Automating Color Ramp Detection and Modification to Enhance Pixel Art

Lenka Hake[1]

Supervisors: Elmar Eisemann[1]**, Petr Kellnhofer**[1]**, Mathijs Molenaar**[1]

[1]**EEMCS, Delft University of Technology, The Netherlands**

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 22, 2025

Name of the student: Lenka Hake
Final project course: CSE3000 Research Project
Thesis committee: Petr Kellnhofer, Mathijs Molenaar, Joana de Pinho Gonçalves

Abstract

Pixel art relies on carefully constructed color ramps to simulate shading and depth within limited palettes. However, editing these ramps remains a tedious and error-prone manual process. This research introduces a semi-automatic tool that supports the detection and modification of color ramps in pixel art. The system builds a graph to model relationships between image colors based on perceptual similarity and spatial adjacency, then extracts and validates color ramps using customizable criteria. Afterwards, if a user edits a color, changes propagate consistently along the defined ramps, preserving their visual structure. The method streamlines palette editing while respecting artistic intent, offering both automation and control.

1 Introduction

Pixel art originated during the early days of digital graphics, when hardware limitations imposed strict constraints on memory and resolution [1]. These restrictions strongly affected the art, for instance, in the number of colors that could be displayed simultaneously. Artists were forced to work with very small palettes, often with less than 32 colors, requiring careful control to achieve the desired visual effects [2]. Even as modern hardware eliminated these limitations, pixel art has remained popular. Many contemporary artists still choose very small color palettes voluntarily, seeing it as a way to preserve a nostalgic style or stimulate their creativity [1].

The most common and widely adopted strategy for building minimalistic pixel art palettes involves the construction of **color ramps**: sequences of colors forming meaningful progressions, often from dark to light or through subtle shifts in hue (Figure 1). These ramps enable artists to simulate depth, shading, and material changes within tightly constrained palettes. For many pixel artists, ramps are fundamental organizational units for gaining the biggest visual impact with very few colors [3–6].

Figure 1: Example of a pixel art palette built from color ramps. Pixel artists often organize their palettes as grids where each horizontal and vertical segment is seen as a separate ramp. However, any path through the grid that does not repeat steps can be considered a ramp.

Despite their popularity, ramp design remains laborious and mentally demanding. Selecting appropriate colors is seldom straightforward and often requires ongoing refinement as the artwork evolves. Altering even a single color typically triggers adjustments across entire ramps to preserve their intended progression. The challenges are further amplified when multiple ramps overlap within the palette. In some cases, artists work with existing images where the original ramps are no longer known, requiring them to manually trace and reconstruct color relationships. As a result, modifying the colors without compromising the artwork's integrity can be a painstaking process [6, 7].

Previous efforts to enable automatic ramp management remain limited in scope. Existing pixel art tools typically represent palettes as unstructured lists or grids of colors, without support for organizing them into ramps or propagating edits across related colors [8, 9]. Image recoloring and palette management methods from other fields—such as photo editing or data visualization—cannot typically operate on semantically meaningful subsets like color ramps, or do not facilitate fine-grained user control, which is critical to pixel art. General-purpose palette extraction methods similarly fall short when it comes to identifying color ramps [10, 11].

This leads us to the main research question of this work: *How can we automate the detection and modification of color ramps in pixel art to streamline palette editing?*

We propose a novel tool that enables the identification of color ramps in pixel art through a semi-automatic pipeline and simplifies their editing. Ramp detection begins with the construction of a **Color Connectivity Graph**, which captures relationships between image colors based on perceptual similarity, spatial adjacency, or a combination of both. From this graph, the system extracts candidate ramps using flexible validation strategies aimed at identifying meaningful color transitions. These candidates are then filtered and clustered to remove redundancy and resolve overlaps. Once the final set of ramps is selected, the tool enables intuitive editing through **ramp-aware change propagation**—a mechanism that automatically distributes changes made to a single color across all associated ramps, preserving their original structure.

Key contributions include graph-based strategies for modeling color relationships in pixel art using perceptual and spatial connections; methods for identifying and evaluating color ramps aligned with artistic goals; and a change propagation mechanism that simplifies ramp editing.

2 Related Work

This section surveys prior work relevant to our approach, spanning color theory and perception models, pixel art conventions, editing software, ramp detection strategies, and palette-based recoloring methods.

2.1 Color Models and Perception

Color can be represented using different models, each optimized for specific tasks. The **RGB** (Red, Green, Blue) model is the most common one in digital systems, describing colors as additive mixtures of primary light intensities. While efficient for rendering and storage, RGB does not align well with how humans perceive or reason about color differences [12].

For this reason, many artistic workflows and color editing tools use the **HSV** (Hue, Saturation, Value) model, which represents colors through more perceptually meaningful components [12]. HSV is particularly popular among pixel artists because it enables intuitive control over color ramps and transitions [6]. However, it is not **perceptually uniform**, i.e. changes in the HSV components do not always translate to equally perceivable changes for the human eye [13].

To compensate for the limitations of RGB and HSV, color science introduces **perceptual color models** like **CIELAB**, which aim to better reflect human vision. The **LCH** (Lightness, Chroma, Hue) model is a cylindrical transformation of CIELAB that separates lightness from chromatic attributes. While not perfectly uniform, LCH offers greater perceptual consistency than HSV, and its structure makes it easier to understand and manipulate than CIELAB [14].

The **CIEDE2000** formula builds upon CIELAB by introducing a refined definition of color difference. It computes a single ΔE value that quantifies how perceptually distinct two colors are, factoring in complex interactions between lightness, chroma, and hue [10].

In our tool, HSV, LCH, and CIEDE2000 serve complementary roles. HSV offers interpretability and fine-grained control aligned with pixel art workflows; CIEDE2000 excels at measuring overall perceptual difference between colors; and LCH provides a perceptually aligned framework for analyzing directional trends in color transitions.

2.2 Pixel Art Color Practices

There is little academic literature on pixel art and its use of color. Most knowledge comes from community contributions such as **tutorials** [5, 15, 16], **forum posts** [3], and **blogs** [4, 6]. These materials, often written by experienced artists, form the practical foundation of pixel art. They emphasize the use of limited palettes and offer best practices for building color ramps, such as avoiding excessive saturation or value that might "burn" the viewer's eyes [3]. While there is no formal definition of what makes a good color ramp, artists commonly advise that ramps should be smooth and consistent, with a clear direction of change in hue, saturation, and value. Ramp construction is typically done through trial and error, relying on intuition rather than formal rules [4, 6].

2.3 Software Support for Color Ramps

Modern pixel art editors such as **Aseprite**[1], **Pro Motion**[2], and **GraphicsGale**[3] offer robust palette management features, including gradient generators. However, they provide little support for organizing colors into ramps, treating palettes as unordered lists or grids, and relying on the artist to infer color relationships.

Some attempts have been made to automate color ramp detection. The Aseprite extension **Color Ramp Sort** [9] clusters colors into linear ramps using RGB collinearity heuristics. While effective for simple, disjoint ramps, it struggles with overlapping or complex structures and only groups colors visually, without supporting ramp-level modifications. In contrast, Kensler's blog post **Mapping Pixel Art Palettes** [8] models palettes as directed graphs based on perceptual distance and lightness order. Although more structurally insightful, this method does not isolate individual ramps. While both approaches are valuable early contributions, they remain limited and do not enable dynamic ramp editing.

[1]https://www.aseprite.org/

[2]https://www.cosmigo.com/promotion

[3]https://graphicsgale.com/us/

2.4 Palette Extraction and Image Recoloring

Palette extraction from natural images is a well-established research area. Techniques such as **k-means clustering**, **median cut**, and **mean-shift** are commonly used to identify representative colors by reducing the full color range to a smaller, usable set. More recent methods leverage deep learning to infer more complex palettes [10, 17]. However, these approaches are designed for tasks like **image abstraction**, **style transfer**, or **color theme generation** [10, 18, 19], and do not meet the specific demands of pixel art, which already uses limited color palettes. In pixel art, palette extraction means preserving **all unique colors** and ideally organizing them in a way that reflects their visual relationships.

Related to this is **image recoloring**, which involves adjusting image colors while preserving structure or content. Palette-based recoloring methods often work by decomposing the image into layers and remapping the palette [11, 20]. Some tools propagate changes based on color similarity or texture boundaries [19]. This project introduces a ramp-based propagation strategy, where edits automatically affect all related colors within associated color ramps.

3 Methodology

Our tool enables the detection and editing of color ramps in pixel art through a semi-automatic pipeline (Figure 2). The process begins with the construction of a **Color Connectivity Graph** (Section 3.1), which links colors that are likely to form ramp sequences based on perceptual similarity, spatial adjacency, or both. The tool then extracts candidate ramps from the graph using one of two validation strategies: one based on HSV components, and the other combining CIEDE2000 and LCH color metrics (Section 3.2). The extracted candidates are refined by filtering out redundant ramps (Section 3.3) and clustering similar ones to select representative examples (Section 3.4). After choosing the final ramp set, the user can apply **ramp-aware color modifications** (Section 3.5), where changes to a single color automatically propagate across all associated ramps while preserving their structure. User-defined parameters guide key decisions throughout the pipeline, allowing flexible adaptation to different palette structures and artistic goals.

3.1 Building a Color Connectivity Graph

To identify color ramps, we start by constructing a **Color Connectivity Graph** $G = (V, E)$, where each node $v \in V$ corresponds to a unique color in the input image. An edge $(u, v) \in E$ exists if the color pair satisfies a predicate $R(u, v)$, meaning u and v are likely to form successive steps in a ramp. The graph narrows the search space for ramp extraction: instead of evaluating all color sequences, we only need to explore paths through connected nodes. Depending on the construction strategy chosen by the user, R may reflect color similarity, spatial adjacency, or a combination of both. Illustrative examples are shown in Figures 4–5 in the Results section. The following sections define each predicate variant.

Color Similarity Graph

The first graph construction strategy connects colors that are **perceptually similar**, regardless of their spatial arrangement

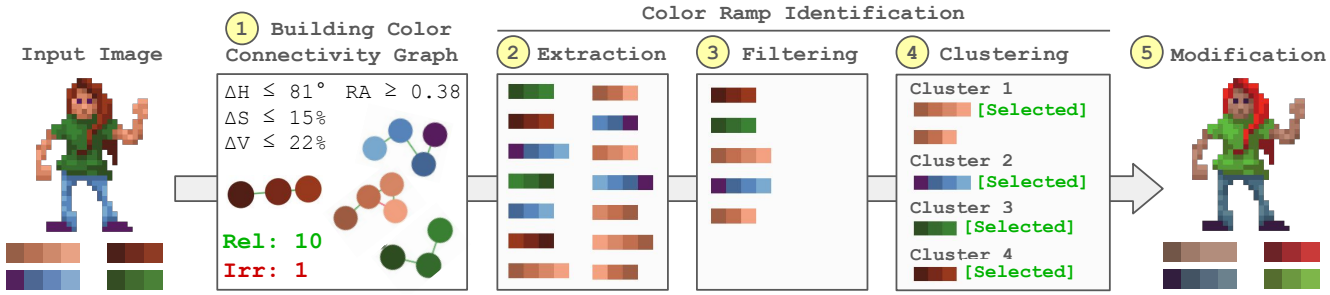

Figure 2: Overview of the semi-automatic pipeline for ramp-based color modification. The process consists of five steps: (1) constructing a Color Connectivity Graph based on similarity and spatial adjacency; (2) extracting candidate color ramps; (3) filtering out redundant results; (4) clustering similar ramps to select representative examples; and (5) applying ramp-aware modifications, where edits to one color propagate across all related ramps. *Input image was inspired by Daniel Silber's work [21].*

in the image. Similarity can be evaluated using either the HSV color space or the CIEDE2000 perceptual difference metric, depending on the selected method.

In the **HSV-based** approach, bounds are applied to each component individually. Let $\mathbf{c}(u) = (h(u), s(u), v(u))$ denote the HSV vector of color u, with separate thresholds $\boldsymbol{\theta} = (\theta_h, \theta_s, \theta_v)$. The predicate is defined as: $R_{\text{sim}}^{\text{HSV}}(u,v) = \bigwedge_{i=1}^{3} \left(|c_i(u) - c_i(v)| \leq \theta_i\right)$.

In the **CIEDE2000-based** approach, similarity is measured using the perceptual difference $\Delta E(u,v)$ with a single threshold $\theta_{\text{sim}}^{\Delta E}$: $R_{\text{sim}}^{\Delta E}(u,v) = \left(\Delta E(u,v) \leq \theta_{\text{sim}}^{\Delta E}\right)$.

Both methods identify perceptually similar colors but differ in behavior and suitability (see Section 2.1). CIEDE2000 provides perceptual uniformity and a single scalar threshold, making it useful for general similarity detection. HSV offers more targeted, component-wise control that aligns with artistic workflows, but may behave unpredictably in certain regions of color space. In both cases, the thresholds that define similarity can be configured by the user to reflect desired sensitivity.

Similarity graphs are useful for capturing perceptual relationships between colors, but they may also connect colors from unrelated regions, as they ignore spatial context.

Spatial Adjacency Graph

In pixel art, colors that appear adjacent on the canvas are often functionally related. To capture this, the second graph strategy adds an edge $(u,v) \in E$ if the colors u and v frequently appear next to each other in the image.

For each pixel of color u, its four immediate neighbors are examined. If a neighbor has color $v \neq u$, the adjacency count for the pair (u,v) is incremented. Let $A(u,v)$ denote the number of adjacent occurrences of (u,v), and $T(u)$ the total number of pixels of color u. The **Relative Adjacency (RA)** is defined as $RA(u,v) = \frac{A(u,v)}{T(u)}$. Since the graph is undirected, an edge is added if the maximum RA in either direction exceeds the user-defined threshold θ_{adj}: $R_{\text{adj}}(u,v) = \left(\max\left(\text{RA}(u,v), \text{RA}(v,u)\right) \geq \theta_{\text{adj}}\right)$.

RA is more robust than using $A(u,v)$ alone, as it accounts for under-represented colors and normalizes across image size. However, it cannot detect relationships between colors that never appear directly adjacent. As such, important ramp connections may be missed in highly segmented art styles.

Hybrid Graph

The two previous strategies capture complementary aspects of how ramp relationships arise in pixel art. Similarity-based graphs connect perceptually close colors, even across disjoint regions, while spatial adjacency focuses on local usage patterns within the image. Since each method has limitations when used in isolation, the proposed **hybrid approach** combines both strategies to mitigate their respective weaknesses.

An edge $(u,v) \in E$ is added based on either the intersection or union of the similarity and adjacency graphs:

$$R_{\text{hybrid}}^{\cap}(u,v) = R_{\text{sim}}(u,v) \wedge R_{\text{adj}}(u,v)$$

$$R_{\text{hybrid}}^{\cup}(u,v) = R_{\text{sim}}(u,v) \vee R_{\text{adj}}(u,v).$$

Intersection mode retains only edges between colors that are both perceptually similar and frequently adjacent, reducing noise and ambiguity, but it may exclude valid ramp connections if either condition is not met. In contrast, **union mode** includes all edges from both graphs, offering broader coverage but potentially introducing more irrelevant links.

Together, these strategies define four graph construction modes: **color similarity**, **spatial adjacency**, **intersection**, and **union**. Each emphasizes different color relationships, and their suitability depends on the palette structure and image layout. The tool does not impose a fixed choice—users can select the method and thresholds that best align with their artistic intent and the image style. The resulting graph then forms the foundation for extracting candidate color ramps.

3.2 Extracting Cadidate Ramps

Once a graph has been constructed, the next step is to extract sequences of connected colors that form valid ramps. To do this, the tool performs multiple runs of **depth-first search**, each starting from a different node in the graph and recursively exploring paths through neighbouring nodes. At each step, the current path is evaluated against ramp validation criteria: smoothness, monotonicity, and meaningfulness. If the path satisfies the criteria, the search continues; otherwise, the longest valid prefix is stored as a candidate ramp, and the algorithm backtracks. This process repeats until all nodes have been used as starting points, resulting in a comprehensive set of candidate ramps.

To ensure the extracted sequences represent visually coherent ramps, we define the three validation criteria as follows:

smoothness ensures that step sizes between colors are consistent; **meaningfulness** ensures that steps are neither imperceptibly small nor excessively large; and **monotonicity** requires a clear direction of change without reversals [4, 5]. Figure 3 shows examples of ramps that violate these principles.

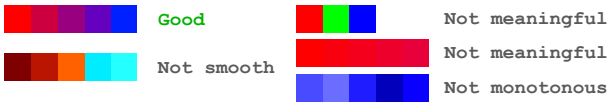

Figure 3: Examples of invalid ramps. The top-left ramp satisfies all validation criteria, while the others each violate one: smoothness (inconsistent step sizes), meaningfulness (steps that are too small or too large), or monotonicity (directional reversals).

The three criteria can be evaluated using either an HSV-based approach or a combination of CIEDE2000 and LCH perceptual metrics. While both enforce the same underlying principles, they rely on different models and evaluation strategies, which are detailed in the following sections.

Ramp Validation Using HSV

The first validation method checks whether a sequence of colors forms a valid ramp by independently evaluating changes in hue, saturation, and value. It relies on simple, interpretable parameters defined by the user, each corresponding to one of the three ramp quality criteria. This makes it straightforward to extract ramps that follow specific HSV progressions, which are common in pixel art workflows.

Each component is evaluated using four tunable parameters: the **minimum** and **maximum step** sizes, which enforce meaningfulness by filtering out steps that are either too subtle to perceive or too extreme to remain coherent; the **maximum variance**, which ensures smoothness by limiting fluctuations in step size along the ramp; and a **monotonicity constraint**, which optionally requires that a specific component (e.g., value) consistently increases or decreases throughout the sequence.

Ramp Validation Using CIEDE2000 and LCH

The second validation method combines CIEDE2000 and LCH to evaluate ramps in a way that reflects human color perception more accurately than HSV. CIEDE2000 provides a precise measure of perceived color difference, making it well-suited for evaluating step size and consistency. However, it treats differences as scalar values and does not capture the direction of color change. To address this, we supplement CIEDE2000 with LCH, which enables geometric assessment of directional trends in lightness, chroma, and hue. Compared to HSV, LCH offers a more perceptually meaningful structure, although it is still less accurate than CIEDE2000 for judging overall color difference.

This method uses four tunable parameters to evaluate the ramp quality criteria. **Meaningfulness** is enforced through the **minimum** and **maximum step sizes** in ΔE, filtering out transitions that are either too subtle or too abrupt. **Smoothness** is ensured by limiting the **maximum variance** in ΔE, promoting consistent spacing throughout the ramp. **Monotonicity** is enforced by constraining the **maximum angular deviation** in LCH space, ensuring a consistent direction of change.

While this method is robust and simplifies the validation of ramps with complex multidimensional changes, the HSV-based approach remains valuable when finer control is needed over ramp progression and the behavior of individual color components.

3.3 Filtering Out Redundant Ramps

Due to the repetitive nature of the extraction process, the resulting set of candidate ramps often contains redundant or overlapping sequences. Many candidates differ only slightly in order, direction, or length, which inflates the results and complicates ramp management. In response, the system applies filtering and clustering to consolidate similar candidates and produce a cleaner, more manageable set.

The filtering stage removes two main types of redundancy. First, it eliminates **permutations**—ramps that contain the same set of colors but in a different order. Second, it discards **subsequences** that are fully contained within longer ramps. While some shorter ramps may be meaningful on their own, filtering out subsequences effectively reduces noise while preserving longer ramps that are equally—or more—important.

3.4 Clustering Similar Ramps

To further consolidate the candidate ramps and ensure a diverse final set, we cluster them using a **perceptually-aware edit distance**. Each ramp is treated as a sequence of colors, and pairwise distances are computed using a modified Damerau–Levenshtein metric [22]. Color substitutions are allowed at zero cost when the perceptual difference ΔE is below 10 (relatively similar) [23], reducing the distance between visually similar ramps. Insertions, deletions, and transpositions are assigned fixed costs, with transpositions allowed for adjacent color swaps. The recursive distance is defined as:

$$D(i,j) = \min \begin{cases} D(i-1,j)+1 & \text{(deletion)} \\ D(i,j-1)+1 & \text{(insertion)} \\ D(i-1,j-1)+\delta(c_i,d_j) & \text{(substitution)} \\ D(i-2,j-2)+0.5 & \text{(transposition)} \end{cases}$$

with base cases $D(0,j) = j$, $D(i,0) = i$, and $D(0,0) = 0$. The substitution cost $\delta(c_i, d_j)$ is set to 0 if $\Delta E(c_i, d_j) < 10$, and 1 otherwise.

The resulting distance matrix is used for **hierarchical clustering** with average linkage, grouping visually similar ramps based on structure and color content.

To select a representative ramp from each cluster, we apply a **quality scoring function** based on four factors: smoothness, meaningfulness, monotonicity, and length. CIEDE2000 is used to assess perceptual step sizes and their consistency, while HSV is used to evaluate monotonicity, reflecting its alignment with artistic workflows and leveraging the strengths of each color model.

The scoring function consists of the following components. First, the **step size penalty** is calculated by comparing each ΔE step to an ideal perceptual range (10–50) [23], and computing the root mean square (RMS) of the deviations. This

favors ramps with perceptually meaningful transitions. Second, the **step consistency penalty** measures the uniformity of step sizes along the ramp, computed as the RMS of differences between consecutive ΔE values and scaled by 0.1 to normalize its weight. Third, monotonicity is assessed in HSV space by counting direction changes in each channel. Each component contributes a score of $1/(\text{changes} + 1)$, and the three values are summed to yield the overall **monotonicity score**. Finally, a small **length bonus** (0.05 per step) is added for ramps longer than the shortest ramp in the cluster, promoting longer ramps when quality is otherwise comparable. The monotonicity score and length bonus are added, while the two penalties are subtracted, to compute the final score.

Scoring is applied only within clusters, rather than across the full candidate set, to avoid favoring trivial structural variations. This ensures that the final output includes one high-quality, visually distinct ramp per group.

3.5 Final Ramp Selection and Color Modification

The tool presents the final candidate ramps—one from each cluster—for user inspection and optional refinement. Users may make small adjustments, such as adding or removing steps, to fine-tune the ramp structure before proceeding to color modification.

Once ramps are finalized, individual colors can be edited in RGB, HSV, or hexadecimal formats. When **propagation mode** is enabled, the tool automatically applies changes to all ramps containing the edited color.

Propagation is performed in HSV space, aligning with how artists typically understand ramps—as progressions in hue, saturation, and value. When a user modifies a color $\mathbf{c}_k = (h_k, s_k, v_k)$ to a new value $\mathbf{c}'_k = (h'_k, s'_k, v'_k)$, the tool computes propagation for each color $\mathbf{c}_i = (h_i, s_i, v_i)$ in the ramp (with $i \neq k$). The hue shift is preserved relative to the original color, using $\Delta h_i = (h_i - h_k + 0.5) \pmod 1 - 0.5$, and the propagated hue is computed as $h'_i = (h'_k + \Delta h_i) \pmod 1$. Saturation and value are scaled proportionally: $s'_i = s'_k \cdot \frac{s_i}{s_k}$ and $v'_i = v'_k \cdot \frac{v_i}{v_k}$.

The ramp's style is preserved, while the user's adjustment redefines the base from which the colors are derived.

3.6 Implementation Details

The tool is implemented in **Python 3.13** and uses **PyQt6** for the graphical user interface (GUI). Its modular architecture separates functionality into distinct components for image viewing, color manipulation, and graph visualization. Core scientific libraries—including **NumPy**, **Matplotlib**, and **NetworkX**—are employed for color space calculations and graph-based operations. Data persistence is managed through **JSON serialization**, enabling saving and loading of color ramps. The application supports real-time color updates via an event-driven architecture built on Qt's signal mechanism, and maintains global state using dedicated manager classes for color and ramp data. While the tool includes a functional GUI for interaction and editing, interface design is not the focus of this research.

4 Results

We evaluate three core components of the proposed method: **Color Connectivity Graph** construction, ramp identification, and ramp-aware color modification. The evaluation is conducted on a set of ten pixel art images, either created in-house or sourced from public repositories [13] and pixel art communities [3]. For each image, we have access to the original color ramps used by the artist, allowing us to assess how effectively the tool identifies meaningful ramps. These images are relatively small (up to 120×120 pixels), each using 5–15 colors across 1–4 ramps, with ramp lengths up to 14. Additionally, for two images, we include artist-modified versions to compare against the results produced by our propagation mechanism.

In this section, we present selected examples that illustrate key behaviors and trade-offs. The full dataset, including all ten test cases and their corresponding parameter settings, is provided in Appendix B.

4.1 Comparison of Graph Construction Strategies

We evaluate graph strategies across ten pixel art images, using the number of **relevant (Rel)** and **irrelevant (Irr)** edges as the primary criteria. An edge is considered relevant if it connects two consecutive colors from a known ramp. Relevant edges aid ramp extraction, while irrelevant ones add ambiguity and overhead. The best graph for each image is the one that maximizes relevant edges and minimizes irrelevant ones. In Figures 4 and 5, we highlight the best-performing graph for each case in green.

Thresholds are tuned by gradually tightening permissive settings to preserve relevant edges while minimizing irrelevant ones. For the color similarity graphs, both HSV- and CIEDE2000-based methods are tested per image, and the version yielding better results is selected.

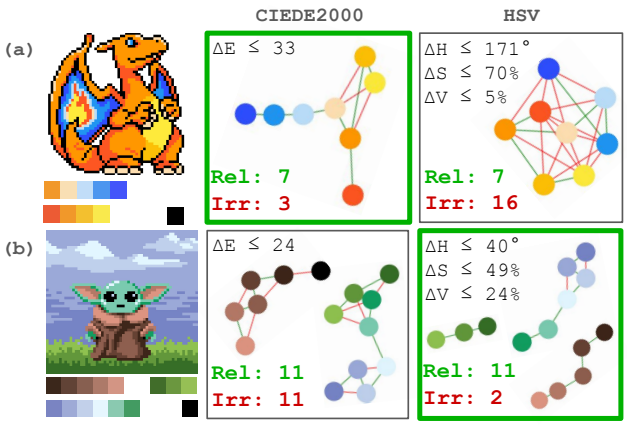

Figure 4: Comparison of CIEDE2000 and HSV similarity graphs. (a) Colors are perceptually similar but vary widely in HSV, making CIEDE2000 less noisy. (b) HSV performs better due to consistent ramp structure.

Figure 4 illustrates how HSV and CIEDE2000 graphs capture different types of color relationships, depending on the structure of the palette. In image (a), the ramps contain large

jumps in hue and saturation that nonetheless appear perceptually smooth. HSV needs loose thresholds to preserve relevant edges, resulting in more irrelevant links. CIEDE2000, being perceptually grounded, retains the intended relationships with fewer spurious edges. In image (b), the ramps follow consistent HSV progressions, allowing HSV to apply tighter thresholds that cleanly isolate ramp sequences. In contrast, CIEDE2000 tends to overconnect colors in the frame ramps due to their overall perceptual similarity, even when they form distinct linear progressions.

Figure 5 presents representative results on four images, each selected to highlight cases where one of the four graph construction strategies—color similarity, spatial adjacency, union, or intersection—performs best. Across the ten-image test set, the intersection graph performs best on 4 images, color similarity on 3, union on 2, and spatial adjacency on 1. When a hybrid graph (union or intersection) yields results equivalent to a base graph, we prefer the simpler option to reduce computational cost.

Intersection graphs consistently produce the cleanest output with the fewest irrelevant edges but can miss relevant connections when color or spatial continuity is disrupted, as in images (b) and (c) of Figure 5, which feature strong outlines. **Color similarity** and **union** graphs are more inclusive and reliably capture all relevant edges, though they tend to introduce more irrelevant links, particularly noticeable in image (c). **Spatial adjacency** graphs introduce the most noise and perform poorly on segmented images like (b) and (c), but work well in simpler cases with sharp color transitions, as demonstrated in image (d).

4.2 Ramp Identification Results

From the original ten-image dataset, we select six structurally diverse examples for ramp identification. The remaining four are excluded as they closely resemble other images in the set.

Figure 6 shows the whole identification process for three representative cases. In each test case, we apply the best-performing graph strategy determined earlier, except for images (a) and (b), where we deliberately use slightly suboptimal graphs to evaluate the robustness of the identification pipeline. Despite this, the system successfully recovers the exact ramps used in the original artwork in all six cases.

Ramp identification begins with extracting many initial candidates, which are then sharply reduced through filtering and clustering. For example, in image (c), the system reduces over 400 initial candidates to just 6. Although this is a substantial reduction, the final result still includes some clutter, as only one ramp is actually used in the artwork. In all other images, only the original ramps are retrieved. This is largely due to image (c) having the most colors and the longest ramp, exposing the limitations of filtering and clustering for longer or more complex palettes. Image (b), by comparison, demonstrates successful disambiguation from a dense cluster of six similar ramps.

Figure 7 presents an example of using HSV-based validation to extract ramps with minimal hue variation. While all six ramp reconstruction tests use the CIEDE2000 and LCH method for its simplicity, only the HSV method allows explicitly constraining hue change, something that is not possible with CIEDE2000 alone. By setting a low maximum hue step, the tool can isolate ramps that lack hue shifts, useful for identifying shading sequences that diverge from the common pixel art practice of shifting hue along with value.

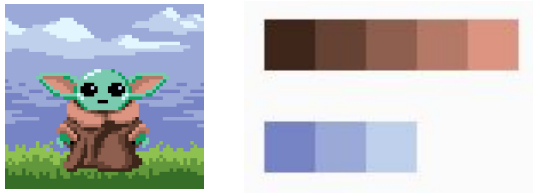

Figure 7: Results of ramp extraction using HSV validation with a hue step threshold of 10. The method isolates ramps that vary mainly in value, highlighting shading sequences with minimal hue change.

Tool interaction also revealed that ramp identification is highly sensitive to graph quality and parameter settings. Clean graphs with few irrelevant edges consistently yield more accurate results, while noisier graphs inflate the candidate set and increase the risk of discarding valid ramps. Careful parameter tuning is especially important: the CIEDE2000-based method proved sensitive, with small changes in perceptual distance (ΔE) affecting ramp validity. The HSV method is more tolerant but has parameters to tune. These findings emphasize that effective use of the tool requires thoughtful configuration tailored to each image's palette structure.

4.3 Color Modification and Propagation Examples

To evaluate ramp-aware color editing, we selected two images for which we had access to artist-created modified versions. In image (a), the artist reduced eye strain by lowering overall saturation and value. In image (b), the goal was to make the character's hair stand out more and adjust the surrounding colors accordingly for better contrast.

Using our tool, we applied two and four edits respectively, modifying selected ramp entries with propagation enabled when applicable. This eliminated the need to adjust each color individually. After each step, we computed the root mean square error (RMSE) in sRGB relative to the artist's version. As shown in Figure 8, the RMSE decreased consistently with each edit, indicating convergence toward the intended result. Both sequences concluded with a normalized RMSE of 0.03 or lower, producing images visually close to the artist-authored changes.

While ramp-aware propagation streamlines the editing process, it cannot fully replicate the nuanced adjustments artists often apply when modifying a palette. In image (a), for instance, preserving the highlight required manually increasing the value of the yellow color with propagation turned off. In image (b), although the main changes were applied automatically, the resulting contrast in some areas—such as the green shirt and light brown skin—remained lower than in the artist's version. These examples illustrate both the strengths and limitations of the method, showing that combining propagation with occasional manual edits remains important for achieving artist-level refinements.

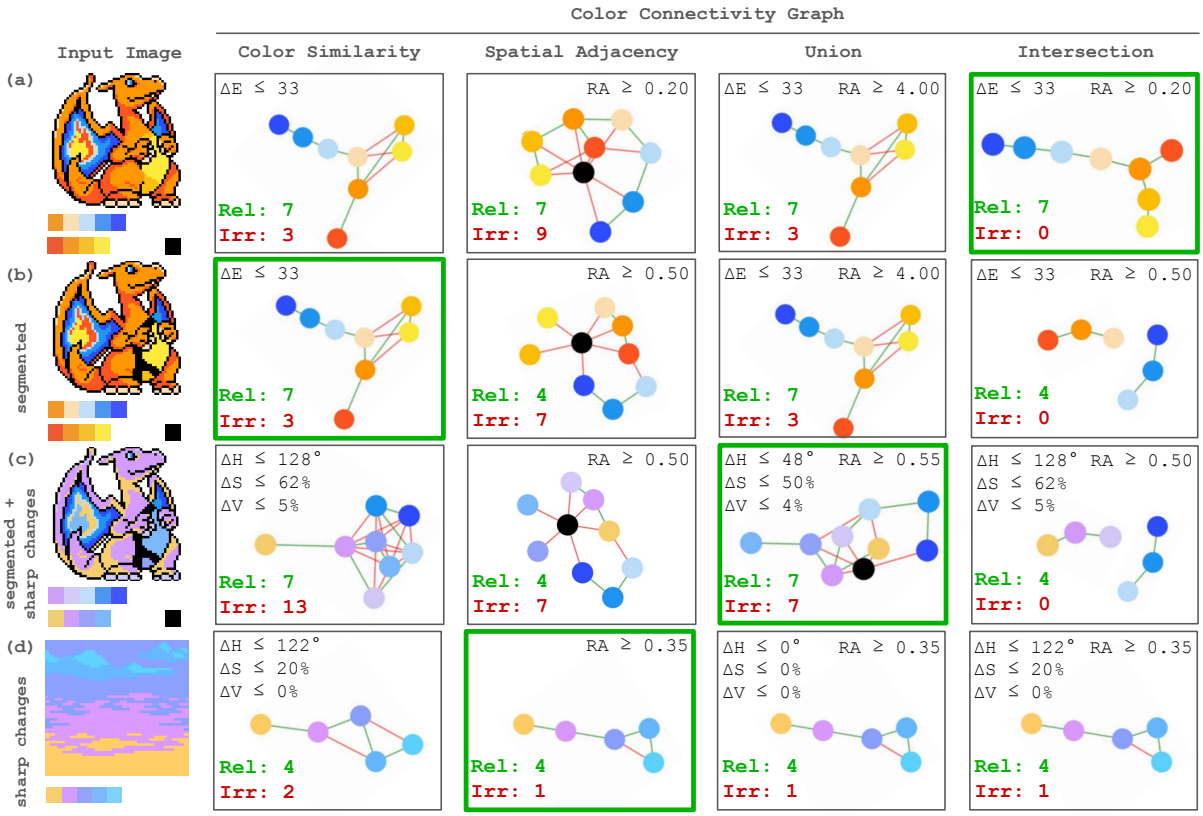

Figure 5: Representative results for four graph construction strategies on selected images. Each subfigure highlights a case where one method—color similarity, spatial adjacency, union, or intersection—performs best. Images (b) and (c) show challenges with segmentation and outlining, while (c) and (d) demonstrate cases with sharp color transitions.

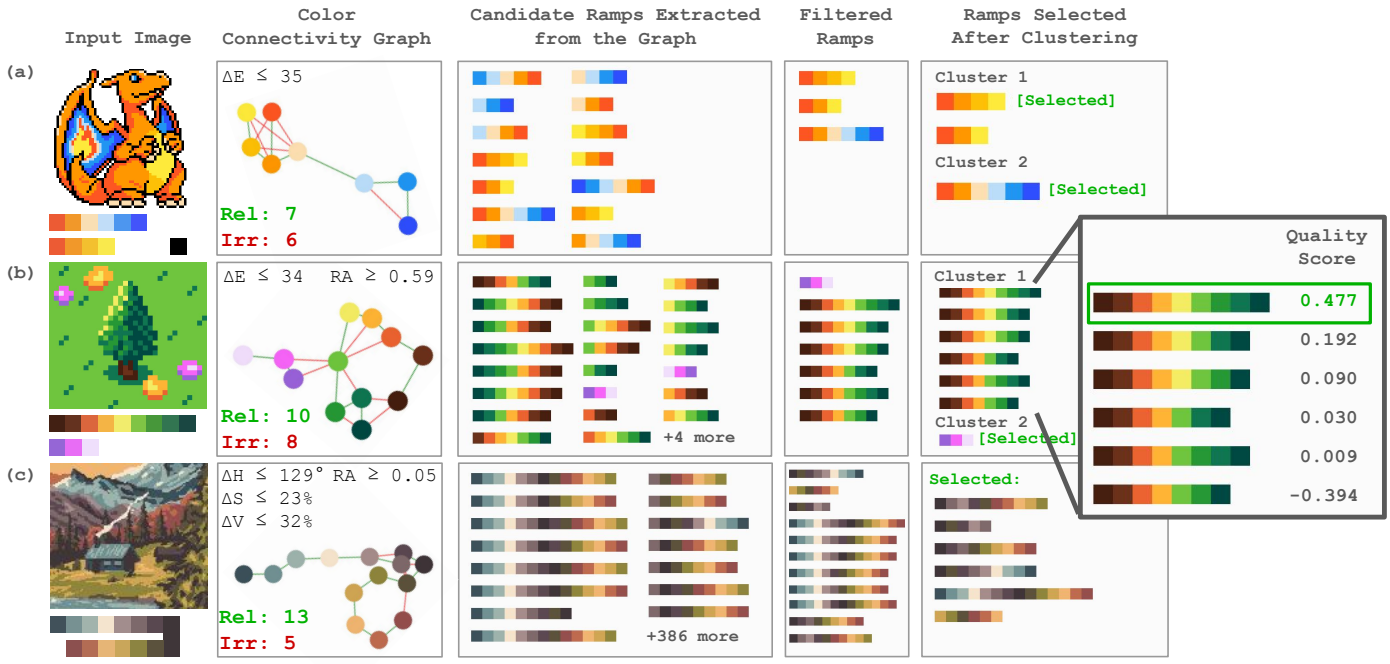

Figure 6: Ramp identification results for three representative images. In each case, the correct ramps used in the artwork are successfully recovered. Example (b) demonstrates effective disambiguation within a dense candidate cluster, while (c) highlights residual clutter despite reducing over 400 initial candidates, revealing the limitations of filtering and clustering when handling longer ramps. *Image (c) is cropped from original artwork by Jimison3 [24].*

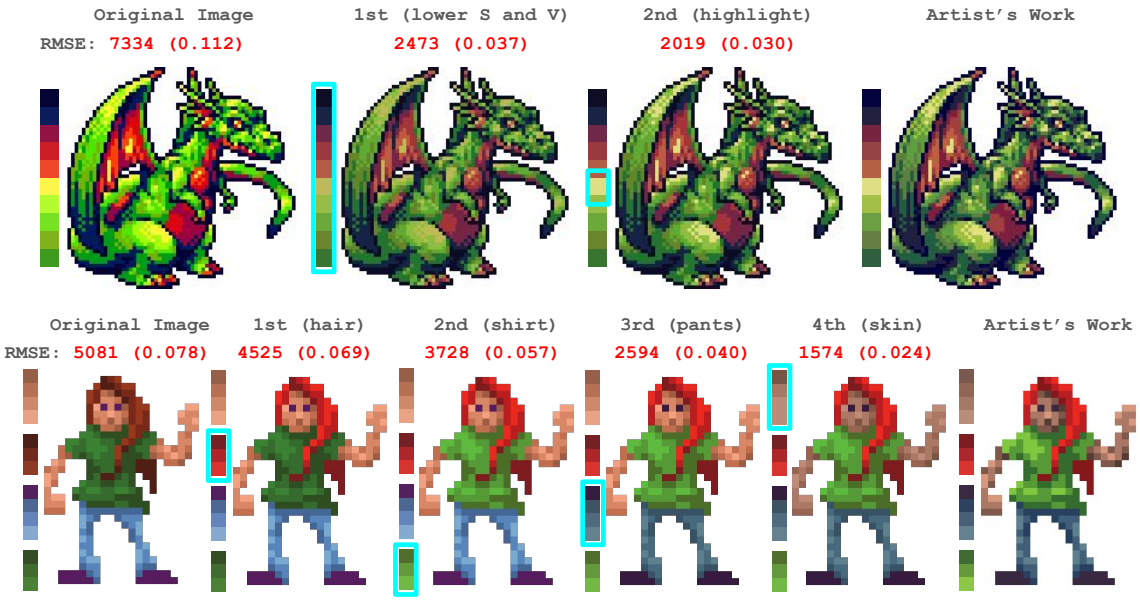

Figure 8: Ramp-aware color editing results compared to artist-modified target images. In both cases, a small number of edits with propagation progressively reduces the visual difference, measured as RMSE in sRGB. Final outputs closely match the artist's version, with normalized RMSE at 0.03 or below. A manual adjustment was required in (a) to preserve the highlight, while some contrast differences remain in (b). *Inputs and artist-modified images in (a) are redrawn after a post by Cure [3]; (b) were inspired by Daniel Silber's work [21].*

5 Discussion

This work introduces a semi-automatic tool that supports the identification and editing of color ramps in pixel art, addressing a longstanding challenge in palette-based workflows. Key contributions include graph-based strategies for modeling color relationships through perceptual and spatial connections, allowing the tool to adapt to diverse image structures and ramp styles. We also define clear validation criteria—smoothness, meaningfulness, and monotonicity—for identifying and evaluating color ramps in a way that aligns with artistic intent. Finally, the proposed ramp-aware change propagation mechanism simplifies the editing process by automatically distributing color edits across related ramps, reducing manual effort while preserving the intended visual structure. Together, these components form an integrated framework, supporting artists in managing complex palettes with greater efficiency.

While ramp-aware editing significantly reduces manual effort, ramp extraction remains a bottleneck. The process requires careful tuning of the graph and validation parameters, especially for complex or segmented images. Although ramps only need to be identified once and can be reused, initial setup can be time-consuming. The current method also struggles with nuanced artistic edits and lacks automated parameter tuning or suggestions for corrections.

Although the initial results are promising, our evaluation of the tool remains limited in scope. The ten test images were selected to reflect a variety of palette structures and challenges, but they do not capture the full diversity of pixel art styles. To better understand the tool's general applicability and limitations, broader testing is needed, both in terms of image variety and feedback from practicing artists using the tool in real-world contexts.

Future work will focus on improving both performance and generalization. This includes optimizing graph traversal and clustering algorithms, introducing smarter default thresholds, and exploring data-driven approaches for parameter tuning. Enhancing the modeling of spatial relationships and integrating them more effectively with perceptual similarity could improve ramp detection across a wider range of art styles. More advanced propagation techniques may also enable complex adjustments, such as selectively enhancing contrast or reinforcing color harmony. Incorporating guided corrections or automatic detection of low-quality ramps could further assist users in refining results. Finally, extending the method to support higher-resolution images could help bridge the gap between pixel art tooling and broader digital art workflows.

6 Conclusion

This work set out to address a critical gap in software support for pixel art: the lack of tools that enable structured, intuitive management of color ramps, a central organizational unit in pixel art palettes. In response, we developed a semi-automatic system to detect ramps in pixel art images and streamline their editing. The tool models color relationships as graphs using perceptual and spatial information, identifies ramp structures based on artist-aligned criteria, and enables ramp-aware color adjustments. Our approach offers a practical solution to the longstanding challenges of manual palette editing, preserving artistic intent while reducing the required effort. Although some limitations remain, particularly in parameter tuning and complex ramp detection, the system represents a meaningful step toward automating ramp identification and simplifying color modifications in pixel art.

7 Responsible Research

This section reflects on the responsible research practices followed throughout the project, with a focus on reproducibility, research integrity, and broader impact.

7.1 Reproducibility and Research Integrity

To support reproducibility, all source code used in this project has been made publicly available in a **GitHub repository**[4], along with full documentation. The methods described in the report include all relevant implementation details, design decisions, and formulas. The algorithm is fully deterministic, with no random or stochastic components, ensuring that results are repeatable under the same conditions. The parameters used to generate graph results are included in the figures and all other parameters are in Appendix B. Together, the code, documentation, and detailed reporting make it possible for others to reproduce all key findings without requiring further clarification.

We evaluated our tool on ten pixel art images. Seven were created specifically for this project. Of the remaining three, one was a cropped version of an artwork from a public gallery, one was inspired by existing work, and one was a redrawn version of a public tutorial example. All sources and credits were clearly indicated in figure captions and references.

An AI coding assistant (a plugin for PyCharm) was used during development to support general debugging, code documentation, refactoring, and the implementation of the tool's graphical user interface. The assistant's role was limited to practical implementation support and did not influence the method design or overall research direction. All ideas and design decisions were original and grounded in insights from prior related work. A selection of representative AI prompts is included in Appendix A for transparency.

7.2 Impact on Pixel Art Community

Pixel art is a medium where artists value precision and direct control over every visual element, including the palette. In this context, automating ramp-based color modifications may conflict with the core values of the community. This research acknowledges that tension and responds by designing a tool that prioritizes user agency at every stage. Advanced parameters are exposed for all major processes, such as ramp detection, validation, and color propagation. This enables artists to guide and refine the results according to their intent.

Rather than replacing artistic judgment, the tool aims to support it. It reduces repetitive tasks while allowing full transparency and customization in how ramps are identified and edited. By aligning with the creative values of the pixel art community, the tool has the potential to be adopted as a technical aid in creative workflows.

The broader influence could include encouraging more structured palette practices, lowering entry barriers for newer artists, and potentially inspiring further tool development that respects artist intent. The goal is not to standardize or automate artistic decisions, but to give artists better infrastructure for expressing themselves.

[4] https://github.com/MineaSolas/AutomatingColorRamps

The tool supports export and import of detected color ramps, enabling reuse and iterative refinement across editing sessions. However, this functionality currently relies on a custom JSON-based format, which is not compatible with existing pixel art editors such as Aseprite, Pro Motion, or Pyxel Edit. As a result, integration with other tools is limited at this stage. Future versions could improve on this by adopting or translating to widely used palette formats.

References

[1] S. Paez, "A visual renegade: A phenomenological and aesthetical examination of pixel art," Master's thesis, Vrije Universiteit Amsterdam, 2022.

[2] N. Kylmäaho, "Pixel graphics in indie games," Bachelor's thesis, Tampere University of Applied Sciences, 2019.

[3] Cure. "The pixel art tutorial." (2010), [Online]. Available: https://pixeljoint.com/forum/forum_posts.asp?TID=11299 (visited on 05/02/2025).

[4] R. Schlitter. "Pixelblog 1: Color palettes." (2018), [Online]. Available: https://www.slynyrd.com/blog/2018/1/10/pixelblog-1-color-palettes (visited on 05/02/2025).

[5] G. Function. "Development workflow – chapter 9: Creating a color palette (part 2)." (2023), [Online]. Available: https://gbstudiocentral.com/tips/dwf-c9-creating-a-color-palette-part-2/ (visited on 05/02/2025).

[6] Lux. "Color theory for pixel artists: It's all relative." (2020), [Online]. Available: https://pixelparmesan.com/blog/color-theory-for-pixel-artists-its-all-relative (visited on 05/02/2025).

[7] M. Azzi, *Pixel Logic: A Visual Guide to Pixel Art*. Self-published, 2017.

[8] A. Kensler, *Mapping pixel art palettes*, http://eastfarthing.com/blog/2016-05-27-mapping/, Accessed: 2025-06-02, 2016.

[9] M. Sagadin, *Color ramp sort: An aseprite extension for sorting palettes by color ramps*, https://github.com/matsagad/color-ramp-sort, Accessed: 2025-06-02.

[10] Y. Gao, J. Liang, and J. Yang, "Color palette generation from digital images: A review," *Color Research & Application*, vol. 50, no. 3, pp. 250–265, 2025.

[11] S. Yan, S. Xu, W. Yang, *et al.*, "Image recoloring based on fast and flexible palette extraction," *Multimedia Tools and Applications*, vol. 82, pp. 47 793–47 810, 2023. DOI: 10.1007/s11042-023-15114-5.

[12] N. Ibraheem, M. Hasan, R. Z. Khan, and P. Mishra, "Understanding color models: A review," *ARPN Journal of Science and Technology*, vol. 2, Jan. 2012.

[13] J. Lv and J. Fang, "A color distance model based on visual recognition," *Mathematical Problems in Engineering*, vol. 2018, no. 1, p. 4 652 526, 2018. DOI: https://doi.org/10.1155/2018/4652526. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1155/2018/4652526.

[14] D. Starov, *Perceptual color models*, Accessed: 2025-06-10, 2025. [Online]. Available: https://chromatone.center/theory/color/models/perceptual/.

[15] Les Forges, *Chapter 5: Color palettes*, https://opengameart.org/content/chapter-5-color-palettes, Accessed: 2025-06-02, 2023.

[16] A. N. Jansson. "Pixel art tutorial." (2010), [Online]. Available: https://androidarts.com/pixtut/pixelart.htm (visited on 05/02/2025).

[17] J. Delon, A. Desolneux, J.-L. Lisani, and A.-B. Petro, "Automatic color palette," vol. 2, Jan. 2005, pp. 706–709. DOI: 10.1109/ICIP.2005.1530153.

[18] M.-R. Huang and R.-R. Lee, "Pixel art color palette synthesis," in *Information Science and Applications*, K. J. Kim, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 327–334, ISBN: 978-3-662-46578-3.

[19] G. Greenfield and D. House, "Image recoloring induced by palette color associations.," vol. 11, Jan. 2003.

[20] H. Chang, O. Fried, Y. Liu, S. DiVerdi, and A. Finkelstein, "Palette-based photo recoloring," *ACM Transactions on Graphics (Proc. SIGGRAPH)*, vol. 34, no. 4, Jul. 2015.

[21] D. Silber, *Pixel Art for Game Developers*. CRC Press, 2015.

[22] F. J. Damerau, "A technique for computer detection and correction of spelling errors," *Commun. ACM*, vol. 7, no. 3, pp. 171–176, Mar. 1964, ISSN: 0001-0782.

[23] S. Minaker, R. Mason, and D. Chow, "Optimizing color performance of the ngenuity® 3d visualization system," *Ophthalmology Science*, vol. 1, p. 100 054, Aug. 2021. DOI: 10.1016/j.xops.2021.100054.

[24] Jimison3, *Harvest Palette*, https://lospec.com/palette-list / harvest, Accessed: 2025-06-22, Lospec, 2025. [Online]. Available: https://lospec.com/palette-list/harvest.

A Use of AI Coding Assistant

During development, an **AI-powered coding assistant** (a plugin for PyCharm) was used to support the implementation of the tool. The assistant provided help with debugging, refactoring, documentation, and graphical user interface (GUI) development. Its use was limited to practical software engineering tasks and did not influence the research design, methodology, or results.

Several built-in **AI Actions** were applied directly to selected code snippets using the plugin's options. These included:

- **Explain Code** – to better understand broken logic.
- **Suggest Refactoring** – to restructure code for improved readability and maintainability.
- **Find Problems** – to detect bugs and code smells.
- **Write Documentation** – to generate docstrings and brief explanations.

These actions required no written prompts and were used to improve the quality of the codebase.

The assistant's chat interface, based on the Claude 3.5 Sonnet model, was also used to support GUI development and code troubleshooting. It had access to the project codebase and was queried through natural language prompts. The interaction focused on practical implementation support and UI behavior refinement. The used prompts include:

- *"I created a project called PixelArtColorProcessor. It only contains .venv for now. How do I create the basic UI and run it using Qt Designer in PyCharm?"*
- *"The image viewer makes the pixel art blurry. I would like to either display it in actual size or scale it in a way that doesn't blur it."*
- *"Give me code that allows the user to select zoom factor using slider and doesn't blur images."*
- *"Highlight all pixels with the given color when the user hovers over a pixel or a color in the palette."*
- *"Now, instead of displaying a box with color info on hover, I would like to have a fixed place on the screen that displays the hovered color and its information (RGB, hex, HSV)."*
- *"With no scaling, the image position is sometimes in the top left corner, sometimes in the middle."*
- *"How do I make palette color squares bigger? I would like constant spacing that would make them stick together."*
- *"I want the min and max labels to be above their sliders and show current value."*
- *"I want the hue, saturation, value labels to be vertical before each box instead of above."*
- *"Can I add a vertically scrollable container for ramps?"*
- *"Make it so the ramp squares don't have outlines and they are completely stuck to each other."*
- *"I would like to add a dropdown that allows me to switch between color extraction mechanisms."*
- *"I also implemented the graph creation, and ramp sampling algorithm. Now I would like to add a dropdown on top of the top right section to allow selecting a different graph generation method."*
- *"I would like to clean up the codebase to make it more readable and maintainable. I would like to start by defining a separate widget for each of the 4 regions in the ramp extraction window."*
- *"Zooming still makes the window grow."*
- *"I would like to add a box around the color details overlay with white background and allow the user to copy the color values."*
- *"I would like to disable the ramp extraction button when there are no nodes in the graph."*
- *"Instead of printing the ramp extraction updates into the console, I would like to display a progress bar over the ramp extraction window."*

- *"How do I align the control column contents to the top?"*
- *"I want the graph type label to hug the text horizontally and the selector to fill the rest of the space."*
- *"I want to remove the padding from the control panel and reduce spacing inside the HSV slider boxes."*
- *"Above the tool buttons I want a small rectangle filled with the currently selected color and 'No color' label if none is selected."*
- *"I changed my mind, I want hover to override selected. But for some reason hover style is applied to selected color if I move my mouse outside image."*
- *"The zoom steps are now very weak. I want 20 to mean zoom 1x and 40 to mean 4x."*
- *"Can I add margin to the right of the slider so it's centered again?"*
- *"I would like the color square to be a bit bigger and the RGB, HSV and hex to be below each other."*
- *"I would like the user to be able to override the threshold but how should I decide the initial one?"*
- *"There should be a new section to the right of the image viewer in the main scene where the resulting ramps will eventually be displayed."*
- *"In top left, a smaller version of the image viewer should be visible with all the functionality including the unique color palette, scaling, highlighting, selecting and displaying color details."*
- *"The graph should be displayed in the top right corner of the new scene."*
- *"I added the dropdown at the top of the section. But it drops to the bottom when graph is generated for second time."*
- *"That doesn't help, I need a persistent container for graph that will remain inside controls, below the dropdown."*
- *"I would like to keep the method selector for color similarity graph as well but change the options to HSV or CIEDE2000."*
- *"The image label should fill all space in the widget but it shouldn't make the widget grow if the image overflows due to scaling."*
- *"Instead of two checkboxes below each other, I want one row starting with label Remove: and then checkboxes Reverses, Subsequences."*
- *"I would like a checkbox for removing Similar Ramps."*
- *"I would like to add a slider to the general part of the ramp extraction controls to limit the max length of a ramp with 3–20 range."*
- *"The buttons seem really small."*
- *"If I close and reopen ramp extraction window I get RuntimeError."*
- *"MouseMoveEvent doesn't seem to be triggered."*
- *"I want to put the graph into a box too, but the graph holder doesn't seem to fill it in."*
- *"I would also like to highlight the whole row of the hovered color ramp."*
- *"I would like broader zoom scale that also allows me to zoom out."*
- *"How can I make the width of the zoom label constant?"*
- *"I would like to add some space between method selection dropdown line and the controls and reduce spacing between rows inside hue, value, saturation boxes."*
- *"The background doesn't seem to be set, it's white."*
- *"The _setup_ui seems to have a lot of unresolved attributes, check again."*
- *"I want special color highlight just for the hovered color in ramp if the tool is active."*
- *"Instead of displaying box with color info on hover, I want a fixed location showing RGB, HSV, and hex values."*
- *"Can I put the hue, saturation, value labels vertically before each box instead of above?"*
- *"The factor label column is unnecessarily wide, I want it to hug the labels."*
- *"The ramp container and the controls panel should be next to each other."*
- *"I want the selector to fill the remaining space beside the graph type label."*
- *"If the graph has no nodes, the extract ramps button should be disabled."*
- *"Instead of printing ramp extraction updates into the console, I want a progress bar over the entire ramp extraction window."*
- *"The progress bar should be centered and overlay the whole ramp extraction panel."*
- *"I would like to add a zoom control with a slider that doesn't blur images and keeps proper pixel scaling."*
- *"I want hover styles to override selection and reset correctly when the mouse leaves the image area."*
- *"Make all graph containers persistent even if the graph is regenerated."*
- *"The control panel contents should be aligned to the top."*

B Full Evaluation Dataset

This appendix contains the complete evaluation dataset used in this project. It includes all ten graph construction results (Figures 9 and 10) used for ramp detection experiments, as well as the six final ramp extraction results selected for analysis (Figure 11). Each graph result is presented with its relevant parameters, graph type, and edge statistics. Each ramp extraction result includes the candidate ramps, filtered results, and final selected ramps. Additionally, all parameter values used in the extraction pipeline are listed in full for each test case (Figure 12). The dataset is provided to support transparency, reproducibility, and further inspection of the tool's behavior across diverse pixel art examples.

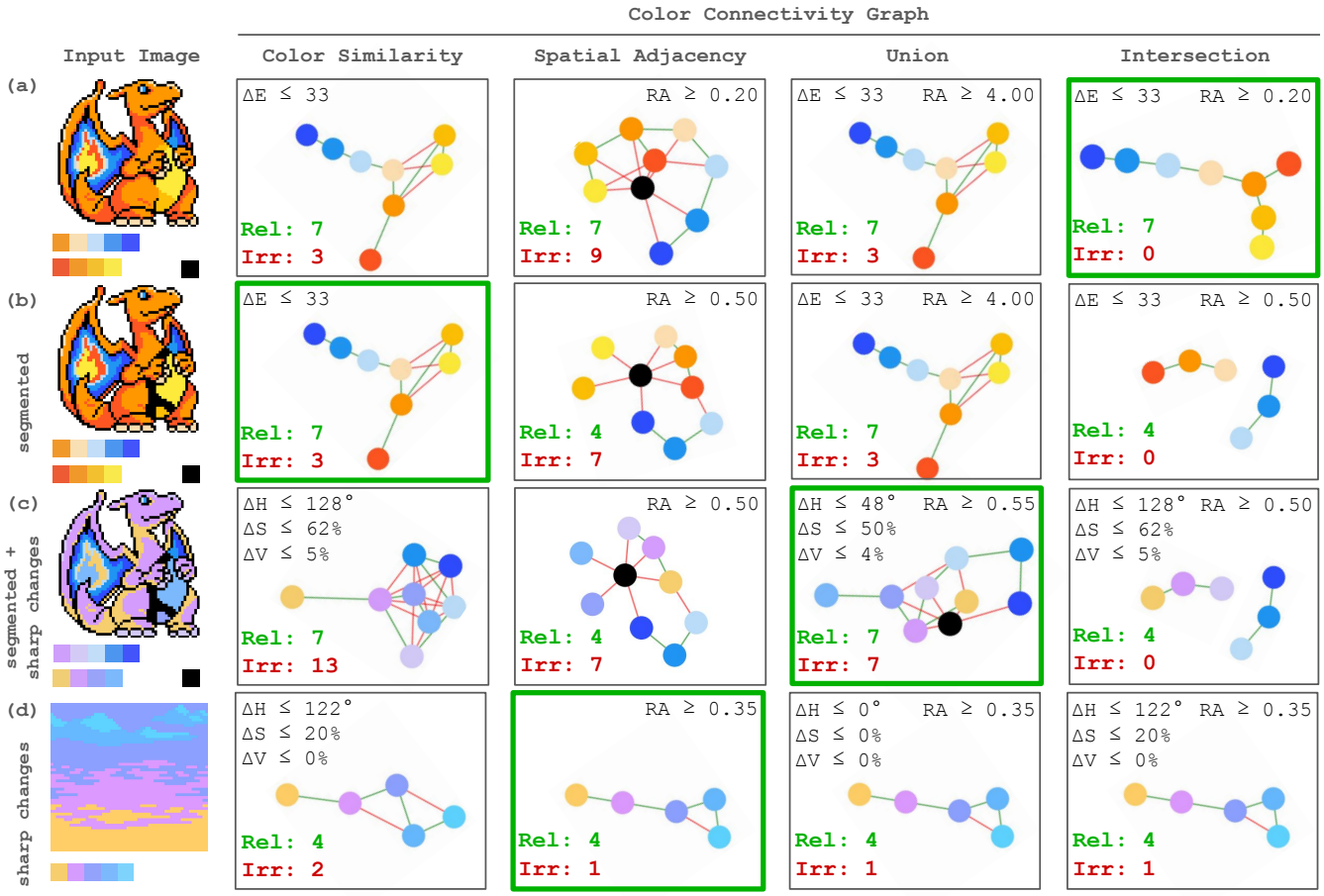

Figure 9: Graph construction results for four representative test images (reproduced from the main report). Each subfigure shows the best-performing graph for a given image (green), along with relevant and irrelevant edge counts. Graph types include color similarity, spatial adjacency, and hybrid strategies.

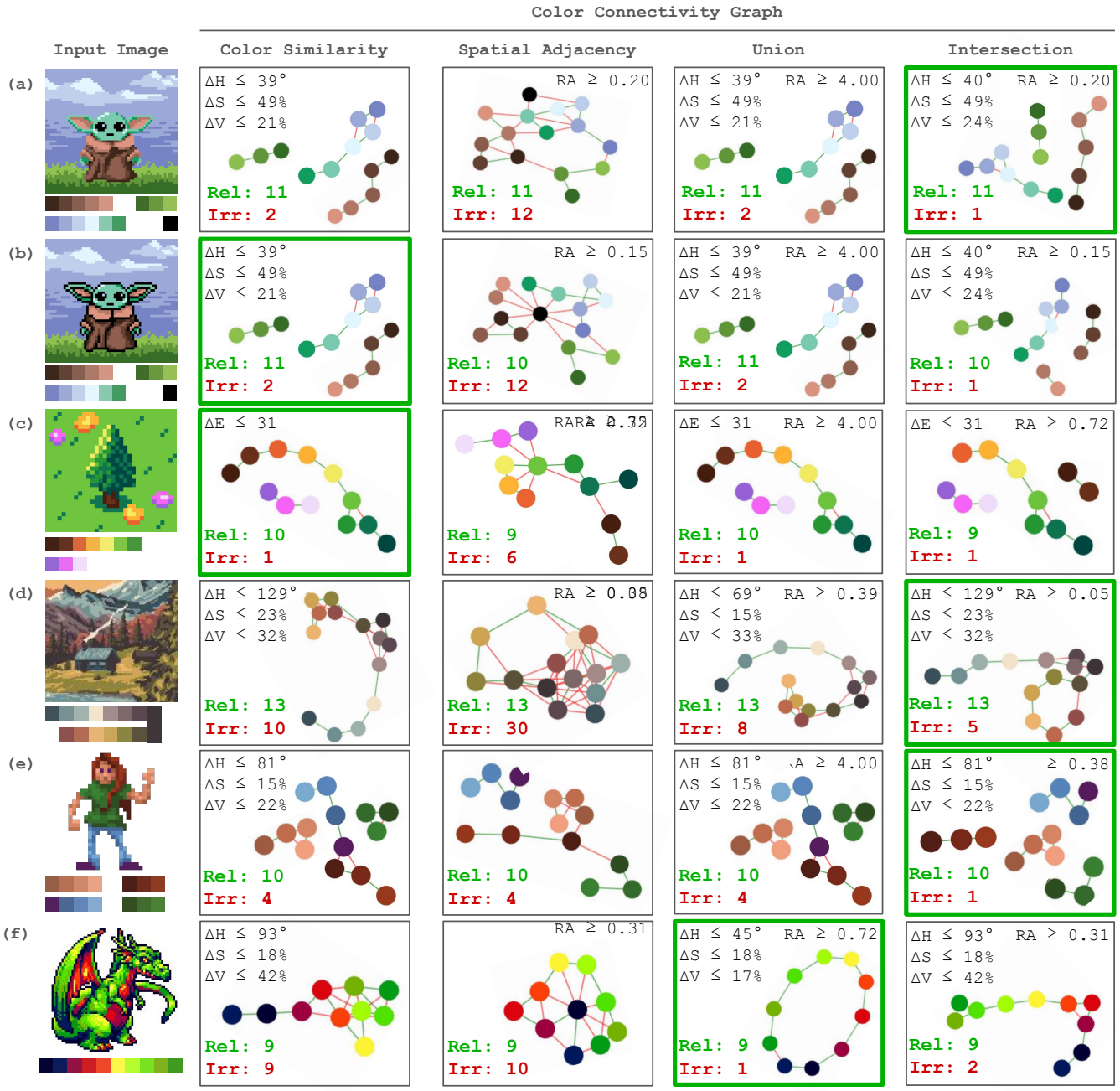

Figure 10: Graph construction results for the remaining six test cases that are not included in the original report. Each subfigure shows the best-performing graph for a given image (green), along with relevant and irrelevant edge counts. Graph types include color similarity, spatial adjacency, and hybrid strategies.

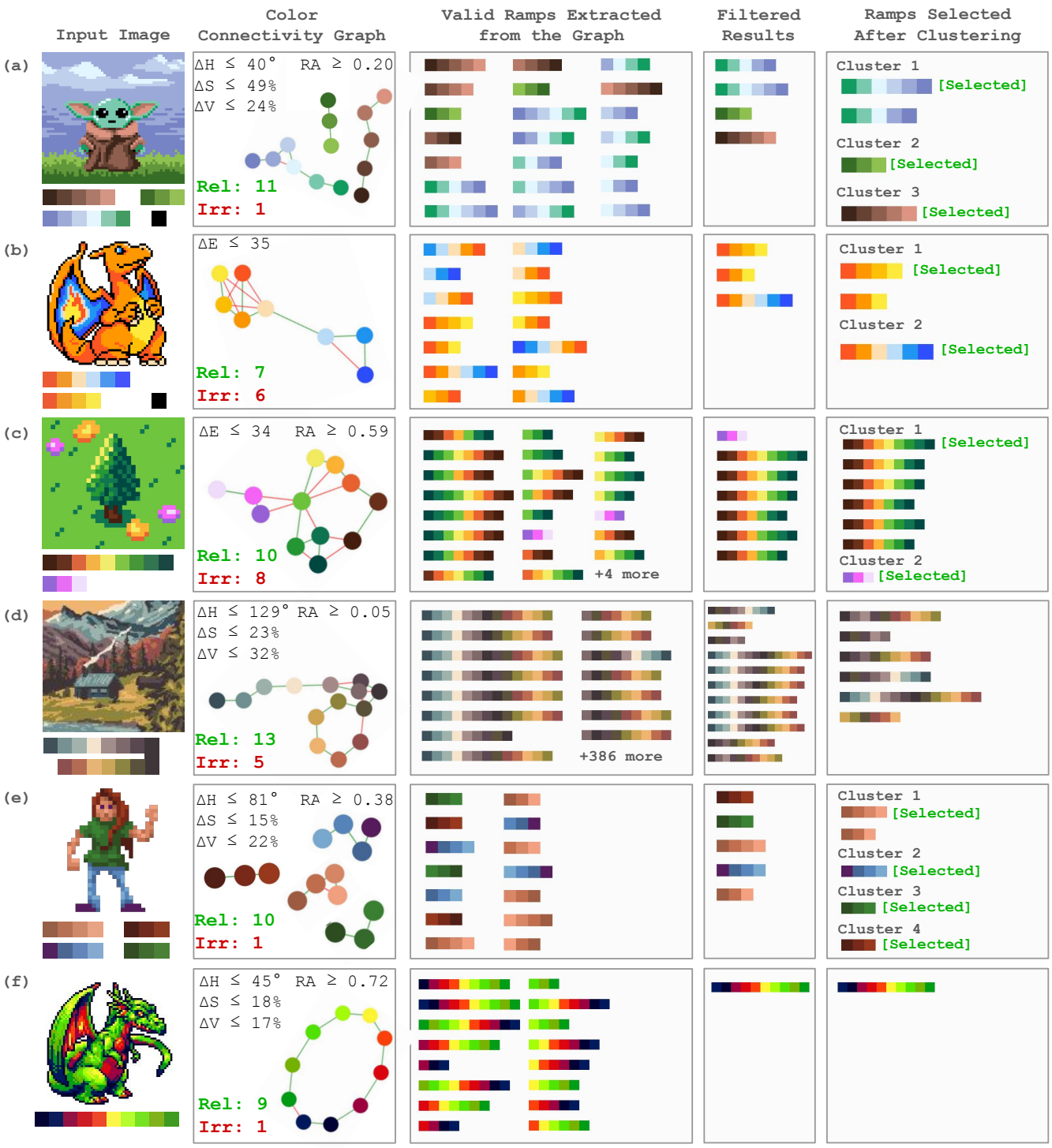

Figure 11: Ramp extraction results for six structurally diverse images. For each case, the figure includes candidate ramps from the graph, filtered results after redundancy removal, and the final selected ramps following clustering.

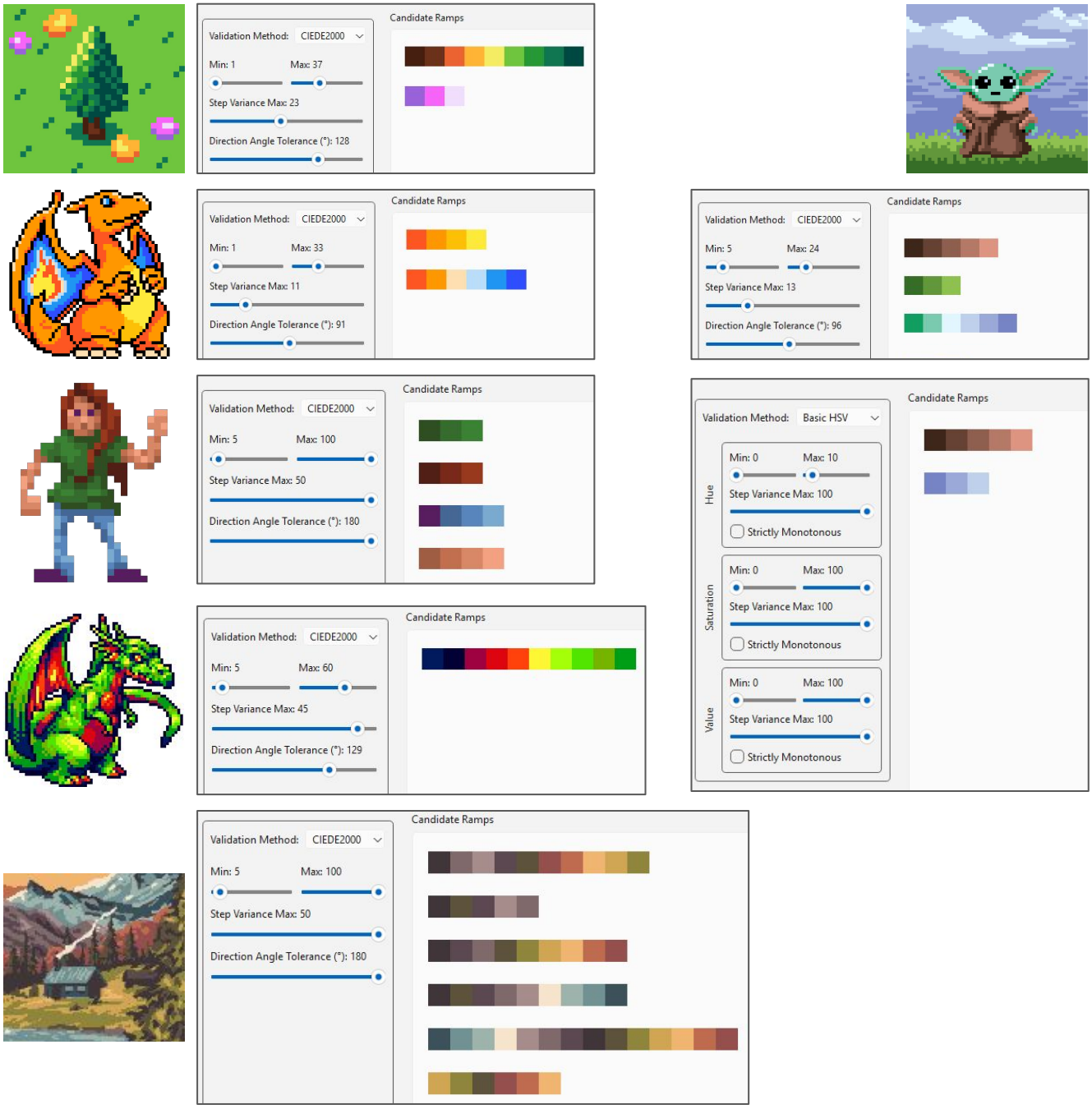

Figure 12: Validation parameter values used during extraction and evaluation for all six test images.