# Visual Analysis of Popping in Progressive Visualization: Supplementary Material

E. Waterink, J. Kosinka [ID], and S. Frey [ID]

Bernoulli Institute, University of Groningen, the Netherlands

## 1. Adaptive Image Space Sampler: Details

While ray tracing is capable of generating realistic and detailed images, this comes with a great computational cost, especially for high-resolution images. For progressive visualization, there is a budget for how many pixels can be sampled each iteration. After this, a full image for presentation is reconstructed from the obtained samples via interpolation, and where to sample next is determined in a stochastic manner based on the color variation of the pixels. Initially, the image is very rough with just few samples, but with every new sample the image becomes more accurate, until all pixels are sampled and the final image is obtained. Note that in our experiments, we use a simple simulation of ray tracing: instead of actually obtaining pixel values by sending a ray through the pixels, it just draws values from a provided image. This simulates an iterative refinement process on which we can perform experiments and apply visualizations. In this use case, the pixels are the individual elements $\mathbf{e}$ and the (reconstructed) images the spatial objects $\mathbf{O}$.

We outline the process of the adaptive image space sampler and discuss the sampling and reconstruction scheme in detail as well as some optimizations thereof.

### 1.1. Outline

The adaptive image space sampler takes in an image $I$, which is also the final result of the process. We define $S$ as the set that contains all sampled pixels, and its complementary set $U$ of non-sampled pixels. Let $P$ be the set of all pixels in $I$. We then have that

$$S \cup U = P, \quad S \cap U = \emptyset, \tag{1.1}$$

i.e. $S$ and $U$ are disjoint. We denote with $X_n$ the state of $X$ at time step $n \geq 0$. From (1.1) it follows that $S_n = P \setminus U_n$ and $U_n = P \setminus S_n$, so we can derive them from each other at any step.

Initially, $S_0 = \emptyset$ and $U_0 = P$ as no pixels are sampled yet. Then, for $n \geq 1$ we *sample* pixels by taking them from $U_{n-1}$ and adding them to $S_{n-1}$:

$$S_n = S_{n-1} \cup \mathcal{S}(U_{n-1}), \tag{1.2}$$

where $\mathcal{S}$ is the sampling function. Equivalently, we have that $U_n = U_{n-1} \setminus \mathcal{S}(U_{n-1})$. In the iterative process, as more pixels are sampled, $S_n$ will grow while $U_n$ will shrink, and ultimately, $S_{n_{max}} = P$ and $U_{n_{max}} = \emptyset$.

---

**Algorithm 1:** Iterative refinement process of the adaptive image space sampler with sampling scheme $\mathcal{S}$ and reconstruction scheme $\mathcal{R}$.

**input** : Image $I$
**output**: Image $I$

1   $S_0 := \emptyset$
2   $U_0 := P$      // $P$ is the set of all pixels in $I$
3   **for** $n \leftarrow 1$ **to** $n_{max}$ **do**
4     $S^* = \mathcal{S}(U_{n-1})$
5     $S_n = S_{n-1} \cup S^*$
6     $U_n = U_{n-1} \setminus S^*$
7     $I = \mathcal{R}(S_n)$

---

The sample set $S_n$ is then used to *reconstruct* an approximation $R_n$ of $I$:

$$R_n = \mathcal{R}(S_n), \tag{1.3}$$

where $\mathcal{R}$ is the reconstruction function. The values of the pixels in $S_n$ are known, and are used to estimate the values of the pixels in $U_n$ by means of multivariate interpolation. This creates an intermediate result of the rendering process. The sampling and reconstruction is then repeated until the process reaches $n_{max}$ and we have that $R_{n_{max}} = \mathcal{R}(P) = I$.

The iterative refinement process of the adaptive image space sampler is summarized in Algorithm 1. Two crucial parameters are the *number of nearest neighbors* and the *power parameter*. The former is used in the sampling scheme (Section 1.2) and indicates how many nearest neighbors (in $S_{n-1}$) are to be considered for determining the color variation and deciding which pixels to sample next (from $U_{n-1}$). The latter is used in the reconstruction scheme (Section 1.3) in the inverse distance weighting method for multivariate interpolation, where it controls how much the distance (inversely) affects the weight.

### 1.2. Sampling

In each iteration, pixels are sampled from $U$ and added to $S$. As $U$ can only be sampled a finite number of times, $n_{max}$ is known. In the adaptive image space sampler, we sample a constant number of

pixels $s$ at each iteration. Alternatively, we can specify a percentage $s_\%$ of the total number $|P|$ of pixels, i.e. $s = \lceil \frac{s_\%}{100}|P| \rceil$. In either case, $n_{\max} = \lceil \frac{|P|}{s} \rceil$. Note that if $|P|$ is not a multiple of $s$, the sample size of the final iteration is $\min(s, |U_{n_{\max}-1}|)$.

The selection of the initial sample is random as nothing is known about the image yet. While the next samples could be random as well, we can do better by an informed way of sampling by using the already sampled pixels in $S_{n-1}$. The general idea is that if a pixel is reconstructed from very different sample values, this indicates that there is a lot of variation and it makes sense to sample there next. Conversely, pixels reconstructed from similar sample values are less interesting. Hence, we compute a metric $M(\mathbf{p})$ for $\mathbf{p} \in U_{n-1}$ using the pixels in $S_{n-1}$. For practical purposes, one does not consider all sample values, but just the closest ones; that is, its $N$ nearest neighbors. We define $N(\mathbf{p})$ as the $N$ nearest neighbors (that is, nearest pixels in $S_{n-1}$) of $\mathbf{p}$ based on Euclidean distance. As we are working with colors, the variation can be computed as the (Euclidean) color distance between all the pairs in $N(\mathbf{p})$ and summing them:

$$M(\mathbf{p}) = \sum_{\mathbf{q} \in N(\mathbf{p})} \sum_{\mathbf{r} \in N(\mathbf{p})} d(V(\mathbf{q}), V(\mathbf{r})), \qquad (1.4)$$

where $d(\mathbf{c}_1, \mathbf{c}_2) = \sqrt{(A_2 - A_1)^2 + (B_2 - B_1)^2 + (C_2 - C_1)^2}$ and $V(\mathbf{p})$ is the $(A, B, C)$ color value of $\mathbf{p}$, where ABC is any color space with three channels, e.g. RGB or LCH. The metric $M$ can then be used in two ways to determine which pixels to sample: deterministic or stochastic.

**Deterministic** In the deterministic approach we use the distance metrics directly. That is, the $s$ pixels with the highest color variation are selected. That way, pixels in regions with high color variation are always sampled before those in low color variation regions.

**Stochastic** Instead of using differences directly for selecting new samples, in the stochastic approach they are used as weights for random selection. That way, each pixel has a chance of being sampled, albeit not the same one. Pixels with a higher color variation are more likely to be sampled. We convert the distance metrics to probabilities by $\frac{M}{\sum_{\mathbf{q} \in U_{n-1}} M(\mathbf{q})}$ and use these for $\mathcal{S}$.

We then obtain the values of the newly sampled pixels. In a real-world application, such as volume rendering, we shoot rays through the volume for the sampled pixels, followed by interpolation and classification: $V(\mathbf{p}) = \mathcal{F}(t, T), \quad t : [0, T] \to \mathbb{R}$, where $\mathcal{F}$ is the so-called ray function. As input, $\mathcal{F}$ has a scalar-valued function $t : \mathbb{R}_+ \to \mathbb{R}$ defined on $[0, T]$, where $T$ is the length of the ray segment. As output, $\mathcal{F}$ produces a color. In other words, $\mathcal{F}$ synthesizes the RGB color of the pixel $\mathbf{p}$ from the scalar values along the ray corresponding to $\mathbf{p}$ [Tel15]. In ray tracer applications, the color is obtained from the intersected objects and their reflection/refraction. However, as mentioned before, in the adaptive image space sampler we assume $I$ is known such that we can simply take the colors directly: $V(\mathbf{p}) = I(\mathbf{p})$.

## 1.3. Reconstruction

The purpose of reconstruction is to produce an estimation of the final result using current knowledge. Given sample set $S_n$, $\mathcal{R}$ creates an approximation of $I$ by means of multivariate interpolation. One such method is the Inverse Distance Weighting (IDW), which is a deterministic method for multivariate interpolation with a known scattered set of points. The assigned values to unknown points are calculated with a weighted average of the values available at the known points:

$$\mathcal{R}(\mathbf{p}) = \begin{cases} \frac{\sum_{\mathbf{q} \in S_n} w_q(\mathbf{p}) V(\mathbf{q})}{\sum_{\mathbf{q} \in S_n} w_q(\mathbf{p})}, & \text{if } \mathbf{p} \notin S_n, \\ V(\mathbf{p}) & \text{if } \mathbf{p} \in S_n, \end{cases} \qquad (1.5)$$

where $w_{\mathbf{q}}(\mathbf{p}) = \frac{1}{d(\mathbf{p}, \mathbf{q})^p}$ is a simple IDW weighting function, $d(\mathbf{p}, \mathbf{q}) = \sqrt{(q_x - p_x)^2 + (q_y - p_y)^2}$, i.e. the Euclidean distance from the known pixel $\mathbf{p}$ to the unknown pixel $\mathbf{q}$, and $p$ a positive real number called the power parameter. Here, the weight decreases as the distance increases from the interpolated points. Greater values of $p$ assign greater influence to values closest to the interpolated point, with the result turning into a mosaic of tiles with nearly constant interpolated values for large values of $p$. For two dimensions, power parameters $p \le 2$ cause the interpolated values to be dominated by points far away.

When working with RGB images, interpolating the $R$, $G$ and $B$ components independently offers no guarantee on the hue of the intermediate colors. A new hue appears because the RGB space does not capture how humans perceive colors very well. So instead, we use the LCH color space for color interpolation, because equidistant colors in the LCH space are also perceived as equidistant [Zuc16].

## 1.4. Optimization

Looking at the mathematical functions in Section 1.2 and Section 1.3 just above, we note that some can be optimized. For the sampling, computing the nearest neighbors requires computing the distance to all currently sampled pixels and sorting them. However, once we computed the nearest neighbors $N(\mathbf{p}) \in S_n$, we do not have to consider the pixels in $S_n \setminus N(\mathbf{p})$ anymore, as we know they are always further away. Hence, we only have to check if the newly sampled pixels $\mathcal{S}_n$ are closer than the previous nearest neighbors $N_{n-1}(\mathbf{p})$, i.e. we check in $N_{n-1}(\mathbf{p}) \cup \mathcal{S}_n$. So, for every pixel we store its $N$ nearest neighbors.

Moreover, computing $M(\mathbf{p})$ as in (1.4) essentially sums over a distance matrix, which is symmetric and its main diagonal contains only zeros. Hence, we can instead sum the values in the upper triangle, which would yield half the value. This does not change the order of $M$ or the probabilities computed from it. We use SciPy's `distance.pdist` function to create a condensed matrix which is summed.

For the reconstruction, we also have to compute the distance to all currently sampled pixels. In the naive approach (1.5), the IDW weighting functions for $\mathbf{p} \in U_n$ are recomputed for $\mathbf{p} \in U_m, m > n$.

**Figure 1:** *Input images (left to right, top to bottom): volume rendering of a human head CT scan using a colored maximum intensity projection, a photo-realistic scene with translucent and reflective objects created by a ray-tracing application, a colorful photo of some fruit, a still from the animated movie Rango [Ver11].*

However, we have that:

$$
\begin{aligned}
\mathcal{R}_n(\mathbf{p}) &= \frac{\sum_{\mathbf{q} \in S_n} w_\mathbf{q}(\mathbf{p}) V(\mathbf{p})}{\sum_{\mathbf{q} \in S_n} w_\mathbf{q}(\mathbf{p})} \\
&= \frac{\sum_{\mathbf{q} \in S_{n-1} \cup \mathcal{S}_n} w_\mathbf{q}(\mathbf{p}) V(\mathbf{p})}{\sum_{\mathbf{q} \in S_{n-1} \cup \mathcal{S}_n} w_\mathbf{q}(\mathbf{p})} \\
&= \frac{\sum_{\mathbf{q} \in S_{n-1}} w_\mathbf{q}(\mathbf{p}) V(\mathbf{p}) + \sum_{\mathbf{q} \in \mathcal{S}_n} w_\mathbf{q}(\mathbf{p}) V(\mathbf{p})}{\sum_{\mathbf{q} \in S_{n-1}} w_\mathbf{q}(\mathbf{p}) + \sum_{\mathbf{q} \in \mathcal{S}_n} w_\mathbf{q}(\mathbf{p})} \\
&= \frac{a_{n-1}(\mathbf{p}) R_{n-1}(\mathbf{p}) + \sum_{\mathbf{q} \in \mathcal{S}_n} w_\mathbf{q}(\mathbf{p}) V(\mathbf{p})}{a_{n-1}(\mathbf{p}) + \sum_{\mathbf{q} \in \mathcal{S}_n} w_\mathbf{q}(\mathbf{p})},
\end{aligned}
\tag{1.6}
$$

where $a_n$ is the accumulated norm at step $n$ defined as $a_n = a_{n-1} + \sum_{\mathbf{q} \in S_n} w_\mathbf{q}(\mathbf{p})$ with $a_0 = 0$. Then, we use $a$ together with the value in the previous reconstructed image and only compute the distances to the newly sampled pixels.

Note that these optimizations are memory versus speed trade-offs; they do not change the results.

### 1.5. Popping Threshold

For a perceptually meaningful threshold the reconstructed images are transformed to the CIELAB color space [Sch16], where we compute $d = \Delta E^* = \sqrt{(L_q^* - L_p^*)^2 + (a_q^* - a_p^*)^2 + (b_q^* - b_p^*)^2}$. On a typical scale, the $\Delta E^*$ value ranges from 0 to 100. A general guide for $\Delta E^*$ values and their perception is shown in Table 1.
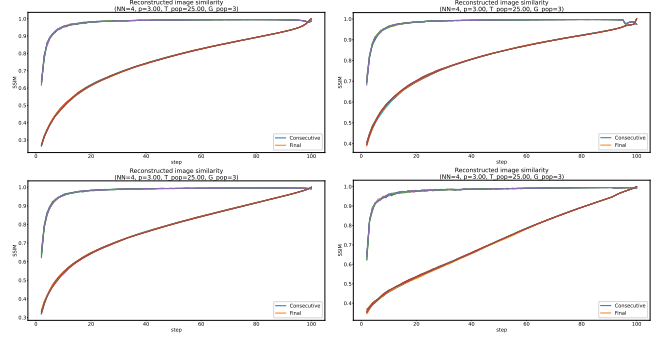


**Figure 2:** *Structural similarity of reconstructed images against their predecessor (consecutive) and the final image. The plots correspond to the input images in Figure 1.*

## 2. Adaptive Image Space Sampling: Additional Results

The first input image in Figure 1 was already discussed in the main paper, but is shown here for comparison. The second is a photo-realistic scene with translucent and reflective objects created by a ray-tracing application. The third image is a photo of some fruit, which is not actually created by a ray-tracing application, but is merely a colorful image to test the visualizations on. The fourth image is a frame of the 2011 American computer-animated Western comedy film Rango [Ver11]. This particular scene has a simple background and contains edges mainly around and on the character.

The images have a resolution of $256 \times 256$, which is comparably small by today's standards but suffices to assess the properties of the method for a certain type of image. Unless described otherwise, these parameter settings were used, yielding good results in our experiments: the sample size is set to 1% (i.e., $\lceil 0.01 \cdot 256 \cdot 256 \rceil = 655$ pixels for $n_{\max} = \lceil \frac{256 \cdot 256}{655} \rceil = 100$ iterations), the number of nearest neighbors is set to 4, and the power parameter to 3.

**Refinement Characteristics.** The similarity plots in Figure 2 show the same behaviour for the first three cases. The fourth, however, shows a nearly linear increase until the final iteration for the similarity with respect to the final image, meaning that no structures are unexpectedly revealed throughout.

For the reconstructed pixel difference (Figure 4), all plots show different characteristics. The first is quite standard: linear decrease with few surprises (sudden increases). The second is similar, but does have several sudden increases, as well as popping near the final iteration after a period of no popping. The third has high dis-

**Table 1:** *General guide for $\Delta E^*$ values and their perception [Sch16].*

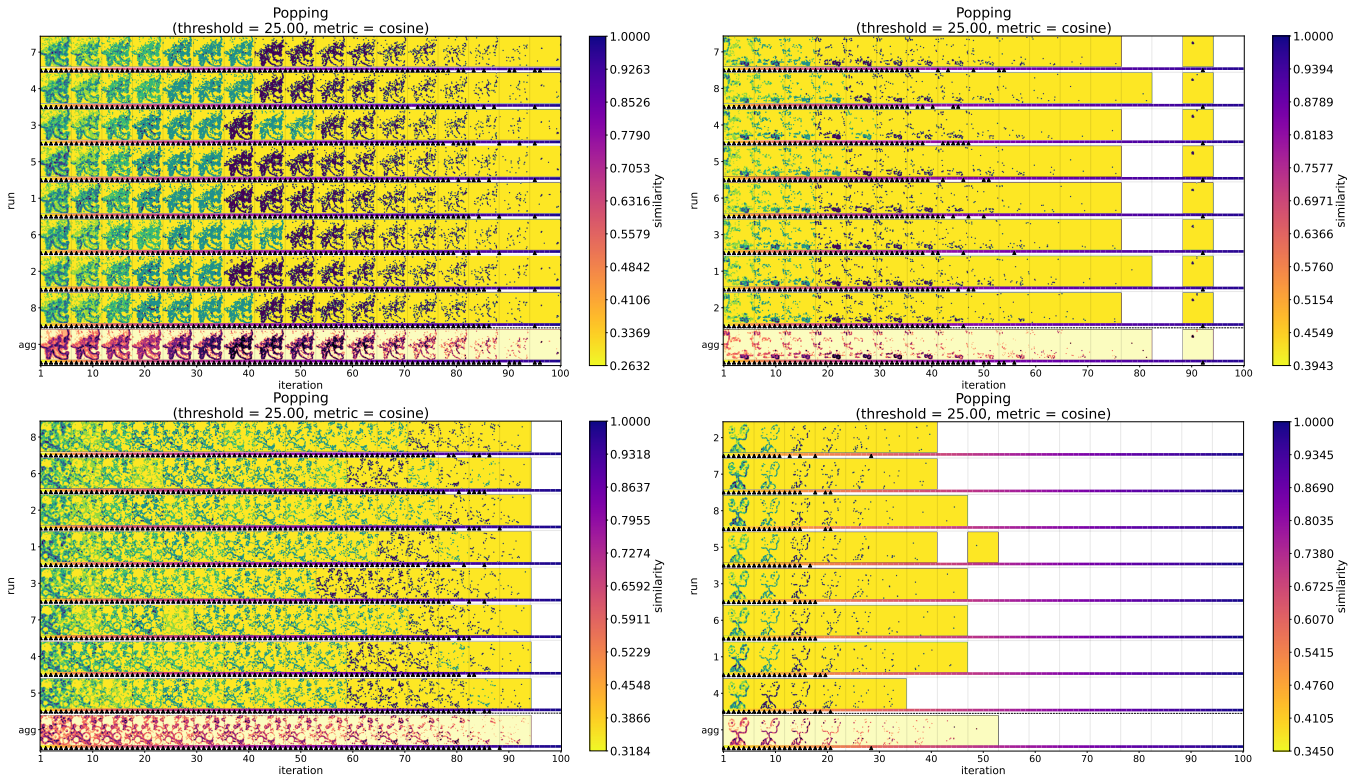| $\Delta E^*$ | Perception |
|---|---|
| $\leq 1.0$ | Not perceptible by human eyes. |
| $1 - 2$ | Perceptible through close observation. |
| $2 - 10$ | Perceptible at glance. |
| $11 - 49$ | Colors are more similar than opposite. |
| $100$ | Colors are exact opposite. |

**Figure 3:** *Multi-run comparison of the adaptive image space sampler. It is applied to 8 different runs of 100 iterations with interval size b = 6 using the cosine distance for reordering.*
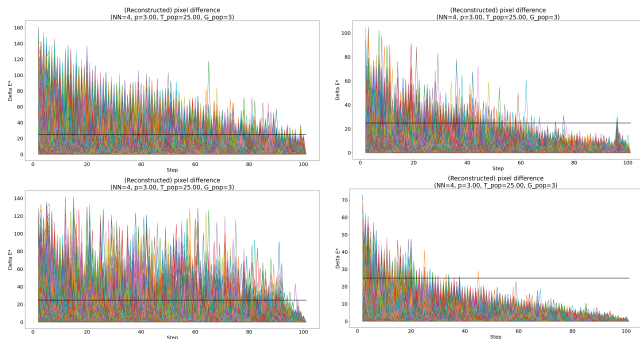


**Figure 4:** *Plot of the color distance between two consecutive reconstructed images for every pixel; cf. Figure 2.*

tances until the end and, consequently, many popping artifacts. The fourth decreases relatively quickly below the threshold bar.

**Popping Detection.** These results are run with the same popping detection and parameter values as in the main paper, but the popping threshold is lowered to 25 (see also Section 1.5).

**Multi-Run Overview Visualization of Popping Artifacts.** The multi-run overview visualization (Figure 3) shows that many poppings occurred until the final iterations. The second pops until

halfway, stops and suddenly pops again (in the same location for the different runs). The third shows similar behaviour to the first. In the fourth many poppings occur in the first few iterations, but it quickly decreases in frequency and stops after about 30 iterations. For all four cases the structures in the images are clearly visible.

**Positional Analysis: Popping Distribution and Sampling Time.** As the threshold is lower, many more popping artifacts occurred (compared to the case with $T_{pop} = 38.00$ in the main paper) in the first image. Little popping happened in the background regions. However, if we set the power parameter $\leq 2$ the background reconstructed colors would be dominated by colors far away, and so they could in fact pop once sampled. The second experiences less popping throughout, but pops late due to the flat color regions. The third popped all over, because the image is very colorful. The fourth shows early popping which occurred only along the character's edges (Figure 5).

The time step is mapped to a color and overlaid on a gray-scale version of the reconstructed image:

$$W_n(\mathbf{p}) = \begin{cases} \texttt{cmap}(\text{when}(\mathbf{p})) + \texttt{rgb2gray}(R_n(\mathbf{p})) & \text{if } \mathbf{p} \in S_n, \\ \texttt{rgb2gray}(R_n(\mathbf{p})) & \text{if } \mathbf{p} \notin S_n, \end{cases}$$

where when$(\mathbf{p}) = n$ stores when pixel $\mathbf{p}$ was sampled, $W$ is the image created from it, `cmap` is the color map, and `rgb2gray` converts an image with RGB channels into an image with a single grayscale channel. Note that when$(\mathbf{p})$ is only defined for pixels
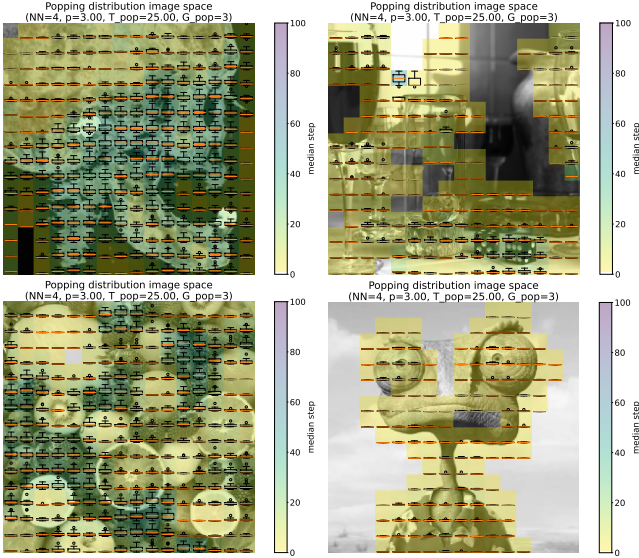
**Figure 5:** *Temporal distribution of popping artifacts on a $16 \times 16$ grid ($T_{\text{pop}} = 25$, $G_{\text{pop}} = 3$).*
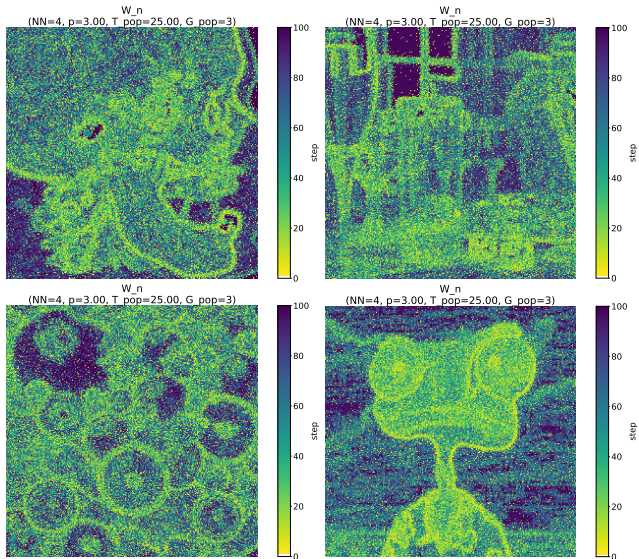


**Figure 6:** *Final visualization showing when pixels are sampled with a color bar (without gray-scale).*

that are sampled, i.e., when $\mathbf{p} \in S_n$. Figure 6 shows that pixels are sampled first at the edges of objects, and that flat regions are sampled last for all four cases.

## 3. Tile Grid Generation: Details

The aim of tile grid generation is to convey an overview on a large collection of objects [SG14, FDH*15, QSST10]. The general concept is to place images that are similar close to each other in order to get an overview and compare what similar types there are.

The generation of tile grids is a complex problem due to the in-

---

**Algorithm 2:** Outline of progressive tile grid generation.

**Input** : Collection of objects
**Output:** Grid layout **G**

1  $\mathbf{G}_0 :=$ random grid placement
2  **for** $n \leftarrow 1$ **to** $n_{\max}$ **do**
3  $\quad$ $\mathbf{G}_n :=$ exchange images of $\mathbf{G}_{n-1}$ in groups of size $k$
4  $\quad$ **if** *exchange is not beneficial* **then**
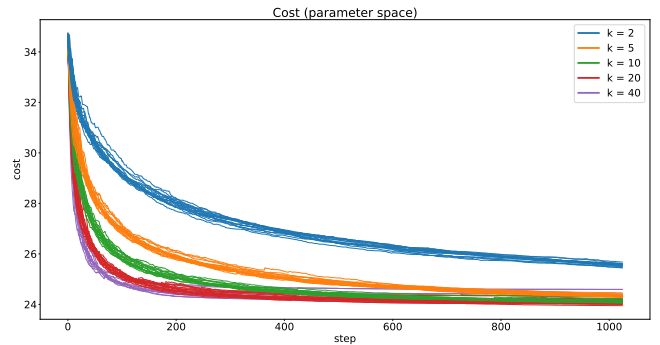5  $\quad\quad$ $\mathbf{G}_n := \mathbf{G}_{n-1}$



**Figure 7:** *Cost progression for different values of k.*

terdependencies between individual elements. The algorithm considered here is a current research prototype that addresses this task with an iterative, stochastic process. In each iteration, it partitions all images into random groups of size $k$, and then within each group solves a combinatorial optimization problem: it reorganizes the tiles such that the images are as similar to their neighbors as possible in total (Algorithm 2). The cost function is comprised of distances to neighbors and the optimization goal is to minimize this cost during the iterations.

This kind of process practically never really converges to a point with no further changes, but merely the rate of improvement slows down; and it is not inherently clear when it is slowing down. The analysis goal is to identify when and how often larger changes occur, how stable this is across runs, and what the impact of group size $k$ is on this behavior. This can further indicate when a run can be deemed to be sufficiently stable.

In our experiments, we placed 1024 images from the Caltech 101 database [LAR03] (based on generated feature vectors from a neural classifier). We considered 16 different runs for 1024 iterations in our experiments. Note that in this use case, absolute positions are not meaningful, only relative positioning between tiles is. We thus omit a location-based analysis here.

**Parameter Study.** Besides popping, parameter $k$ influences, of course, the cost value and the computation time. We observe that higher values of $k$ yield lower cost values (Figure 7), but require more computation time (Figure 8).

## References

[FDH*15] FRIED O., DIVERDI S., HALBER M., SIZIKOVA E., FINKELSTEIN A.: Isomatch: Creating informative grid layouts. *Com-
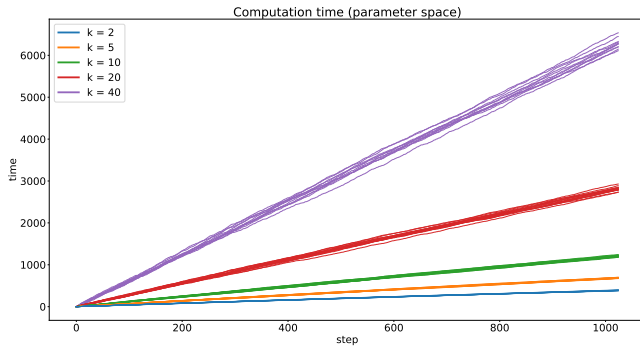
**Figure 8:** *Computation time for different values of k.*

*puter Graphics Forum 34*, 2 (2015), 155–166. `doi:10.1111/cgf.12549`. 5

[LAR03] LI F.-F., ANDREETTO M., , RANZATO M. A.: Caltech 101, 2003. URL: `http://www.vision.caltech.edu/Image_Datasets/Caltech101/`. 5

[QSST10] QUADRIANTO N., SMOLA A. J., SONG L., TUYTELAARS T.: Kernelized sorting. *IEEE Transactions on Pattern Analysis and Machine Intelligence 32*, 10 (Oct 2010), 1809–1821. `doi:10.1109/TPAMI.2009.184`. 5

[Sch16] SCHUESSLER Z.: Delta e 101, 2016. URL: `http://zschuessler.github.io/DeltaE/learn/`. 3

[SG14] STRONG G., GONG M.: Self-sorting map: An efficient algorithm for presenting multimedia data in structured layouts. *IEEE Transactions on Multimedia 16*, 4 (June 2014), 1045–1058. `doi:10.1109/TMM.2014.2306183`. 5

[Tel15] TELEA A. C.: *Data Visualization: Principles and Practice*, 2 ed. CRC Press, 2015. 2

[Ver11] VERBINSKI G.: Rango, 2011. Feature film. 3

[Zuc16] ZUCCONI A.: The secrets of colour interpolation, 2016. URL: `https://www.alanzucconi.com/2016/01/06/colour-interpolation/`. 2