# Preset Based Interaction with High Dimensional Parameter Spaces

Jarke J. van Wijk, Cornelius W.A.M. van Overveld

Eindhoven University of Technology
Dept. of Mathematics and Computer Science

## Abstract

Many systems require the setting of a large number of parameters. This is often a difficult and time consuming task, especially for novice users. A framework is presented to simplify this task. Settings are defined as a weighted sum of a number of presets, thereby bridging the gap between the expert mode of setting all individual parameters and the novice user mode of selecting presets. Several methods are presented to set the weights of the presets. With an interactive graphical widget, the preset controller, the user can change many parameters simultaneously in an easy and natural way. Also, methods for automated scanning of the parameter space are described. Three applications are presented: Color editing, morphing of drawings, and the control of a sound synthesizer.

**CR Categories:** H.5.2 [Information Interfaces and Presentation]: User Interfaces; I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction techniques

**Keywords:** Applications, HCI (Human-Computer Interface), Misc 2D graphics, User Interface Design, User Interfaces

## 1 INTRODUCTION

Many systems require their users to specify the values of a large number of input parameters. The system consumes this input, performs a simulation or changes something in the real world, and presents the result to the user. The user observes the result, and possibly modifies his input in order to improve the result. Let us consider some typical examples.

*Simulation:* the use of simulation is nowadays widespread, for instance for product design. The user can vary geometrical, chemical, physical, and other parameters, and perform simulations to determine widely varying functional characteristics such as strength, cost, and appearance. An approach to simplify interactive simulation is given in [9].

*Professional Equipment:* medical equipment for image acquisition, such as X-ray and MRI scanners, have a large number of controls concerning for instance dose, focus, and deflection, which have to be set per patient.

*Art and entertainment:* sound synthesizers are often based on the use of a network of components, such as oscillators and filters, each of which has several parameters. The setting of these parameters was, among others, addressed by Buxton et al. [3]. One of their requirements was that the system should encourage exploration and experimentation.

*Animation:* human body models for computer animations have many parameters to control aspects like the posture and facial expression. For instance, the character *Jack* [10] has 88 degrees of freedom, for newer cartoon characters several hundreds of parameters are used.

Such systems share the following characteristics:

- Two phases can be discerned. In the first phase a model or configuration is defined, in the second phase parameters are set. The first phase is crucial, but usually domain specific. In the remainder of this paper we will focus on the second phase.
- Many parameters, typically tens or hundreds, must be set. The size of the corresponding high-dimensional parameter spaces is overwhelmingly large;
- On the other hand, usually only a small subset of all possible settings is of interest and has practical value;
- The control of such systems requires much skill and knowledge, and is therefore often restricted to professional users;
- Especially in simulation, the relation between the result and each individual parameter is often not obvious;
- Several parameters must be changed simultaneously to achieve a certain effect;
- Often no hard and fast specifications of optimal results can be defined, which rules out automated optimization;
- A single optimum does not exist, different circumstances require different settings;
- Dynamic, or even real-time control is often needed.

As a result, finding suitable settings is often a process that requires much time, expertise, and effort. We propose a generic framework to control such systems in a more efficient and effective way. Within this framework the user can:

- Define subspaces of high-dimensional parameter spaces;
- Define new settings in an intuitive way;
- Explore the parameter space both manually and automatically;
- Take advantage of preset settings and explore their environment.

First we describe the basis of the framework: The use of weighted averages of presets. Next, an intuitive controller to define such weighted averages manually is presented, followed by two methods for automated scanning of the parameter space. The framework is applied to color editing, morphing of drawings, and the control of a sound synthesizer. Finally, the results are discussed.

The central theme of the work presented here is interaction, but there are two strong connections with computer graphics. First, the framework itself relies heavily on interactive computer graphics and visualization. Second, many computer graphics applications may benefit from the methods described here, because often a large parameter space has to be explored, looking for effective and interesting results.

## 2 INTERPOLATION OF PRESETS

An often used alternative to changing individual parameters is the use of presets. Experts prepare a number of settings for the parameters, such that each setting is useful in some sense. A sound synthesizer is a good example of this: Presets are provided to simulate a set of real and virtual musical instruments as well as possible. The

task of setting all parameters is now reduced to selecting a preset, so that also novice users can use the system. More experienced users can use presets as startpoints for variation of the basic parameters.

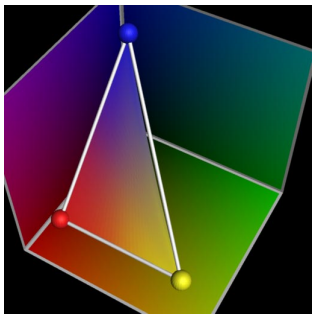

Figure 1: Interpolation in color space

Some systems offer an option to interpolate between two presets. Our framework is based on a generalization of this idea: A parameter setting is calculated as a weighted average of a number of presets. Specifically, we use a convex combination. Suppose that $n$ real valued parameters must be specified. The input is hence a vector $\mathbf{x} = [x_1, x_2, \ldots, x_n]$. Suppose we have $m$ presets $\mathbf{p}_i$, $i = 1, \cdots, m$, where each preset contains the values for all $n$ parameters. These presets can be interpolated via:

$$\mathbf{x} = \sum_{i=1}^{m} u_i \mathbf{p}_i$$

under the constraints:

$$u_i \geq 0, \sum_{i=1}^{m} u_i = 1,$$

where $u_i$ is the weight for preset $\mathbf{p}_i$. The weights sum up to one, each weight lies in between zero and one. The $m$-dimensional vector $\mathbf{u}$, together with the presets, define the setting.

The weighted presets define a convex subset of the parameter space, typically with a lower dimensionality than the original parameter space. A simple example is shown in figure 1. Suppose we must specify a color as an RGB-triplet [Red, Green, Blue] ($n = 3$), and suppose further that we do like mixes of primary colors. We therefore use three preset colors ($m = 3$): red [1,0,0], yellow [1,1,0], and blue [0,0,1], depicted as spheres. The triangle in between them represents all valid values for the $u_i$'s, from which we can pick our prefered color. At a corner, one single $u_i$ is 1 and the others are 0; along an edge, one $u_i$ is zero; in the interior all $u_i$ are non-zero.

This example shows also that the definition of the parameters is important. Here we use RGB as color model for tutorial purposes, mainly because the parameter space can be simply visualized as a cube. For application in practice, the definition of colors in a perceptually meaningful color space (such as Hue, Saturation, and Value) is of course a better choice, not only because the presets are easier to define, but also because interpolation gives more intuitive results.

This example generalizes to higher values of the number of parameters $n$ and the number of presets $m$. If $m$ presets are used, then the valid parameter settings form an $m$-1 dimensional convex object, embedded in an $n$-dimensional parameter space. For $m = 4$ this object is a tetrahedron.

One important assumption we make is that interpolation of the parameters is feasible and meaningful. For real valued parameters interpolation itself is no problem. However, some systems put additional constraints on sets of parameters, such that not each setting in between presets is allowed.

Ordinal parameters can be mapped on a real valued scale and rounded afterwards. For instance, we use 0 for *low*, 0.5 for *medium*, and 1 for *high*. Often, such a real valued scale is also used internally in the system, ordinal parameters with a limited number of steps are offered for user convenience.

Nominal parameters (such as country, type of vehicle) present a somewhat harder problem. A remedy here is to use the setting for which the sum of the relevant $u_i$'s is the highest.

These last methods for interpolation will not always give a meaningful result. However, even if we limit ourselves to straightforward systems that take in only real parameters without additional constraints, the number of applications is still very large.

## 3 PRESET CONTROLLER

The user can now control his input by changing the weights $u_i$. One way to realize this is via a set of sliders, one per $u_i$. The user can drag a slider to increase and decrease the influence of a preset. The other weights must be updated by the system, to make sure that all weights sum up to 1. This can be done by substracting the change from these other weights, taking care that they do not become negative.

Such a simple approach is not particularly attractive. Most of the sliders have to move simultaneously. As a result, it is often hard to return to an earlier setting. The visualization of the setting as a row of sliders is fairly abstract and hard to remember. Summarizing, this does not invite to play and experiment.

We have found a very simple and effective alternative way to control the weights. Figure 2 shows our preset controller. It consists of a rectangular area, where each preset is depicted via an iconic representation, here a circle and a label (A, B, ..., G). Also, the current setting is represented with an icon (here a cross). Typically, this controller is displayed on a computer screen and controlled via a mouse, but other realisations could apply as well. For example, for real-time control the current setting can be controlled with a joystick or via a touchscreen, for applications where multiple users participate the icons can be replaced by physical objects on a table [6, 11].

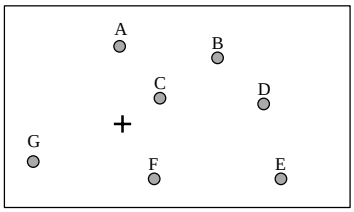


Figure 2: Preset controller

The values for the weights $u_i$ are set inversely proportional to the geometric distances $d_i$ from the circles to the cross: The closer a circle is to the cross, the larger its influence. Specifically, we use Shepard's method [13] to calculate the weights:

$$u_i = d_i^{-p} / \sum_{j=1}^{m} d_j^{-p},$$

with $p \geq 1$, typically 1 or 2. If a certain $d_i$ is zero, we set $u_i$ to one, and all other $u_j$'s to zero. Much more sophisticated methods for the interpolation of scattered points have been developed, for instance for cartography [7]. For our application, inverse distance weighting

usually suffices. In section 5 we will discuss some additional ways to modify the weights.

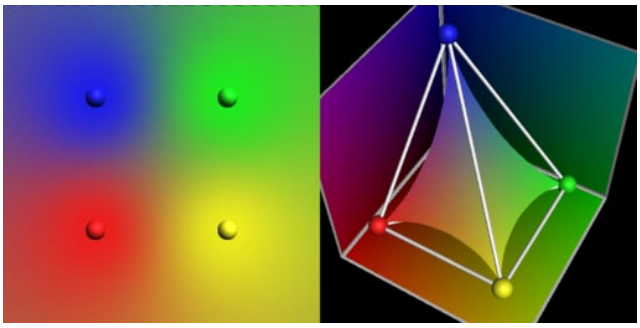

Figure 3: Preset controller for color

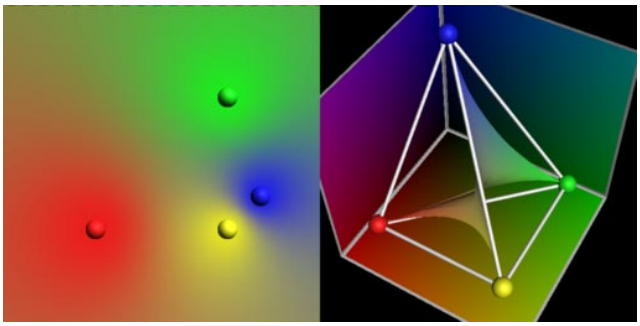

Figure 4: Blue control point moved

The user can now interact with this controller in two ways:

- by dragging circles, thereby changing the influence of the various presets on the current setting;
- by dragging the cross, thereby changing the influence of all presets simultaneously.

We found that this controller is easy and natural to use. It relies on a simple metaphor: Close means a strong influence. Also, it reduces the dimensionality of the search space further. The controller introduces a two-dimensional surface, on which the user can search for new settings by dragging the cross. The meaning of each position on this surface is defined by the position of the circles.

As an example, we use color editing again (figure 3). On the left the preset controller is shown. Four preset colors are used. The advantage of this simple example is that we can show the result of the interpolation directly in the controller itself, for most other applications this will not be true. On the right is shown how the presets span up a tetrahedron in parameter space. Also, the surface that results from mapping each point of the 2D surface of the controller to the parameter space is shown. This surface interpolates the presets. The presentation on the right is shown here just to visualize the principle of the method, it does not aid the end-user and can not be generalized to higher dimensional input space.

The user can pick colors from the surface; the shape of the surface can be modified by dragging the presets (figure 4). Finally, the presets themselves can be edited, thereby changing the position of the vertices in parameter space.

Superficially, the preset controller resembles images produced by statistical analysis techniques such as Multi-Dimensional Scaling [5]. The aim of these methods is to analyze n-dimensional data via projection on the plane such that interpoint distances are preserved as well as possible, thereby preserving structure. With the preset controller the user is completely free to position his presets on the plane. The tool is meant for design, not for analysis.

### 3.1 Ranges of settings

Often a range of parameter settings has to be found instead of a single parameter setting. For instance in our example, not a single color, but a color map for data visualization has to be found. Animation is another typical example: Instead of a single posture of a cartoon character, a fluent motion has to be defined. Concerning sound synthesis: in real time performance it is desired to vary timbral properties with continuous one-dimensional controllers, such as pedals and sliders.

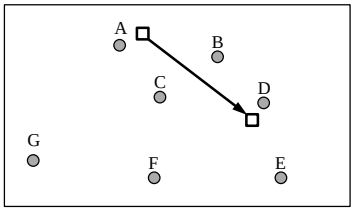


Figure 5: Range control

The task now is to define a continuous function $\mathbf{u}(s)$, with $s \in [0, 1]$, such that the weighted presets trace out a curve in parameter space. Figure 5 shows how this can be realized as an extension of the preset controller. Instead of a single icon for the current setting, the user can drag the begin and endpoint of a line segment, here depicted as an arrow with two boxes. At the start, the parameter setting will be similar to preset A, at the end similar to preset D, while C and B influence the settings in between. Instead of a line segment, more complex curves can be used here, but the added complexity does not always pay off. The user has already enough flexibility in the definition of the range by dragging the icons of the presets, and the begin and endpoint.

### 3.2 Shading

We have experimented with several ways to give the user more feedback on the weights of the presets when manipulating the controller. One option tested was to show dynamic rulers, including tick-marks, from the current setting to the presets, but this added more confusion than support. Another approach was more succesful: Filling in the background with a grey shade to visualize the structure in the positions of the preset icons.

Specifically, we consider the preset controller as a top view on a curved surface $z(x, y)$, given by:

$$z(x, y) = a \min_{i=1}^{m}(d_i^2(x, y)),$$

where $d_i$ is the geometric distance of a point $[x, y]$ to preset icon $i$ at point $[x_i, y_i]$, and $a$ is a (usually negative) scale factor. In other words, we put a paraboloid below each preset icon. This surface is transformed into a grey shade image via a simple diffuse reflection model. A normal $\mathbf{n}$ on the surface is given by:

$$\mathbf{n} = [2a(x - x_j), 2a(y - y_j), 1],$$

where $j$ is the index of the nearest preset icon. The intensity $I$ is then given by:

$$I = I_a + I_s \max(0, \frac{\mathbf{n}\cdot\mathbf{l}}{|\mathbf{n}||\mathbf{l}|})$$

where $I_a$ is the intensity of ambient light, $I_s$ is the intensity of a directional light source, and $\mathbf{l}$ is a vector that points towards this light source.

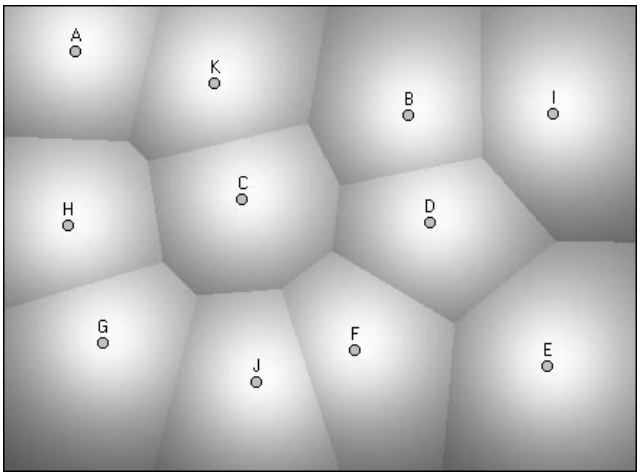


Figure 6: Shaded Voronoi diagram

A result is shown in figure 6. An almost perpendicular light direction was used: $\mathbf{l} = [0.05, 0.1, 1.0]$. Reactions of users fell into two categories. Some had a strong association with a steeldrum, others recognized the image as a shaded Voronoi diagram. A Voronoi diagram is a fundamental geometric datastructure [14, 2]. Given a space $S$ and a set of $n$ points (or *sites*) $\mathbf{p_i}$, a Voronoi diagram is a partioning of $S$ into regions $s_i$, where each region $s_i$ is the set of points that have site $\mathbf{p_i}$ as its nearest neighbor.

It is obvious that the shaded surface leads to a Voronoi diagram. The surface $z(x, y)$ is smooth, except when two preset icons are at the same minimal distance. The shading emphasizes the discontinuities. The shaded background provides a direct, visual answer to various user questions: Which preset has most influence, and at which locations have two or three presets the same, maximal influence. For these questions, a line-drawn Voronoi diagram would also suffice. The shading, however, has some additional advantages. The grey shade gives an additional cue on distance, and makes it easier to locate the site that defines a region. Further, the raster graphics grey shade image in the background can be separated effortlessly from the vector-oriented markers. The contrast can be tuned via the scaling factor $a$. Finally, the shaded background simply looks attractive.

Images such as figure 6 can be generated in real time, so the user is provided with continuous feedback while dragging the preset icons. We achieved this performance as follows. A quad-tree approach was used to reduce the number of sites to consider per pixel. The recursive process proceeds as follows. The input is a rectangle and a list of sites which can cover the rectangle. For all sites in the list the minima and maxima of $d_i$ for the rectangle are determined. Next, sites $j$ for which $\min(d_j) > \min(\max(d_i))$ are removed from the list, because they do not contribute to the image in the rectangle. If one or two sites remain, the rectangle is scan converted, else the rectangle is split into four quadrants, and the same procedure is applied recursively again. As a result, only near points with three incident edges the recursion progresses to pixel level.

The scan conversion of the rectangles is accelerated as follows. Fixed point arithmetic is used in the inner loops. Initially the values for $a(x - x_i)^2$ are calculated for $x = x_{min}, \ldots, x_{max}$. The calculation of $d_i^2$ per pixel thus reduces to the form $A \leftarrow B[x] + C$. Normalization of the lighting vector, the normal and multiplication by $I_s$ is accelerated via table look-up. An inverted table is calculated before rendering the image, which gives for each discrete grey level $I_s/(|\mathbf{n}(d)||\mathbf{l}|)$ the corresponding value of $d^2$. As starting point for the search in the table the grey level of the previous pixel is used.

With these measures we achieved a frame rate of 16 fps for a 400×400 image on a PC with a Pentium-II 350 MHz processor, without using a 3D graphics accelerator.

# 4 AUTOMATED SCANNING

In the previous section we focussed on user controlled setting of the weights. An alternative is to generate sequences of settings automatically, which sample the space of all possible combinations of presets. This is useful for exploration of the parameter space, searching for new and unexpected settings. We have been inspired by the Grand Tour method of Asimov [1]. Asimov considers the projection of multidimensional data as 2-dimensional scatterplots. His Grand Tour method produces a sequence of projections, such that the multidimensional data are viewed from many different directions. His main requirements are similar to ours:

- The sequence should be *dense*, i.e. the members of the sequence should be distributed uniformly over the high-dimensional space;
- The sequence should be simple to calculate;
- The sequence must be *continuous*, in the sense that the human observer has the impression of a smoothly varying setting;
- The method must have flexibility, such that the user can make various trade-offs for the preceding requirements.

Asimov gives several methods for generating sequences of projection planes. Similar methods can be used to generate sequences of settings for the weights. We have developed two methods: the bounce method and the random method.

## 4.1 Bounce Method

The idea of the bounce method is to generate a sequence of settings $\mathbf{u}_t$ along a line in the $m-1$ dimensional convex object defined by the presets. Each time the line hits a boundary of the object (i.e. a constraint $u_i \geq 0$ is violated), the line is reflected in a reverse direction. For the startpoint $\mathbf{u}_0$ we can pick any point, for instance the center of the convex object (set all $u_{0,i}$ to $1/m$), or a random point generated with the method described in the next section.

Let $\mathbf{v}$ be the desired change vector of $\mathbf{u}$ per step with length $s$. We initialize $\mathbf{v}$ as follows:

1. For $i := 1, \cdots, m$ set $v_i$ to the square root of the $i$'th prime;
2. $S := \sum v_i$; for $i := 1, \ldots, m$ do $v_i := v_i - S/m$;
3. $\mathbf{v} := s\mathbf{v}/|\mathbf{v}|$.

In the first step, we select independent components for the change vector, such that using a linear combination of them will never result on a return to the same point. In the second step we make sure that all components sum up to 0. As a result, if we add $\mathbf{v}$ to $\mathbf{u}$ the constraint that the components must sum up to 1 is maintained. Finally, we adjust the length of the vector.

Each new $\mathbf{u}_{t+1}$ is generated via $\mathbf{u}_{t+1} = \mathbf{u}_t + \mathbf{v}$. However, if we hit a boundary (a component $u_{t+1,k}$ becomes negative or larger than 1), we calculate the intersection point and continue with a new change vector $\mathbf{v}'$. For this we use

$$v_i' = \begin{cases} -v_k & \text{if } i = k \\ v_i + 2v_k/(m-1) & \text{otherwise} \end{cases}$$

This new change vector has its $k$'th component reversed, while its length remains equal to the length of the original change vector, and its components still add up to 0.

This method produces a fairly smooth sequence of settings. The user can watch the results and stop when an interesting result is shown. Also, the method can be used to generate interesting animations, to be shown when the computer is not used for other tasks.

### 4.2 Random Method

The random method produces sequences of random settings for the weights, distributed evenly over the space of all valid weight vectors. One way to realize this is to use the preceding algorithm with a large step size $s$, but these random vectors can also be generated directly.

A simple approach is to assign random values to all weights $u_i$, drawn from a uniform distribution over the unit interval, and to divide them afterwards by their sum. However, this does not give a uniform distribution: If $m \geq 2$, then low values of $u_i$ should have a higher probability to be selected. Specifically, for $u_k$ this probability is given by:

$$P(u_k \leq x) = 1 - \left(1 - \frac{\min(x, a)}{a}\right)^{k-1},$$

under the assumption that $u_{k+1}$ to $u_m$ have been generated, and that

$$a = 1 - \sum_{i=k+1}^{m} u_i.$$

This leads to the following algorithm:

1. $a := 1$;
2. For $k := m, \cdots, 2$ do
   (a) Draw $y$ randomly from $[0, 1]$;
   (b) $u_k := a\left[1 - (1-y)^{\frac{1}{k-1}}\right]$;
   (c) $a := a - u_k$;
3. $u_1 := a$.

In step 2(b) the inverse of the probability distribution was used to make the step from a sample $y$ from a uniform distribution to a value $u_k$. This algoritm generates a true random $\mathbf{u}$, where each possible valuation has an equal probability to be selected.

The user can press a stop button again if an interesting result is shown. A drawback of this method is that there is no relation between succeeding settings, hence the user has to be alert. An alternative is to put the generation of random settings under full user control: A button to produce a new random setting can be provided, which the user can press for new inspiration.

## 5 TRANSFORMATION

Weighting a number of presets can give mediocre results. Mixing dyes is a suitable metaphor: The result of mixing three or more dyes is often a dull brown. We can force settings for the weights (found either manually or automatically) into more sparkling versions by applying the following two steps:

1. Increase contrast: Raise high values and reduce low values for all $u_i$'s;
2. Normalize: Divide the new $u_i$'s by their sum.

The first step can be done in many ways. A simple, continuous method is to raise all $u_i$'s to a power $r$, where $r$ is for instance 2, 3 or 4. Small values will become very small, values close to 1 will remain so.

An alternative is to use a discrete method: Set the $N$ smallest presets to 0. As a result, only up to $m - N$ weights will be non-zero, i.e. only $m - N$ presets will be active. However, this gives rise to discontinuities: Even upon small changes of the weights, the modified values will jump from 0 to some non-zero value and back, and hence also the interpolated setting will change discontinuously. A solution to this is to modify also the active weights. From each we substract the value $d$ of the largest non-active weight. When the value of the smallest active weight approaches $d$, its modified value is continuously diminished to 0.

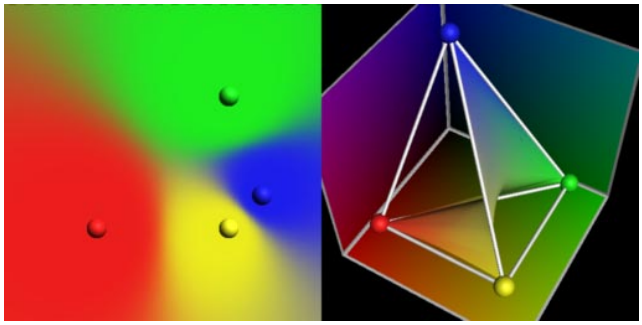

Figure 7: Raising $u_i$'s to the third power

Figure 7 shows the effect on the controller for our color example. Here all $u_i$'s are raised to the third power and normalized. Each control point has a stronger influence on the surrounding area. In parameter space, the surface approaches edges and faces of the tetrahedron more closely. Again, this generalizes to higher dimensions. The current setting will approach the boundary of the $m$-dimensional object more closely, or in other words, the setting is a mix of only a few presets.

## 6 APPLICATIONS

We have developed and tested our methods for two applications now, morphing of drawings and control of a sound synthesizer, which are presented in this section. Furthermore, a general pattern for enhancing existing applications with preset blending methods is presented.

### 6.1 Morphing

Our first application was inspired by one of the earliest examples of computer generated art: the image "Running Cola is Africa", produced by the Computer Technique Group from Tokyo. It is a black-and-white morphing sequence showing the transformation of a runner into a Coca-Cola bottle which then morphed into the map of Africa [12].

In our system, each preset consists of a colored polygon with an arbitrary number of points. These presets can be edited interactively by the user. If the number of points varies per preset, before summing them we determine the highest number of points used, and resample the polygons along the boundary. The result is shown as a smooth B-spline curve that approximates the points.

Figure 8 shows a result. We have drawn eight animals here, each defined by forty points and a color, i.e. each animal is a sample point

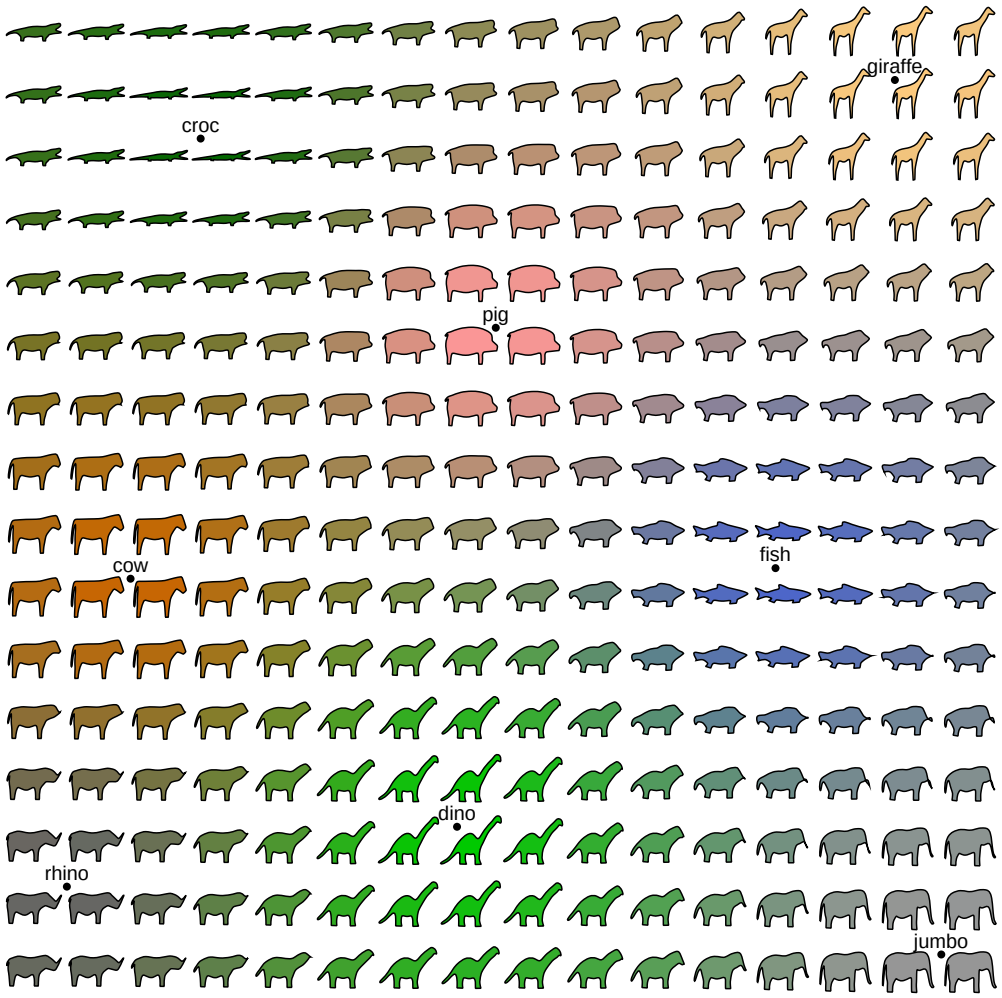


Figure 8: Morphing animals

in a parameter space with 83 dimensions. One possible distribution of these presets over the surface of our preset controller is shown. Many variations on animals can be generated in real-time by dragging the cross of the preset controller, and by dragging and editing the presets. Each animal displayed here shows the result of dragging the cross of the preset controller to the position of that animal.

A related type of application that we foresee is the use of the preset controller for the definition of collections of varying items. Many recent movies are populated by computer generated flocks of dinosaurs, armies of robots, and swarms of space creatures. At first sight, the effect is often highly impressive, but at second sight it is sometimes disappointing that each individual creature is a clone of the same model, leading to a mechanical and unrealistic result. More variation is needed here.

One way to achieve this is obviously to use a parametric model and draw random samples from distributions over the individual parameters. The use of our framework gives improved control. Different typical creatures can be defined, each with its own characteristic setting for parameters such as dimensions, color, proportions. These creatures are used as presets and positioned at the controller. The animator can define the amount of blending (see section 5), the family relations by the relative position of the presets, and the quantity of each type (to be judged by the size of the corresponding area of the Voronoi diagram). Finally, the area of the preset controller is sampled (randomly or regularly), where each sample defines the parameters for an individual creature. In other words, we think that the controller can be used to define a complex multi-dimensional distribution function in an easy and intuitive way.

## 6.2 Sound synthesis

The discovery that *frequency modulation* (FM) can be used as an extremely efficient means of generating musically useful complex waveforms [4] stands as one of the most important discoveries in the field of computer music [8]. FM-synthesis is used in many soundcards and musical instruments, such as the famous Yamaha DX-7.

The principle of FM synthesis is remarkably simple. The basic equation is:

$$y(t) = A\sin(2\pi f_c t + I\sin 2\pi f_m t).$$

The time varying signal $y(t)$ is produced by an oscillator, the carrier, which frequency $f_c$ is varied by another oscillator, the modulator. The frequency $f_m$ of the modulator is usually an integral multiple of $f_c$, which gives a rich, harmonic spectrum. The modulation index $I$ determines the depth of the modulation, while $A$ is the peak amplitude of $y(t)$. This is the simplest set-up, consisting of two os-

cillators, called *operators*. More complex waveshapes can be generated by using multiple carriers and modulators.

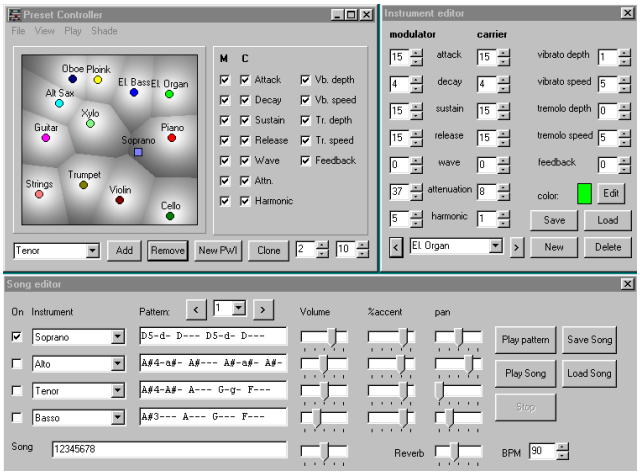


Figure 9: User interface synthesizer

We have developed a small software synthesizer based on FM-synthesis, using two operators. Each operator has the following parameters: overall amplitude, envelope (Attack, Decay, Sustain, Release – ADSR), harmonic (multiplication factor for frequency) and wave shape (sin, square, etc.). Furthermore, frequency and depth for low frequency modulation of pitch and amplitude, i.e. vibrato and tremolo, can be used. With these 18 parameters a wealth of different instruments can be generated, but, also typical for FM-synthesis, it is hard to predict what the effect of a parameter change will be. Hence we used the framework presented here to enable the musician to experiment with different settings and to generate different instruments.

Figure 9 shows the user interface. Instruments can be edited directly with the instrument editor, shown upper right; preset weighted instruments can be controlled with the preset controller in the upper left window. At the bottom a simple sequencer is shown. The user can define and play songs, consisting of patterns. Each pattern consists of four tracks, each with its own voice. This voice can be one of the preset instruments or a weighted average of preset instruments. For each note that is played the weighted averages of presets are evaluated anew, hence the user can evaluate different settings for them in real time.

Figure 10 shows the preset controller enlarged, with more presentation options enabled: color, polished markers, a different font. This shows that the preset controller can be made more attractive for recreational and educational applications.

Application in practice by a small group of amateur musicians revealed a number of interesting aspects. With the preset controller it is easy to generate different sounds. The applicability was readily recognized. A typical situation that occurs when playing a synthesizer is that for instance six different presets for bass guitars are available, where none of them completely satisfies the needs. With the preset controller a nice combination can be selected.

The results were not always predictable though. Some of the parameters are ordinal (harmonic) or nominal (wave type), leading to abrupt changes in the timbral characteristics. This inspired us to develop the shaded Voronoi diagram, which gives the user more feedback on where to expect such changes.

Most interesting and surprising however was that the preset controller was also very effective for more mundane tasks. We had added for each parameter a checkbox (right of the preset controller),

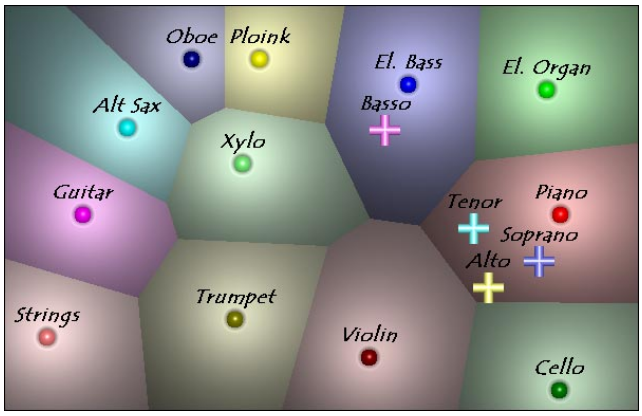


Figure 10: Preset controller for synthesizer

which the user could use to select whether this parameter should be interpolated or keep its value. In practice this was very useful to combine various aspects of instruments. For instance, we can start with a xylophone, freeze the dynamical ADSR properties, and next move the cross to a piano to pick up the timbral characteristics. With a more traditional approach this is much harder, requiring for instance writing down the settings for parameters separately.

Another simplified task was selection of the instrument per track. The sequencer supports this in a standard way via a pull-down menu, commercial synthesizers usually require a numerical selection. Here we often used preset weighted instruments for each track, and selected instruments by dragging the crosses. A typical use case now is that we want to use the same instrument for the first three tracks (soprano to alto). This can be easily realized by positioning the three corresponding crosses close to each other and moving the desired instrument icon close to these crosses. Also, multiple icons can be selected and dragged simultaneously over the area of the controller. As a result, the user can explore many different orchestrations in real time, while interacting with the system in a natural and predictable way.

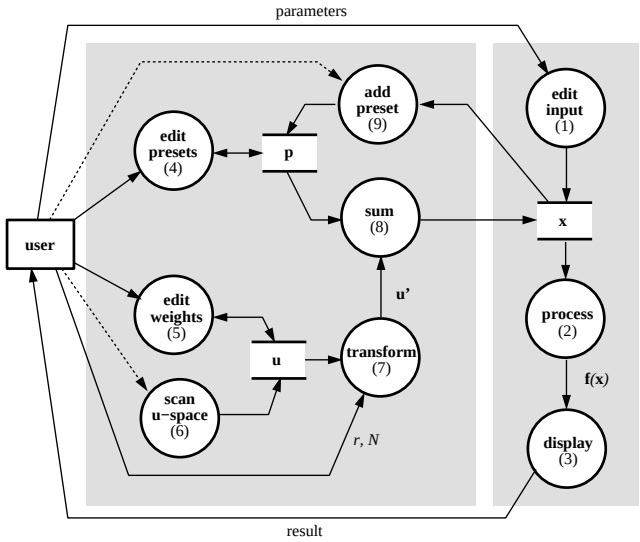


Figure 11: Preset based interaction

### 6.3 In general

Many other applications can be enhanced in a similar way via preset based interaction, hence we describe it in a general way. Figure 11 shows the structure of an enhanced system as a data flow diagram. A standard interactive system is shown in the grey box on the right: The user prepares input (1), this is processed (2), and displayed (3). The user observes the results and readjusts the parameters. The grey box on the left contains a preset based navigation module. The user (or his expert colleagues) edit and manage a set of presets (4). The user edits weights for each preset (5) via a preset controller or, as an alternative, he can start the automatic scanning of the **u**-space (6). Possibly, the weights are transformed (7), and finally the presets are weighted and summed (8), leading to a new input **x**. If this leads to an interesting result, the user can add this setting to the presets (9), and use it as a starting point for a new preset.

## 7 DISCUSSION

We have presented a framework for the exploration of and navigation through high-dimensional parameter spaces. The central concept is the definition of input as the weighted sum of a number of presets, thereby bridging the gap between the expert mode of setting all individual parameters and the novice user mode of selecting presets. Various methods for the control of the weights were presented. The preset controller is the most promising of these: We found that it is very natural and easy to use.

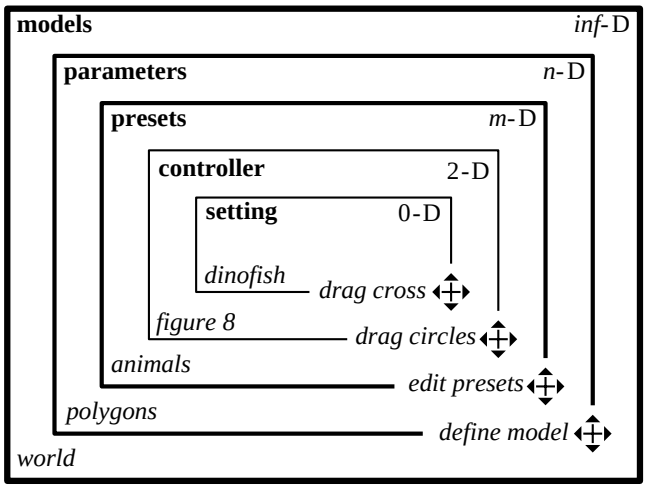


Figure 12: Framework

A schematic representation of our framework as a sequence of nested spaces is shown in figure 12. Each box represents a selection from the space of its surrounding box, which selection can be modified by the user. Going from the outside inwards, we see that the level of abstraction, the dimensionality of the space, and the level of expertise needed decrease.

We see several possibilities for further research. So far we allowed only non-negative weights. Negative weights lead to extrapolation instead of interpolation, and can give unexpected results, outside the scope of the presets. Negative weights can easily be specified via sliders, but we have not found yet an intuitive way to define them with the preset controller.

Another topic is automated segmentation. Given a set of presets, can we automatically determine groups of related parameters, such that we can control each group individually?

Most important however, is further application of the framework in practice. We found application to be highly stimulating, not only because the basic concept fulfilled our expectations, but also because each new application gave rise to further extensions and new ideas. One example is that the discrete transitions in the synthesizer case inspired us to provide additional cues via the shaded Voronoi diagram.

We are especially eager to apply the framework for systems where the relation between input and output is less straightforward than in the examples shown. We expect that our methods will be very useful here: In the best case they can enable a user to perform inverse modelling. For instance, consider a design system where the designer has to enter dimensions and material properties. The system returns functional characteristics such as cost, weight, and strength of the design. Suppose now that various presets are available, where each is optimal with respect to some functional characteristic. We expect that the preset controller will support the designer directly in making a trade-off between the various functional characteristics.

Finally, we hope that this paper has the same function as our preset controller: an invitation to play and experiment!

## References

[1] D. Asimov. The grand tour: A tool for viewing multidimensional data. *SIAM J. Sci. Stat. Comput.*, 6(1):128–143, jan 1985.

[2] F. Aurenhammer. Voronoi diagrams: A survey of a fundamental geometric data structure. *ACM Computing Surveys*, 23(3):345–405, sep 1991.

[3] W. Buxton, S. Patel, W. Reeves, and R. Baecker. Objed and the design of timbral resources. *Computer Music Journal*, 6(2), 1982.

[4] J. Chowning. The synthesis of complex audio spectra by means of frequency modulation. *J. Aud. Eng. Soc.*, 21(7):526–534, 1973.

[5] T.F. Cox and M.A.A. Cox. *Multidimensional Scaling*. Chapman & Hall, London, 1995.

[6] G.W. Fitzmaurice, H. Ishii, and W. Buxton. Bricks: Laying the foundations for graspable user interfaces. In *Proceedings CHI'95*, pages 442–449, 1995.

[7] S.N. Lam. Spatial interpolation methods: A review. *The American Cartographer*, 10(2):129–149, 1983.

[8] F.R. Moore. *Elements of Computer Music*. Prentice Hall, New Jersey, 1990.

[9] J.D. Mulder and J.J. van Wijk. 3d computational steering with parametrized geometric objects. In D. Silver G.M. Nielson, editor, *Proceedings Visualization'95*, pages 304–311. IEEE Computer Society Press, 1995.

[10] C.B. Phillips and N.I. Badler. Interactive behaviors for bipedal articulated figures. *Computer Graphics*, 25(4):359–362, 1991.

[11] M. Rauterberg, M. Bichsel, M. Fjeld, U. Leonhardt, H. Krueger, and M. Meier. Build-it: A planning tool for construction and design. In *CHI'98 Conference Companion*, 1998.

[12] J. Reichardt. *Cybernetic Serendipity: The Computer and the Arts*. Praeger, New York, 1968.

[13] D. Shepard. A two-dimensional interpolating function for irregularly spaced data. In *Proc. of the 23rd ACM national Conference*, pages 517–524, 1968.

[14] M.G. Voronoi. Nouvelles applications des parametres continus a la theorie des formes quadratiques. *J. Reine Angew. Math.*, 134:198–287, 1908.