# Representing Images with Connected Spanning Trees

Mathewe Banda and Declan Galleher

## Abstract

Given a greyscale image, we construct connected spanning trees using king's and knight's moves on a rectangular grid that visually approximate the original image.

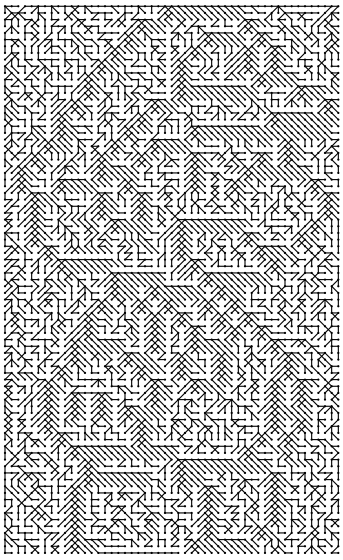

**Figure 1**: Image of the Nintendo character Mario, rendered using king's moves.

## 1 Introduction

This project draws on Robert Bosch's earlier work using spanning trees to re-create images, extending it with a heuristic-driven approach rather than a full optimization. Bosch's grid places vertices at the corners of each square, connected by edges formed from knight's and king's moves. Our variation instead places vertices at the center of every square, using the same move types. This simplifies edge calculations and lets us associate the brightness $0$–$255$ of each pixel with a single vertex rather than several. With a straightforward rule for connecting vertices, we produce recognizable images that satisfy the constraints of a connected spanning tree.

## 2 Construction

Our implementation uses Python 3 to produce `.eps` files for the tree visualizations. We begin with an $x \times y$ pixel image, convert it to greyscale, and scale the dimensions to fit a desired square size $n$.

The image is then represented as an $\frac{x}{n} \times \frac{y}{n}$ 2-D array of vertices called a "grid", where each vertex corresponds to a square object.

Each square stores the average pixel brightness $c$ of an $n \times n$ region of the original greyscale photo, the $x$–$y$ coordinates of its center (representing a vertex), the identities of its neighbors (via knight's and king's moves), and which neighbor has the lowest brightness value (the darkest neighbor).

Once all squares are created and their brightness values computed, we iterate through each square to identify its neighbors and determine the darkest one. Each square also carries a boolean `isConnected` flag, used by the tree creation method described below.

## 3 Tree Creation Method

To form trees in our grid, we use the following algorithm:

```
for each row in the grid, top to bottom do
    for each square s in the row, left to right do
        if s is the first square then
            s.isConnected ← true
            s.darkestNeighbor.isConnected ← true
            draw s and an edge to s.darkestNeighbor
        else if s.isConnected then
            s.darkestNeighbor.isConnected ← true
            draw s and an edge to s.darkestNeighbor
        else
            find the darkest connected neighbor of s
            s.isConnected ← true
            s.darkestConnectedNeighbor.isConnected ← true
            draw s and an edge to s.darkestConnectedNeighbor
        end
    end
end
```

**Algorithm 1:** Heuristic algorithm for creating a tree.

The correctness of this heuristic follows from a basic property of trees: attaching a new vertex to an existing tree with a single edge preserves the tree structure. Because we always connect to a vertex already in the tree—iteratively building along a Hamiltonian path on the grid—we never leave a vertex isolated, never introduce a cycle, and never create a disconnected component. The result is guaranteed to be a connected spanning tree.

Since we always prefer the darkest available connected neighbor, darker squares accumulate more incident edges than brighter ones. Each vertex connects to exactly one other vertex, so heavily connected regions correspond to dark areas of the source image.

This local heuristic turns out to produce surprisingly faithful visual approximations of the underlying black and white images.

**Figure 2**: A photograph of a tree.

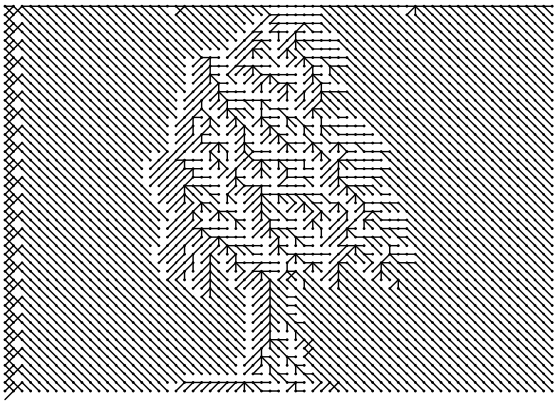

**Figure 3**: Spanning tree using the heuristic algorithm and king's moves.

## 4 Tree Variations

The initial version of the algorithm performed well on high-contrast images, clearly outlining shapes. However, regions of differing brightness tended to look uniform—distinguishable only by their outlines rather than their tone. The checkerboard in Figure 4 is an extreme case: the heuristic produces a nearly uniform edge distribution across both light and dark areas (Figure 5).

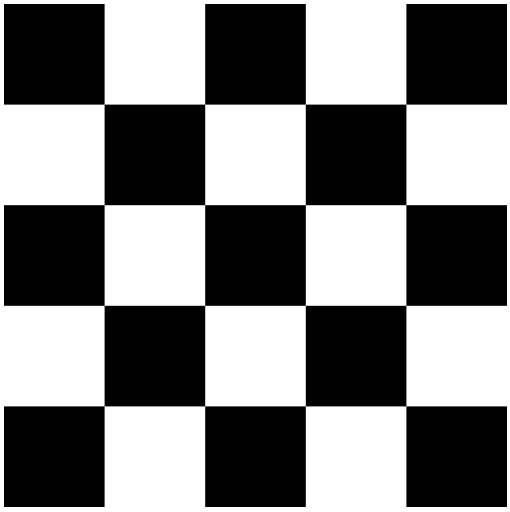

**Figure 4**: Checkerboard source image.

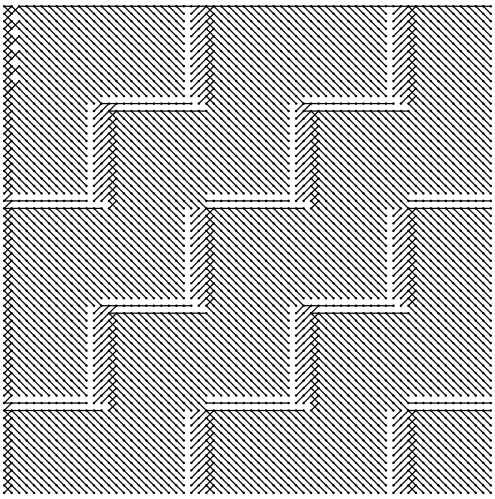

**Figure 5**: Checkerboard using the heuristic algorithm and king's moves.

To address this, we introduced *thresholding* (variant 1). When selecting the darkest neighbor for a square, we check whether that neighbor's brightness exceeds a threshold (by default, $\frac{255}{2}$). If it does, we randomize the choice of neighbor; if not, we keep the darkest. The idea is to make bright regions appear visually disorganized while keeping dark regions structured and dense. As Figure 6 shows, this works particularly well on high-contrast images like the checkerboard.

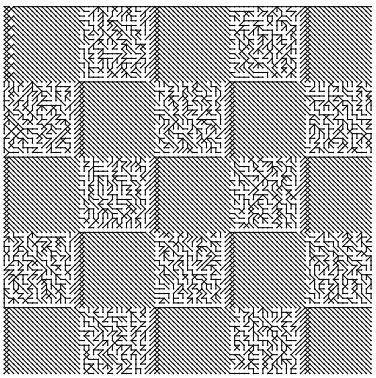

**Figure 6**: Checkerboard using thresholding (variant 1) and king's moves.

A further refinement (variant 2) produced even more compelling results. Rather than randomizing above a threshold, we divide the full brightness range $0$–$255$ into $k$ sections, where $k$ is the total number of moves available to a vertex (up to 8 king's moves plus 4 knight's moves). A square's brightness then determines which specific move its vertex takes. This creates distinct visual "textures" within the tree, allowing the viewer to read differences in tone through differences in local edge pattern, not just edge density. The checkerboard in Figure 7 is clearly rendered without any randomization, purely through deterministic move assignment.

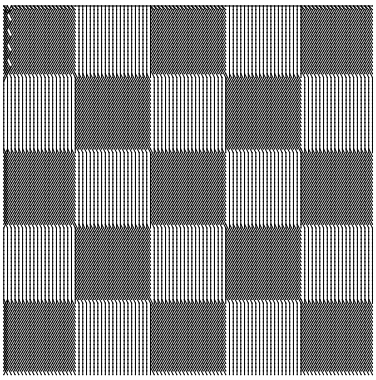

**Figure 7**: Checkerboard using thresholding (variant 2) with knight's and king's moves.

We also experimented with different grid traversal orders. The default top-to-bottom, left-to-right pass introduced directional artifacts. We tried several alternatives—a Hamiltonian spiral, alternating left-to-right and right-to-left rows—and found that traversing along the grid's diagonals (starting from the top-left corner and working toward the bottom-right) produced the cleanest, most artifact-free results. All examples from this point onward use diagonal traversal.

## 5 Future Work

We would like to explore adding RGB color to the edges to see whether chromatic variation can enrich the visual results beyond greyscale.

We are also interested in traversing random Hamiltonian paths during tree construction to understand how path choice affects the visual character of the final tree.

## 6 Acknowledgements

This project grew out of ideas explored in Robert Bosch's work on mathematical art and optimization. His research on tree-based image representations was the foundation for this work, and his guidance throughout the project was invaluable.