Oulasvirta, Antti; De Pascale, Samuli; Koch, Janin; Langerak, Thomas; Jokinen, Jussi; Todi, Kashyap; Laine, Markku; Kristhombuge, Manoj; Zhu, Yuxi; Miniukovich, Aliaksei; Palmas, Gregorio; Weinkauf, Tino

Aalto Interface Metrics (AIM)

DOI:
10.1145/3266037.3266087

Published: 11/10/2018

Document Version
Peer reviewed version

Please cite the original version:
Aalto Interface Metrics (AIM): A Service and Codebase for Computational GUI Evaluation

Antti Oulasvirta¹, Samuli De Pascale¹, Janin Koch¹, Thomas Langerak¹, Jussi Jokinen¹, Kashyap Todi¹, Markku Laine¹, Manoj Kristhombuge¹, Yuxi Zhu¹, Aliaksei Miniukovich², Gregorio Palmas³, Tino Weinkauf³
¹Aalto University, ²University of Trento, ³KTH Royal Institute of Technology

ABSTRACT
Aalto Interface Metrics (AIM) pools several empirically validated models and metrics of user perception and attention into an easy-to-use online service for the evaluation of graphical user interface (GUI) designs. Users input a GUI design via URL, and select from a list of 17 different metrics covering aspects ranging from visual clutter to visual learnability. AIM presents detailed breakdowns, visualizations, and statistical comparisons, enabling designers and practitioners to detect shortcomings and possible improvements. The web service and code repository are available at interfacemetrics.aalto.fi.

INTRODUCTION
AIM is an online service and an open code repository for computational evaluation of graphical user interface (GUI) designs. AIM pools several previously published metrics and models, which have been empirically shown to be predictive of how users perceive, search, and aesthetically experience a design. These metrics range from design heuristics like symmetry to metrics and full-fledged models such as saliency and visual clutter. The source code is open-sourced, inviting contributions from researchers and practitioners. A well-documented Python API enables the system to be easily extended with new metrics.

The prime goal of AIM is to facilitate the use and appropriation of computational methods in design practice. Typically, evaluation in interface and interaction design practice relies on personal experience and empirical testing, and less so on computational modeling. While some previous papers (e.g. [8, 15, 20]) have applied models and metrics to assist designers, they do not offer explanations and automated evaluations. On the other hand, previous work on automated evaluation e.g. [1, 5, 19, 22]) has had limited scope (in terms of number of metrics) or have not been easily extendable. With AIM, we explore a large range of interface metrics, covering various aspects related to usability and performance, and provide a flexible system that can be easily extended to address additional aspects. An overview of the web user interface is given in Figure 1.

A secondary goal of AIM is to facilitate research efforts centered around computational models of human-
METRICS AND MODELS IN AIM

To cover a wide range of criteria important for UI design, we selected 17 metrics and models (listed in Table 1), and implemented these in AIM. They cover four categories:

1. **Color Perception**: These cover different aspects related to the colorfulness of the design, and how this influences perception and usability.
2. **Perceptual Fluency**: These estimate the ease with which the visible information is perceived and processed visually and aesthetically.
3. **Visual Guidance**: These predict visual search performance while navigating the design.
4. **Accessibility**: This estimates whether the design meets relevant accessibility requirements.

IMPLEMENTATION

AIM is implemented as a web application, consisting of two separate components: frontend and backend. The frontend handles the web user interface, including the metrics selection form and the presentation of results. This is implemented using the Vue.js JavaScript framework. The backend handles the evaluation of metrics, and is implemented using the Python-based Tornado web framework. In addition, the backend contains two subcomponents: metrics library and segmentation script. Both the metrics included in the library and the segmentation script are implemented in Python, excluding visual search and grid quality metrics which are implemented in Common Lisp and MATLAB, respectively.

When a user enters an URL and selects which metrics to run a request is made to the backend. Next, the backend captures a screenshot of the target website using Headless Chrome and runs the segmentation script against it to generate a list of visible elements (for segmentation-based metrics only). Each of these elements contain the following properties: 1. Identifier; 2. Absolute Position (x, y); 3. Size (width, height); and 4. Base64-encoded image data. The selected metrics are then computed with the base64-encoded representation of the website and the list of segmentation elements as input arguments. The metrics are independent from each other, and therefore can be run in parallel to increase total performance of the server. Finally, the results from the metrics are pushed one by one to the frontend via WebSocket as and when they become available.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
<th>Comp. time*</th>
<th>Ref</th>
</tr>
</thead>
<tbody>
<tr>
<td>File size†</td>
<td>The file size (JPEG &amp; PNG) of the image in bytes</td>
<td>0.000 (0.000)</td>
<td>[13]</td>
</tr>
<tr>
<td>Color Variability†</td>
<td>The amount of different colors in RGB, HSV, and LAB color spaces</td>
<td>1.946 (0.400)</td>
<td>[4, 12] [13]</td>
</tr>
<tr>
<td>Static Color Clusters†</td>
<td>Number of bins with &gt;5 px. Bins are 32×32×32 px (in RGB)</td>
<td>2.307 (0.671)</td>
<td>[12, 13]</td>
</tr>
<tr>
<td>Dynamic Clusters†</td>
<td>Number of bins with &gt;5 pixels based on distance between pixels</td>
<td>42.435 (41.101)</td>
<td>[12, 13]</td>
</tr>
<tr>
<td>Colorfulness†</td>
<td>The standard deviation of pixels in the RGB color space</td>
<td>3.065 (0.228)</td>
<td>[4]</td>
</tr>
<tr>
<td>Luminance†</td>
<td>Standard deviation of luminance corrected for display perception</td>
<td>5.579 (0.674)</td>
<td>[12]</td>
</tr>
<tr>
<td>Color Harmony†</td>
<td>The sum of the distance of all pixels to a color scheme.</td>
<td>71.516 (59.390)</td>
<td>[2]</td>
</tr>
<tr>
<td>Edge Density†</td>
<td>Ratio of edge pixels to all pixels</td>
<td>0.115 (0.001)</td>
<td>[12, 18]</td>
</tr>
<tr>
<td>Contour Congestion†</td>
<td>Ratio of congested edge pixels to all edge pixels</td>
<td>11.165 (2.696)</td>
<td>[10, 12] [21]</td>
</tr>
<tr>
<td>Figure-Ground Contrast†</td>
<td>The discriminability of the foreground from the background based on contrast.</td>
<td>0.206 (0.214)</td>
<td>[3, 12] [16]</td>
</tr>
<tr>
<td>Symmetry†</td>
<td>Ratio of edges that are mirrored either horizontal, vertical, or diagonal</td>
<td>3.516 (2.197)</td>
<td>[12]</td>
</tr>
<tr>
<td>Visual Complexity†</td>
<td>Balance, symmetry, and equilibration based on quadtree decomposition</td>
<td>17.349 (6.600)</td>
<td>[14, 17] [23]</td>
</tr>
<tr>
<td>Grid Quality†</td>
<td>Alignment to grids</td>
<td>8.020 (0.683)</td>
<td>[13]</td>
</tr>
<tr>
<td>White Space†</td>
<td>Proportion of non-covered space on the website</td>
<td>0.005 (0.007)</td>
<td>[13]</td>
</tr>
<tr>
<td>Itti-Koch Salience§</td>
<td>The degree to which a pixel stands out</td>
<td>0.897 (0.149)</td>
<td>[6, 9]</td>
</tr>
<tr>
<td>Visual Search Performance§</td>
<td>Visual search time for page elements</td>
<td>1.534 (0.954)</td>
<td>[7]</td>
</tr>
<tr>
<td>Color Blindness§</td>
<td>Images as seen by the three common color blindness types</td>
<td>13.369 (2.916)</td>
<td>[11]</td>
</tr>
</tbody>
</table>

†Color Perception, †Perceptual Fluency, †Visual Guidance, §Accessibility

* Avg time (and SD) per screenshot (in seconds), computed using top 10 sites in the Alexa Top 500.

Table 1. Metrics and models in AIM.
ACCESS AND EXTENSIBILITY

The web service and code repository of AIM are fully open-sourced, and available at interfaceometrics.aalto.fi. AIM has been designed from the ground-up with extensibility in mind. As a result, new metrics can be added with relatively small effort using a uniform API. In practice, a new metric is defined in a separate Python file. It takes the screenshot or segmented page as input, and should return numerical scores, or an image, as output. It can be plugged in to the system by registering it in the front- and back-end.

REFERENCES


