions $\phi(\mathbf{p}) \in \mathbb{R}^2 \to \mathbb{R}^2$. Such functions map e. g., the first parameter to the x-axis and the second one to the y-axis in a Cartesian layout, or the first parameter to angle and the second to radius in a radial layout function. In practice different layout functions can be used (Fig. 3). The only requirement for ϕ is that the inverse mapping ϕ^{-1} needs to exist in order to perform the inverse layout (Sec. 5).

Incomplete case We also support to select only one, or no feature at all, i. e., where P does not have full rank with rows of only zeros. In this case, the missing dimensions in $\mathbf{p}$ are created using Multidimensional scaling (MDS) [Cox and Cox 2000] of all remaining features, i. e., the features with columns that have zeros in P. The resulting parameter vector $\mathbf{p}$ can then be fed into ϕ as before.

4.2 Distributing primitives with extent

Preserving a balanced distance to all adjacent primitives is a key to a good layout. The output layout of the feature mapping however can be arbitrary with possibly overlapping primitives, that do not necessarily occupy the given layout space evenly. Further feature mapping only operates on points and has no concept of spatial extent. To distribute primitives with extent we equalize the distance between the boundaries of nearby objects. For primitives of complex shape and varying size, this leads to more pleasant distributions, yet resulting in simple computations that allow for a real-time GPU implementation.

Boundary Voronoi tesselation For point primitives, *Centroidal Voronoi Tesselation* (CVT) [Lloyd 1982] has proven to produce layouts that yield balanced point distances. For our purpose, one option would be to use the extension of Hiller et al. [2003] for general shapes. We implemented this approach but observed unsatisfying results: All but almost ellipsoidal shapes produce unbalanced results that drift (see Fig. 4). This drift arises from that fact, that CVT does not explicitly state boundary distances in its objective function but aligns the centroids of the primitive and its Voronoi region.

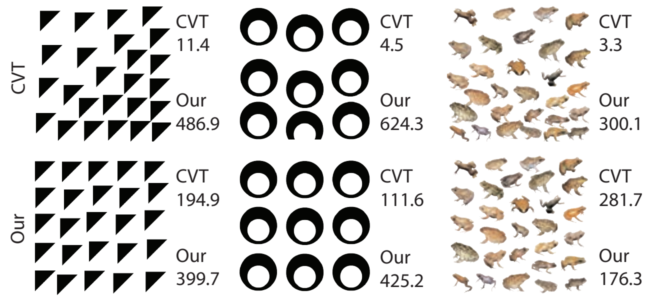
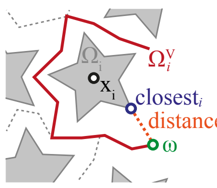


Figure 4: *Results for CVT [Hiller et al. 2003] (upper row) and our relaxation (lower row). Adjacent to the layouts the values of the CVT residual and our residual (Eq. 3). CVT relaxation results in unbalanced layouts and its residual in each column is lower for CVT relaxation. In contrast our residual is lower for the more balanced layouts, indicating that only our residual measures the quality.*

To achieve an even distance between primitives, the boundary distances need to be explicitly included in our objective function. The deviation of a layout X from this equilibrium can be measured by summing the squared distances between each primitive's boundary and its Voronoi region boundary:

$$c(X) = \sum_{i=1}^{n} \int_{\Omega_i^{\mathrm{V}}} \mathrm{distance}_i(\omega, \mathbf{x}_i)^2 \mathrm{d}\omega, \tag{2}$$



where Ω_i^{V} is the boundary of the i-th primitive's Voronoi region and $\mathrm{distance}_i(\omega, \mathbf{x}_i) : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}$ gives the shortest Euclidean distance between ω, a point on the Voronoi region boundary, and the i-th primitive's boundary Ω_i positioned at $\mathbf{x}_i$. The minimum of this cost function $X' = \mathrm{argmin}_X\, c(X)$ yields the optimal solution. This formulation is very similar to Eq. 1 in Dalal et al. [2006], except that we only consider the boundary of the Voronoi region and not its interior (we also omit the rotational parts as we are explicitly only interested in a translation). In practice, we perform all calculations on a discretized grid, i. e., Eq. 2 becomes

$$c(X) = \sum_{i=1}^{n} \sum_{\omega \in \Omega_i^{\mathrm{V}}} \mathrm{distance}_i(\omega, \mathbf{x}_i)^2. \tag{3}$$

As minimizing Eq. 3 is NP-hard, finding the global optimum is infeasible. Moreover, we are explicitly not interested in the global optimum of Eq. 3 as it possibly shuffles all primitive positions to new places, whereas we seek to find a solution, that is similar to the one produced by the forward mapping (Sec. 4.1), but balances primitive distances. Hence the global optimization of Eq. 3 is replaced by a local iterative one, that tries to find a small offset for each primitive position individually, given a static Voronoi diagram per iteration. The formula then becomes

$$c_i(\mathbf{x}_i) = \sum_{\omega \in \Omega_i^{\mathrm{V}}} \mathrm{distance}_i(\omega, \mathbf{x}_i)^2. \tag{4}$$

This equation could be minimized using image correlation, as done by Dalal et al. [2006], whose complexity is $\mathcal{O}(na \log a)$ where n is the number of primitives and a is the total number of pixels of the domain. This complexity is prohibitively expensive for our realtime needs. Hence Eq. 4 should be replaced by an approximation.

A first idea could be to replace $\mathrm{distance}_i(\omega, x_i)$ by a Taylor polynomial [Pottmann and Hofer 2003] of degree one and solve this approximated objective $\hat{c}_i$ instead of Eq. 4. However c_i is only poorly approximated by $\hat{c}_i$ and the error between both can be arbitrarily large (see Fig. 5). Furthermore Taylor polynomials require derivatives, whereas $\mathrm{distance}_i(\omega, x_i)$ is not necessarily continuously differentiable.

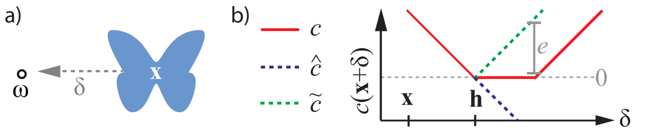


Figure 5: a): *In this 1D example the primitive at* $\mathbf{x}$ *is shifted by* δ*. The distance function* c *and its approximations* $\hat{c}$ *and* $\tilde{c}$ *measure the shortest distance to* ω*.* b): *The distance functios are plotted for an approximation at* $\mathbf{x}$*. For all values* $\delta \leq \mathbf{h}$ *the approximations* $\hat{c}$ *and* $\tilde{c}$ *exactly conform with* c*. For values* $\delta > \mathbf{h}$ *both functions deviate from the correct function, but the error of* $\tilde{c}$ *is bounded by* e*.*

We can reformulate Eq. 4 by expanding $\mathrm{distance}_i$ as

$$c_i(\mathbf{x}_i) = \sum_{\omega \in \Omega_i^{\mathrm{V}}} \|\mathrm{closest}_i(\omega, \mathbf{x}_i) - \omega\|_2^2$$

where $\mathrm{closest}_i(\omega, \mathbf{x}_i) : \mathbb{R}^2 \times \mathbb{R}^2 \rightarrow \mathbb{R}^2$ gives the point closest to ω on the boundary Ω_i of the i-th primitive at position $\mathbf{x}_i$. $\mathrm{closest}_i(\omega, \mathbf{x}_i)$ can be approximated for a point $\mathbf{x}_i + \delta_i$ as

$$\mathrm{closest}_i(\omega, \mathbf{x}_i + \delta_i) \approx \mathrm{closest}_i(\omega, \mathbf{x}_i) + \delta_i, \tag{5}$$

i. e., the closest position to ω on Ω_i at position $\mathbf{x}_i + \delta_i$ is approximately the closest position on Ω_i at position $\mathbf{x}_i$ with the added offset δ_i. This approximation does not involve a derivative and always gives a point on the primitive's boundary. The maximal error is therefore bounded by the maximal distance of two points on the primitive's boundary (see Fig. 5). Using Eq. 5 the cost of an offset δ_i is given for the primitive positions $\mathbf{x}_i$ as

$$\tilde{c}_i(\mathbf{x}_i + \delta_i) = \sum_{\omega \in \Omega_i^{\mathrm{V}}} \|\mathrm{closest}_i(\omega, \mathbf{x}_i) + \delta_i - \omega\|_2^2 .$$

The optimal δ_i' is found by setting its derivative to zero as:

$$\delta_i' = \operatorname*{argmin}_{\delta_i} \tilde{c}_i(\mathbf{x}_i + \delta_i) = \frac{1}{|\Omega_i^{\mathrm{V}}|} \sum_{\omega \in \Omega_i^{\mathrm{V}}} \omega - \mathrm{closest}_i(\omega, \mathbf{x}_i).$$

As an intuition behind this solution, finding the minimum can be regarded as a set of springs located at the Voronoi region boundary that try to push or pull the primitive into the correct place in its Voronoi region. A single step in our iteration has a complexity of $\mathcal{O}(n|\Omega^{\mathrm{V}}|)$ where $|\Omega^{\mathrm{V}}|$ is the total length in pixel of all Voronoi region boundaries.

Except for only considering the Voronoi boundary, for the special case of point primitives CVT is equivalent with our problem statement in Eq. 3. The Voronoi diagram for overlapping primitives with spatial extend is not well-defined. To overcome this, we compute the Voronoi diagram using a two-sided distance to the boundaries of the primitives, such that distances increase inside and outside of the boundary, leading to well-defined Voronoi diagrams. The primitives might have interior boundaries (e. g., the circular cutout of the circles in Fig. 4, b), that should not influence the relaxation. By iterating over the Voronoi boundary the closest point on the primitive boundary by definition is on the outermost boundary. Due to our approximations, in rare cases the converged distributions might still have overlapping primitives.

Implementation We use a GPU to accelerate our relaxation. Two maps are pre-calculated for each primtive: The first one contains the distance of every pixel to the primitive boundary, the latter holds the location of the closest position on the boundary. At runtime, we use rasterization [Hoff et al. 1999] in combination with the distance map to create a Voronoi diagram holding the index of the closest primitive at each pixel and an additional map giving the closest position on the primitive's boundary. The calculation of δ_i then becomes a parallel summation over all Voronoi region boundary pixels with a single diagram lookup per pixel to acquire the closest point on the primitive's boundary. The resulting relaxation is at least as efficient CVT: Typically, we use 30 relaxations in every frame and achieve more than 10 fps for more than 200 primitives including drawing in HD resolution on a Nvidia GTX 680 GPU.

Extensions Arbitrary global boundaries, such as the shape of the butterfly in Fig. 8, can be handled by removing the pixels of the Voronoi regions that are outside of the global boundary. Parts of the primitives might fall outside of the global boundary. Therefore for each pixel of primitive i outside of the global boundary an offset is added to δ_i in the direction of the closest point on the global boundary, pushing the primitive back in.

5 Inverse Layout

The user can define primitive constraints, i. e., force certain primitives with indices $C = \{c_1, \ldots, c_o \in (1, n)\}, o \in \mathbb{N}$ to be located exactly at position $\mathbf{x}_{c_i}$. The inverse layout computes a new feature projection matrix P, a parameter translation vector $\mathbf{t}$ and a new layout function ϕ (as defined in Sec. 4.1) that best "explain" the placement of the o constrained primitives, i. e., given the primitive positions X, their features F and constraints C we try to find ϕ and $(\mathsf{P}, \mathbf{t})$ (cf. Eq. 1) minimizing

$$\sum_{a \in C} \|\mathsf{P}\mathbf{f}_a + \mathbf{t} - \phi^{-1}(\mathbf{x}_a)\|. \tag{6}$$

Before any user interaction, P is set to zero and ϕ to the identity. After a user changed the constraints we try to minimize Eq. 6. Solving this task is split into two parts: enumeration of all plausible *layout functions* ψ and computation of the optimal *feature mapping* for it. Finally we choose the layout hypothesis and feature mapping for which a residual function $r(\psi)$ is minimal (*layout selection*). We will explain both steps in the following paragraphs.

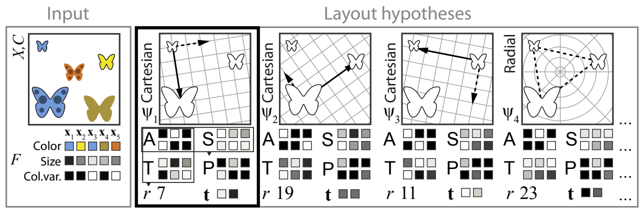


Figure 6: *Inverse layout: Input are the primitives and their positions* X*, constraints* C *and features* F*. From* C *several layout hypotheses* ψ_i *are build and the best feature-dimension mapping* A *and its residual* r *are calculated. Output is the* ψ_i *with the lowest cost, its projection matrix* P *and the translation vector* $\mathbf{t}$ *(*ψ_1*, black border).*

Layout functions We restrict our enumeration to a plausible subset of all possible layout functions (the bijections on $\mathbb{R}^2$), which we call layout hypotheses. The Cartesian and radial layout hypotheses are build independently as follows.

A *Cartesian* layout function hypothesis is defined by two axes. We assume one axis is the connection between two constraint positions, i. e., for each $a, b \in C$ with $a \neq b$ the first axis is $\mathbf{u} = \frac{\mathbf{x}_a - \mathbf{x}_b}{\|\mathbf{x}_a - \mathbf{x}_b\|_2}$. The second axis is then chosen orthogonal, i. e., $\mathbf{v} = (-\mathbf{u}_y, \mathbf{u}_x)^\mathsf{T}$ and the layout function becomes $\psi(\mathbf{p}) = (<\mathbf{u}, \mathbf{p}>, <\mathbf{v}, \mathbf{p}>)^\mathsf{T}$, resulting in $\frac{o(o-1)}{2}$ layout hypotheses.

A *radial* layout function hypothesis is defined by a center position $\mathbf{u}$. Our hypothesis assume that the center position is either located at one of the constraints, i. e., $\mathbf{u} = \mathbf{x}_a$ with $a \in C$ or that its position is the centroid, i. e., $\mathbf{u} = \frac{1}{o}\sum_{a \in C} \mathbf{x}_a$, or the center of the bounding box of all constraints, i. e., $\mathbf{u} = \frac{1}{2}(\min_{a \in C} x_a + \max_{a \in C} x_a)$. The layout function then is $\psi(\mathbf{p}) = (\mathbf{u}_1 + \mathbf{p}_1 \sin \mathbf{p}_2, \mathbf{u}_2 + \mathbf{p}_1 \cos \mathbf{p}_2)^\mathsf{T}$, resulting in $o + 2$ different layout hypotheses.

Feature mapping For a fixed layout hypothesis ψ its residual r and its optimal feature projection matrix P are found as follows: All constraint positions are mapped back to parameter space as $\mathbf{p}_i = \psi^{-1}(\mathbf{x}_i)$, i. e., the inverse of the layout function. Next, we need to find how well a common scalar $\mathsf{S}_{k,l}$ of the k-th dimension-wise difference of the parameter vector could "explain" the differences of the l-th feature. For example, we would like to know how well a common scaling of all butterfly size differences can "explain" the differences of the second layout parameter, which again could be the radii of a radial layout or the vertical coordinates in a Cartesian layout. We combine the cost of "explaining" parameter dimension k and feature l using the scaling matrix $\mathsf{S}' \in \mathbb{R}^{2 \times m}$ as the residual matrix functional $\mathsf{R} \in \mathbb{R}^{2 \times m} \to \mathbb{R}^{2 \times m}$ defined as

$$\mathsf{R}_{k,l}(\mathsf{S}') = \sum_{a,b \in C} \left(\mathsf{S}'_{k,l}(\mathbf{f}_a - \mathbf{f}_b)_l - (\mathbf{p}_a - \mathbf{p}_b)_k\right)^2.$$

For each dimension k and feature l the optimal scaling factor is found by setting its derivatives to zero as

$$\mathsf{S}_{k,l} = \underset{\mathsf{S}'}{\operatorname{argmin}}\, \mathsf{R}_{k,l}(\mathsf{S}') = \frac{\sum_{a,b \in C}(\mathbf{p}_a - \mathbf{p}_b)_k(\mathbf{f}_a - \mathbf{f}_b)_l}{\sum_{a,b \in C}(\mathbf{f}_a - \mathbf{f}_b)_l^2}.$$

Using these scaling factors yields the minimal residual matrix $\mathsf{T} = \mathsf{R}(\mathsf{S})$. Now the assignment of features to dimensions can be found as a solution to the general assignment problem [Martello and Toth 1987], such that every dimension is assigned to exactly one feature, using the costs from matrix T. The result is an assignment matrix $\mathsf{A} \in \{0, 1\}^{2 \times m}$ where $\mathsf{A}_{k,l} = 1$, if feature l is mapped to dimension k, and $\mathsf{A}_{k,l} = 0$ otherwise. As T is small, enumerating all $m(m-1)$ possible partial assignments is feasible. The residual of ψ is then given by

$$r = \|\mathsf{A} \circ \mathsf{T}\|_1, \tag{7}$$

where $\circ$ denotes the component-wise (Hadamard) product of two matrices. In other words the product keeps only the elements of T where A is non-zero. The 1-norm of the resulting matrix then gives the total absolute residual for all dimensions as a single scalar value.

Layout selection A unique best solution to the inverse layout problem can be found for $o \geq 3$. For the hypothesis ψ with the smallest residual r we set $\phi = \psi$ and $\mathsf{P} = \mathsf{A} \circ \mathsf{S}$, i. e., we keep the best layout function with the best feature projection. The optimal parameter translation vector can then be found as the offset of the centroids, i. e., $\mathbf{t} = \frac{1}{o}\sum_{a \in C} \mathbf{p}_a - \mathsf{P}\mathbf{f}_a$.

Incomplete case It might not be possible to reliably infer the intended layout from the user constraints, because either not enough constraints are provided ($o < 3$), they are contradicting or they are non-unique. If the residual for all layouts is too high, we repeat the above procedure for a single dimension only, producing a matrix with a zero second row. As explained in Sec. 4.1, MDS will then be used to create the second parameter coordinate. If the residual for a single component-explanation is still too high or no hypothesis could be build, change MDS is used for both dimensions.

6 Results

Example Layouts In our main result, a user constraints a set of primitives. Starting from the initial distribution she can manipulate the layout and the system tries to infer her layout idea (see Fig. 7 and caption). The special case of packing into a particular container, can be combined with additional constraints and layout functions, as seen in Fig. 8. Finally, our system can be used to interactively explore multi-dimensional datasets, such as the countries in Fig. 9.

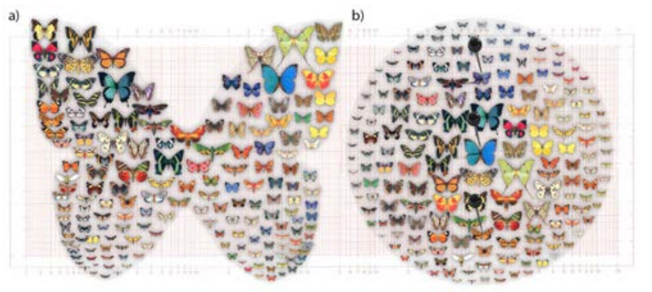


Figure 8: *a) Layout of butterflies inside the boundary of a butterfly. b) A radial layout: radius maps to size and hue to angle.*

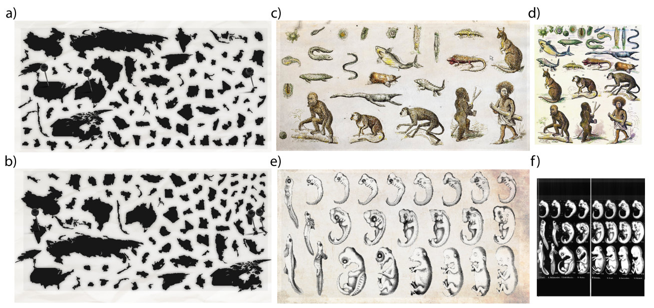


Figure 9: *Semantic layouts:* a): *Layout by infant mortality rate from the World Fact Book.* b): *Now by population size.* c): *A remix of "The Descent of Men" by Ernst Häckel (1834–1919) seen in* (d). e): *A remix of another layout by Häckel* (f).

User Study To design a user study, we first conducted a pilot questioning of two professional artists with a special expertise in producing packing layouts. The full questionnaire with the artists' answers can be found in the supplemental material. According to the artists, besides legal and research issues, the most time-consuming steps are the generation of graphical primitives itself and the final layout. They say, that achieving a balanced primitive distance (Sec. 4.2) is a major challenge. Here the artists wish to have "an automated layout tool creating a spatial boundary between primitives" and that these tools were "easier to work with" than the currently available software. According to the artists, the position of the primitives is governed by their "taxonomic relationship" and by visual features such as brightness, texture or drawing style but also by arranging

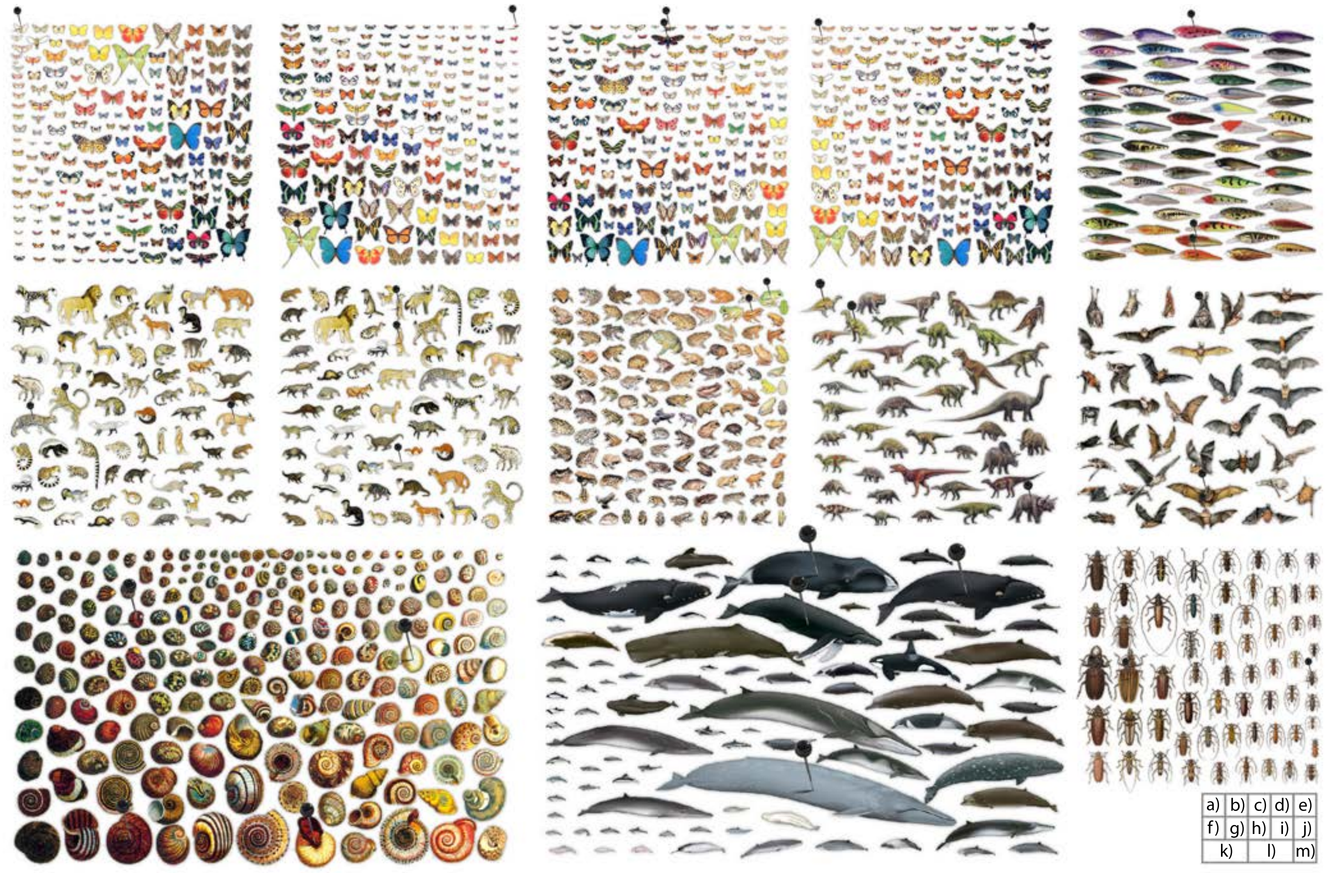


Figure 7: *Results of our approach (layout feature in brackets).* a–d): *Butterflies (brightness, size, shape anisotropy, shape anisotropy plus brightness).* e): *Fishing baits (hue).* f,g): *African carnivores (brightness and variance plus orientation).* h): *Frogs (brightness plus variance).* i): *Dinosaurs (brightness).* j): *Bat layout by (shape anisotropy).* j): *Sea moluscae (size plus brightness).* k): *Whales (brighness).* k): *Beetles (size).*

them in a "visually pleasing way". The programs used by artists seem to be Adobe PhotoShop, Illustrator, InDesign and Corel Draw which all do not offer tools to generate balanced primitive distance in two dimensions.

To assess the usefulness of our system, we conducted a user study, in which 15 naïve subjects were asked to produce a layout with three different user interfaces. The interfaces presented were a commercial software ("commercial interface", interface #1), an interface of our software with relaxation ("relaxation interface", interface #2, Sec. 4.2) and an interface with relaxation and inverse layout step (Sec. 5) ("our interface", interface #3). For the commercial interface, we let the subjects choose from either Adobe Illustrator CS6 or Microsoft PowerPoint 2010 and rate their expertise in these programs for the presented task after the user study on a scale from 1 (novice) to 5 (expert). The task was, to produce a layout, where primitives are organized by specific features and have a balanced distance to each other. The initial layout always was identical: a fully relaxed MDS layout (Sec. 4). Every session was self-timed until the subject was either satisfied with the generated layout or gave up because of fatigue. The order in which the interfaces were presented to the subjects was randomized for each trial. For every session we logged the time spend to produce it and the final layout image. For the relaxation interface and for our interface we additionally logged the cost (Eq. 7) of the final layouts. For the following analysis we excluded cases in which the subjects gave up due to fatigue. Three users chose Adobe Illustrator CS6 and rated their expertise with 3.67 on average, the other twelve users chose Microsoft PowerPoint 2010 with an average expertise rating of 3.5. The layout generation took 16:32 minutes on average for the commercial interface with a standard deviation of 8:07 minutes, for the relaxation interface 7:24 minutes on average with a standard deviation of 2:18 minutes and for our interface 1:49 minutes with a standard deviation of 0:52 minutes. We conclude with statistical significance ($p < 0.0001$, ANOVA) that our interface results in a speedup of approximately one order of magnitude. In a second user study we asked a second group of 19 subjects to rate the layouts produced by the subjects of the first user study. The three layouts of each subject of the first user study were presented to the new subjects and they had to pick the layout which they found best accomplished the task at hand. We found that in 62.63 % of the cases the layout produced by our interface was the preferred one, while 27.37 % preferred the one generated with the relaxation interface and 10 % were in favor of the layout of the commercial interface. We conclude with statistical significance ($p < 0.0004$, binomial between all pairs), that results from our interface are preferred over all alternatives at least by a factor of two. In 69.59% of the cases the subjects' choice correlates with the residual value of the first study.

7 Discussion and Conclusion

This paper presented an approach to perform artistic primitive layouts. Technically, we stated the problem as a projection from the space of visual and semantic features into a lower-dimensional layout space combined with a GPU-based relaxation to achieve an equal distance between primitive boundaries. The parameters of this projection were learned from user feedback.

The feedback received from artists, as well as both parts of the user study show that producing packing layouts is challenging and benefits from computational support. Specifying the underlying model by hand involves selecting the layout function (e. g., linear layout), the feature dimensions (e. g., axis direction), a feature mapping (e. g., size to axis-0), a feature scaling factor and a fitting of the specified layout to the user constraints. Our widget-free interface can not replace professional data analysis. It could, however, be used by novice users, e. g., children in a museum: Such users might not be willing to participate in a formal user dialog involving concepts like "sorting". However they might play around and manipulate butterflies and get insight from how our system reacts, without a particular goal in mind, maybe even without knowing they perform man-machine interaction in the first place.

Our system faces the same challenge as other system making suggestions to users: it might provide undesired suggestions. Also, it is not yet able to infer high-level layout, such as the symmetry in Fig. 2, which would require a more sophisticated search for ϕ. Using a general combination of different features for the layout, instead of only two distinct features, also remains future work. Finally, we would like to complement the system by active learning: If a user organizes bright small eggs (Fig.1) along the horizontal line, the system could ask if the feature that was meant was "bright" or "small" or both by displaying different suggestions.

References

BATEMAN, S., GUTWIN, C., AND NACENTA, M. 2008. Seeing things in the clouds: the effect of visual features on tag cloud selections. In *Proc. ACM Hypertext and Hypermedia*, 193–202.

BEZERRA, H., EISEMANN, E., DÉCORET, X., AND THOLLOT, J. 2008. 3D dynamic grouping for guided stylization. In *Proc. NPAR*, 89–95.

BUZAN, T. 1976. *Use both sides of your brain*. Dutton.

CHAPELLE, O., SCHÖLKOPF, B., ZIEN, A., ET AL. 2006. *Semi-supervised learning*, vol. 2. MIT press Cambridge, MA:.

COX, T., AND COX, M. 2000. *Multidimensional scaling*.

DALAL, K., KLEIN, A., LIU, Y., AND SMITH, K. 2006. A spectral approach to npr packing. In *Proc. NPAR*, 71–78.

DEUSSEN, O., HILLER, S., VAN OVERVELD, C., AND STROTHOTTE, T. 2000. Floating points: A method for computing stipple drawings. In *Comp. Graph. Forum*, vol. 19, 41–50.

GAL, R., SORKINE, O., POPA, T., SHEFFER, A., AND COHEN-OR, D. 2007. 3d collage: expressive non-realistic modeling. In *Proc. NPAR*, 7–14.

HAREL, D., AND KOREN, Y. 2002. Drawing graphs with non-uniform vertices. In *Proc. Working Conference on Advanced Visual Interfaces*, 157–166.

HAUSNER, A. 2001. Simulating decorative mosaics. In *Proc. SIGGRAPH*, 573–580.

HILLER, S., HELLWIG, H., AND DEUSSEN, O. 2003. Beyond stipplingmethods for distributing objects on the plane. *Computer Graphics Forum 22*, 3, 515–522.

HOFF, K. I., KEYSER, J., LIN, M., MANOCHA, D., AND CULVER, T. 1999. Fast computation of generalized Voronoi diagrams using graphics hardware. In *Proc. SIGGRAPH*, 277–86.

HURTUT, T., LANDES, P., THOLLOT, J., GOUSSEAU, Y., DROUILLHET, R., AND COEURJOLLY, J. 2009. Appearance-guided synthesis of element arrangements by example. In *Proc. NPAR*, 51–60.

JACOBS, C., LI, W., SCHRIER, E., BARGERON, D., AND SALESIN, D. 2003. Adaptive grid-based document layout. *ACM Trans. Graph. 22*, 3, 838–847.

KIM, J., AND PELLACINI, F. 2002. Jigsaw image mosaics. *ACM Trans. Graph. 21*, 3, 657–664.

KOH, K., LEE, B., KIM, B., AND SEO, J. 2010. Maniwordle: Providing flexible control over wordle. *IEEE Trans. Vis. Comp. Graph. 16*, 6, 1190–97.

LAGAE, A., AND DUTRÉ, P. 2005. A procedural object distribution function. *ACM Trans. Graph. 24*, 4, 1442–61.

LASRAM, A., LEFEBVRE, S., AND DAMEZ, C. 2012. Procedural texture preview. *Comp. Graph. Forum (Proc. EG) 31*, 413–20.

LÉVY, B., PETITJEAN, S., RAY, N., AND MAILLOT, J. 2002. Least squares conformal maps for automatic texture atlas generation. In *ACM Trans. Graph.*, vol. 21, 362–371.

LLOYD, S. 1982. Least squares quantization in pcm. *IEEE Transactions on Information Theory 28*, 129–137.

LOK, S., AND FEINER, S. 2001. A survey of automated layout techniques for information presentations. In *Proc. Smart Graphics*, 61–68.

MAHARIK, R., BESSMELTSEV, M., SHEFFER, A., SHAMIR, A., AND CARR, N. 2011. Digital micrography. *ACM Trans. Graph. (Proc. SIGGRAPH) 30*, 4, 100.

MARTELLO, S., AND TOTH, P. 1987. Linear assignment problems. *North-Holland Mathematics Studies 132*, 259–282.

MITCHELL, D. 1987. Generating antialiased images at low sampling densities. *Computer Graphics (Proc. SIGGRAPH) 21*, 65–72.

ÖZTIRELI, A. C., AND GROSS, M. 2012. Analysis and synthesis of point distributions based on pair correlation. *ACM Trans. Graph. (Proc. SIGGRAPH Asia) 31*, 6.

POTTMANN, H., AND HOFER, M. 2003. Geometry of the squared distance function to curves and surfaces. In *Visualization and Mathematics III*, Springer, 223–44.

STROBELT, H., SPICKER, M., STOFFEL, A., KEIM, D., AND DEUSSEN, O. 2012. Rolled-out wordles: A heuristic method for overlap removal of 2d data representatives. In *Comp. Graph. Forum*, vol. 31, 1135–44.

TALTON, J., LOU, Y., LESSER, S., DUKE, J., MĚCH, R., AND KOLTUN, V. 2011. Metropolis procedural modeling. *ACM Trans. Graph. 30*, 2, 11.

VOLLICK, I., VOGEL, D., AGRAWALA, M., AND HERTZMANN, A. 2007. Specifying label layout style by example. In *Proc. UIST*, 221–230.

XU, J., AND KAPLAN, C. 2007. Calligraphic packing. In *Proc. GI*, 43–50.

YU, L.-F., YEUNG, S. K., TANG, C.-K., TERZOPOULOS, D., CHAN, T. F., AND OSHER, S. 2011. Make it home: automatic optimization of furniture arrangement. *ACM Trans. Graph. (Proc. SIGGRAPH) 30*, 4, 86.

ZHANG, D., ZHOU, Z., AND CHEN, S. 2007. Semi-supervised dimensionality reduction. In *Proc. SIAM Data Mining*, 629–34.