[[1]](#footnote-1)

Detecting Vanishing Points and Horizon Lines in Photographs

M. Kalra

*Abstract*—This paper presents an experimental method for detecting perspective lines, vanishing points and horizon lines in images with strong perspectival features as a tool for artists, and evaluates the results.

*Index Terms*—Hough transform, Gaussian Sphere, Perspective lines, Horizon lines, Vanishing points

# INTRODUCTION

T

he 2-Dimensional image of 3-Dimensional space mimics dimensionality by mapping all registered objects into perspectival space. The vectors of any parallel lines in the image converge at a distance infinity from the image plane at a point called the Vanishing Point as a result of their transformation into perspectival space. For each set of parallel perspective lines, there is a corresponding vanishing point. The vanishing points that originate horizontal perspective lines in the image are collinear on a line called the Horizon Line. In an image, this would naturally be where the horizon would be if one were present in the image space.

When illustrators and artists are rendering 2-D scenes in perspectival, the location of the space’s horizon line and horizontal vanishing points are paramount as nearly all resulting lines rendered will converge at one of these vanishing points. The location of the horizon line, likewise, corresponds to the viewing angle of the scene and the vantage point of the “camera” viewing the scene. Scenes with low horizon lines correspond to an image taken from ground level, while scenes with high horizon lines correlate with aerial or overhead shots. As such, an understanding of perspectival space and the kinds of images that result from different horizon line and vanishing point placements are crucial for artists.

Outlined in this paper is an experimental tool designed to help artist familiarize themselves with the constructions of perspectival space. The most common perspectival images artists will render are 1-point perspective and 2-point perspective images, describing the respective number of horizontal vanishing points present in the image. As such, the methods described in this paper will detect vertical and horizontal perspective lines in an input image using a Hough Transform. The detected horizon lines are then projected onto a Gaussian sphere as proposed by S. Barnard [1] and a primary horizontal vanishing point is determined. Then a horizon line is extracted from a vertical vanishing point calculated form vertical perspective lines detected. This horizon line is used to then detect if there is a second prominent horizontal vanishing point in the image. In this manner, vanishing points and horizon lines for 1 and 2-point perspective images can be extracted for artists looking to train their perspectival eye and reverse engineer the perspectival laws governing a pre-determined image.

# Methods

For this investigation, nearly all input images used were sourced from an J. Tighe and S. Lazebnik’s Barcelona dataset of street and interior images [2]. Initial test images were chosen for their presence of strong perspective lines and minimal other features, like trees or cars or furniture that might be erroneously detected as perspective lines. All of these images were RGB 480x640 pixel images.

Given an input image, the image first undergoes pre-processing where the image is converted to greyscale and a 5x5 median filter is passed over the image to consolidate and smooth edges while suppressing unnecessary details.

## Extracting Perspective Lines

The image, after undergoing pre-processing, is passed into Matlab’s *edges* function to generate an edge map using the Canny method. This edge map is then built into a custom *houghfunc* which performs a Hough Transform on the edge map as described by R. Duda and P. Hart [3]. An accumulator array of dimensions 181 by (2R+1) where R is the length of the input image’s diagonal is allocated. The *y* dimension of the array spans and the *x* dimension spans . For each possible value of measured from the x-axis of the image, the function parses through every pixel in the edge map and checks to see if the pixel has an intensity of 1, marking an edge. For each edge pixel, there is a line that passes through the pixel whose normal has an angle of drawn from the origin. The length of a line segment with this normal vector connecting the origin and the line passing through the pixel is where:

(1)

is calculated and the cell in the accumulator array that corresponds to and is incremented. As the function progresses, lines described by some and that pass through the most edge pixels will accumulate the highest accumulator values.

The accumulator cells are then thresholded to extract cells with the highest accumulator values and each line is then masked over the edge map. The length of the largest line segment on the edge map that overlaps with each masking line is then saved in the accumulator’s cell for that line, overwriting the previous value. Gaps in line segments less than 5 pixels in length are ignored and segments are considered to end when a discontinuity greater than 5 pixels occurs. The longer a segment, the more reliable it is as a basis of the image’s perspectival space.

The accumulator array now holds the length of the longest segment in the image for each line present in the image. This accumulator is then returned by *houghfunc.*

## 

## FIGURE 1. Input image with extracted horizontal perspective lines. Corresponds to Image 8 in Table 1.

## Extracting the Horizontal Vanishing Point

Given the filled accumulator cell, the accumulator values are sorted to extract the lines with the highest 25% of accumulator values as these will be the most reliable to base analysis of off. In order to determine the horizontal vanishing points, only the intersection of non-vertical lines will be analyzed. For each where (corresponding to lines in the image between and ), the line with the highest accumulator value is plotted on the input image as shown in Figure 1 and the *x* and *y* values of this line extending from *x* = -3000 to *x* = 3000 are saved. By only extracting the line with the highest accumulator value for each , noise in the resulting analysis is reduced. These extracted lines and their *x* and *y* values are then passed into a function called *sphereMap*.

*sphereMap* maps each line onto a great circle in a Gaussian sphere as discussed by S. Barnard [1]. As Barnard describes, to accurately find vanishing points in an image, lines must be mapped onto a finite surface so that, even ostensibly parallel lines will intersect. In this case, the finite surface is the surface of a Gaussian sphere centered at the origin of cartesian space. At a distance *f* along the z-axis is the input image, with its upper right corner at *x* = 0and *y* = 0. The variable *f* represents the focal length of the camera taking the image, though in reality, as L. Quan and R. Mohr attest, *f* can be any arbitrary constant so long as it remains consistent throughout one’s analysis [4].

Given a line in the image plane with two points *a* and *b,* a plane can be formed between *a, b* and the origin. This plane then cuts through the Gaussian sphere, defining a great circle. The intersection between great circles mapped from perspective lines in the image plane defines the vanishing point.

For each line passed into *sphereMap*, two points on the line at *x* = 30 and *x* = 300 are extracted and the origin *o* is set to be (0, 0, -800). The image plane is set to be at *z* = 0. The unit normal *n* of the plane formed by these three points is defined as:

(2)

Where and each respectively represent the azimuth and the elevation of a point on the sphere, the great circle resulting from each line can be defined by:

(3)

A second sphere accumulator array with dimensions of 360 x 181 pixels is thus created whose *y* dimension describes values of ranging from to and whose *x* dimension describes values of ranging from to . For each line and for each possible , values of can be calculated with:

(4)

The sphere accumulator cell corresponding to and can be incremented to mark pixels on the path of each line’s great circle. Additionally, the four neighbor cells are incremented to account for imprecision in the perspective lines.

The sphere accumulator’s values are then sorted and the cells corresponding to the 4 highest values are extracted. These correspond to points on the sphere where the highest number of great circles intersect. Each extracted point is then projected back onto the image plane from the Gaussian sphere using the following two equations where *f* was 800 [4]:

(5)

(6)

If a point in the image plane is within 100 pixels of another extracted points, then the two points are geometrically averaged using their accumulator cell values as weights and their weights are then added together as the new point’s weight. In this manner, any imprecise lines used to produce the extracted vanishing points are suppressed. The extracted and averaged points are returned by *sphereMap.* The point with the highest final weight is determined to be a horizontal vanishing point (if the image has only 1- point perspective it will be the sole horizontal vanishing point). *sphereMap* also returns the unmodified sphere accumulator array to be used in D. below.

## Extracting the Horizon Line

As Barnard discusses, every vanishing point found on the Gaussian sphere is the pole for a great circle normal to the surface vector at that vanishing point. This great circle is a horizon line and is the *dual* of the vanishing point. The dual of a horizontal vanishing point is a vertical horizon line that passes through the vertical vanishing points. Likewise, the dual of a vertical vanishing point will be the horizontal horizon line which we seek to extract.

As such, the vertical vanishing point of the image is found in the same manner that the horizontal vanishing point in B. was extracted, except the lines passed into *sphereMap* are lines extracted from *houghfunc* corresponding to . These are the predominantly vertical lines in the image and will consequently converge at the vertical vanishing point rather than the horizontal vanishing point as, in the image space, these lines are likely not parallel to the horizontal lines extracted in B.

Once the vertical vanishing point is extracted, this point is passed into a *dual* function. Once again, an origin *o* at (0, 0, -800) is set in 3-dimensional cartesian space and the image plane is set at a *z* value of 0. The unit vector describing the line from *o* to the vertical vanishing point is calculated and this vector is passed as ***n*** into (4) to determine the great circle dual of the vertical vanishing point on a Gaussian sphere centered at *o*. The great circle is then projected back onto the image plane using (5) and (6) to produce the desired horizon line.

Experimentally, it was found that, due to imprecisions in the vertical lines used to determine the vertical vanishing point, the output horizon line would be offset by 100 to 300 pixels from the horizontal vanishing point found in B. As such, the horizon line is translated vertically to coincide with the calculated horizontal vanishing point.

## Extracting Second Horizontal Vanishing Point

If the image has 2-point perspective, only one point will have been extracted in B. Additionally, many images may lack enough strong perspective

lines to extract the second horizontal vanishing point using the methods described in B. By the laws of perspective, the second horizontal vanishing point will have to be coincident with the horizon line calculated in C. In order to calculate the location of the second horizontal, the calculated horizon line and horizontal vanishing point from C. and B. respectively as well as the sphere accumulator array from *sphereMap* are passed into a function *extraPoints*.

*extraPoints* then parses through the points of intersection in the accumulator array with the highest 50% of values in the array. If the point is within 50 pixels of the horizon line and not within 200 pixels of the originally detected horizontal vanishing point, then the point is projected back onto the image plane using (5) and (6). If the point in the image plane is within 200 pixels of another extracted point in the image plane, then the two points are geometrically averaged using their accumulator cell values as weights and their weights are then added together as the new point’s weight. The point in the image plane with the highest final weight is determined to be the second horizontal vanishing point and is returned by *extraPoints.* If no points are found, then *extraPoints* returns an empty array and the image is assumed to have only 1 horizontal vanishing point.

## Evaluation

The calculated horizon line, horizontal vanishing points and vertical vanishing point are plotted on the original image along with particularly strong perspective lines as shown in Figure 2.

In order to evaluate the effectiveness of the methods described above, the accuracy of the horizontal vanishing points and horizon line will be measured.

To determine measure this accuracy, a ground truth containing manually generated horizon lines and vanishing points will be created for each input image. To determine the accuracy of each vanishing point, the distance *d* between the measured point and the ground truth will be calculated. Error will then be calculated as follows where *R* is the length of the input image’s diagonal:

(7)

The error will be calculated separately for the first vanishing point found through II.B. and the second vanishing point found in II.D. if applicable. To determine the accuracy of the calculated horizon line, the degree difference in angle between the calculated line and the ground truth will be found as follows:

(8)

This procedure will be performed for ten images chosen for their strong perspectival features, and an average error for each feature will be extracted from these tests. Five of the images will be of indoor scenes and five will be of outdoor scenes and five of the images will have 1-point perspective and five will have 2-point perspective.

# Results

The results of the tests are shown below in Table 1 divided by the kind of perspective in the image and



## FIGURE 2. Input image with extracted horizontal perspective lines, horizon line in blue, first horizontal vanishing point in red and second horizontal vanishing point marked in blue. Corresponds to Image 4 in Table 1.

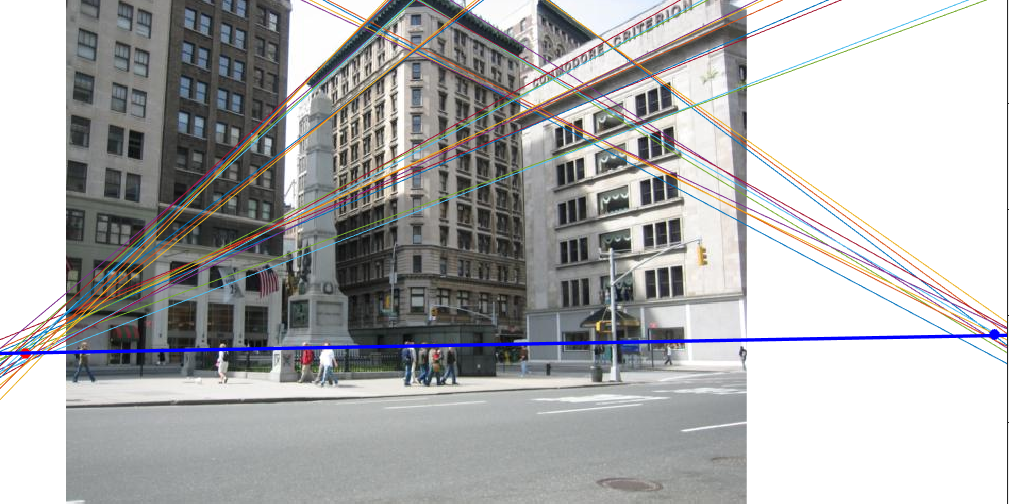
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Image |  |  |  |
| Outdoors | 1 | 4.7% | 5.9% | 1.6% |
| 2 | 0.075% | N/A | 3.1% |
| 3 | 0.51% | N/A | 1.1% |
| 4 | 2.2% | 1.3% | 1.1% |
| 5 | 0.78% | 134% | 3.9% |
| Indoors | 6 | 2.1% | 24.5% | 3.3% |
| 7 | 0.86% | N/A | 4.4% |
| 8 | 0.75% | N/A | 2.7% |
| 9 | 2.1% | 123% | 1.1% |
| 10 | 1.4% | N/A | 2.2% |
|  | Overall | 1.6% | 57% | 2.5% |

## TABLE 1. Error results for each image, separated by whether the image was indoors or outdoors. If the image has 1-point perspective, then the error for Vanishing Point 2 is marked “N/A”.

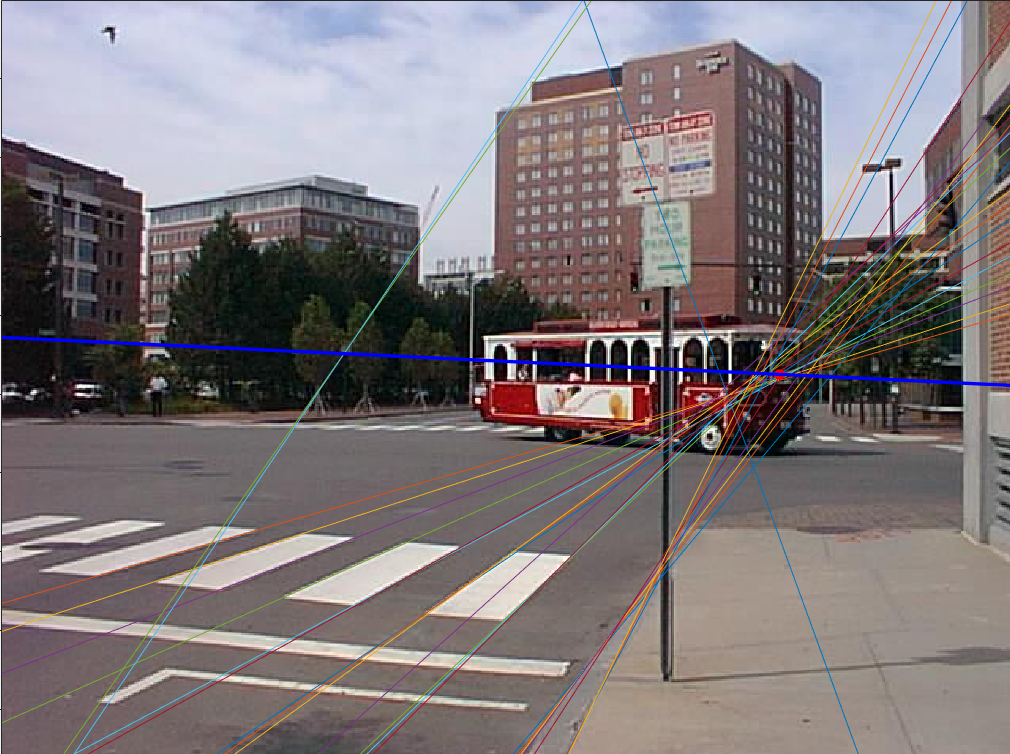
by whether or not the image is indoors or outdoors. The error rate for the primary vanishing point as detected in II.B. was 1.6% while the error rate for the second vanishing point, if there was one, was 57% which was significantly greater. It should be noted that this error rate is skewed by 2 misidentifications of the second vanishing point. When the second vanishing point was identified in the correct region of the image, the error rate was a lesser 10.6%. The overall horizon line error rate was found to be 2.5%. Figure 3 depicts four of the algorithm results.

# Discussion

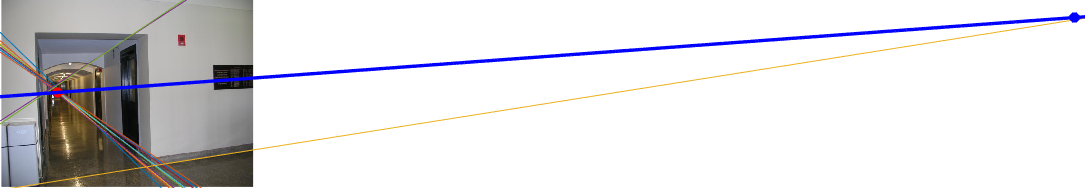
## My algorithm ultimately was very effective at detecting the primary vanishing point in an image (wherever the greatest number of horizontal lines pointed). Detecting horizon lines was quite accurate as well but highly influenced by imprecision in the vectors for vertical lines detected in the image. As such, this imprecision compounded when attempting to detect second vanishing points in the image.

**

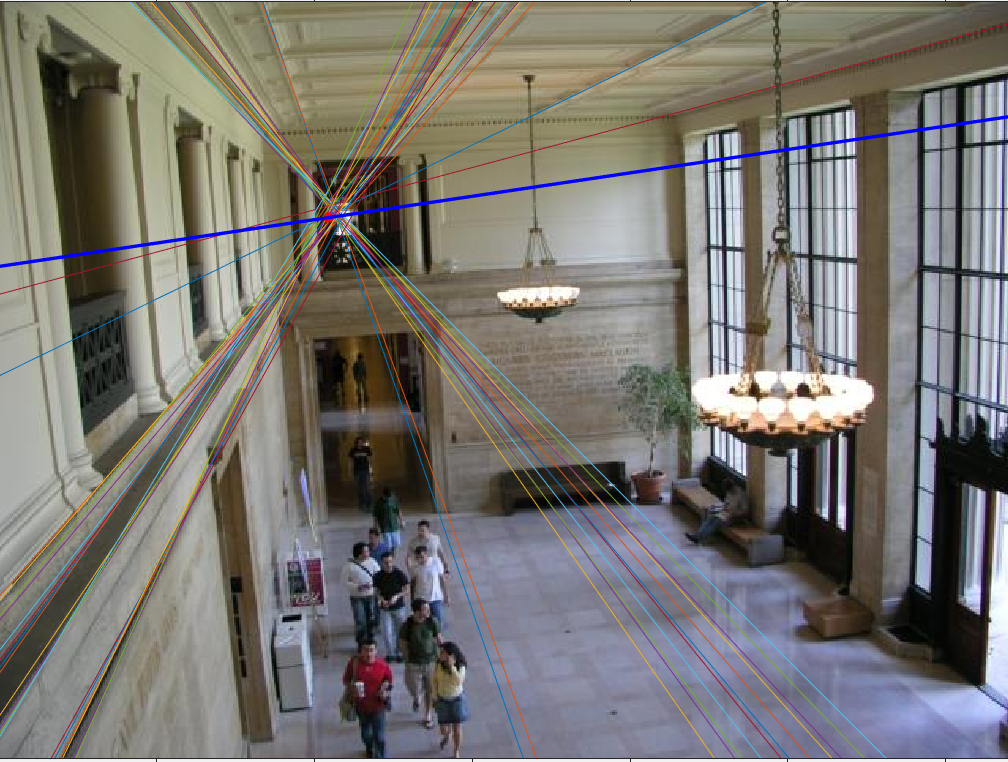
(a)



(b)

**

(c)

**

(d)

FIGURE 3. Horizon Line and Vanishing Point detection in images 1, 2, 6 and 7 from Table 1 shown in a), b), c) and d) respectively.

## If the second vanishing point lacked a preponderance of perspective lines pointing in its direction, the algorithm would easily become confused by prominent lines arising from errant objects in the scene like those stemming from the support of the center traffic light in Figure 4 and would mark vanishing points in incorrect regions of the image. These lines were particularly troublesome when foliage and cars were present in the scene. Additionally, any strong lines in an image, like the tip of a house’s peaked roof would also confuse the algorithm. As such, indoor scenes were much more accurate due to the lack of objects that would distract the algorithm.

## The algorithm is also limited in that it relies on the fact that the image has not been greatly rotated. As such, it is assumed that verticals in the image space also align with approximately vertical lines in the cartesian space in which the algorithm operates. If an input image were rotated greater than 20, then the algorithm would no longer be able to distinguish the vertical perspective lines from horizontal ones and the resulting vanishing points would be skewed by vertical perspective lines present in the calculation.

Further, as Long and Mohr discuss in their paper, there is an inaccuracy to the mapping of lines onto Barnard’s Gaussian sphere [4]. The sphere accumulator, which delineates cells according to their azimuth and elevation values, does not equally divide the sphere. As such, some cells represent larger areas of the sphere and larger areas of intersection between great circles than other cells. This contributes to inaccuracy in the resulting vanishing points and further work would follow

algorithms suggested by Long and Mohr to suppress



FIGURE 4. Image 5 from Table 1. The second vanishing point is incorrectly

detected where diagonals from a traffic light beam intersect the horizon line.

this source of imprecision.

# Conclusion

## Ultimately, the methods describe here provide a strong starting point for artists to begin to train their eyes to recognize and deconstruct perspectival space. Additionally, further work could be done to incorporate more robust recognition of 3-point perspective for artists as well as perspective recognition for the sake of Computer Vision navigation of real-world spaces.

References

1. S. T. Barnard, “Interpreting perspective images,” *Artificial Intelligence*, vol. 21, no. 4, pp. 435–462, Nov. 1983, doi: [10.1016/S0004-3702(83)80021-6](https://doi.org/10.1016/S0004-3702(83)80021-6).
2. J. Tighe and S. Lazebnik,*Barcelona dataset*, Chapel Hill, NC: UNC Chapel Hill, 2014. [Dataset]. Available: <http://www.cs.unc.edu/~jtighe/Papers/ECCV10/>. [Accessed: May 14, 2020].
3. R. O. Duda and P. E. Hart, “Use of the Hough transformation to detect lines and curves in pictures,” *Commun. ACM*, vol. 15, no. 1, pp. 11–15, Jan. 1972, doi: [10.1145/361237.361242](https://doi.org/10.1145/361237.361242).
4. L. Quan and R. Mohr, “Determining perspective structures using hierarchical Hough transform,” *Pattern Recognition Letters*, vol. 9, no. 4, pp. 279–286, May 1989, doi: [10.1016/0167-8655(89)90006-8](https://doi.org/10.1016/0167-8655(89)90006-8).

1. M. Kalra is currently an undergraduate Electrical Engineering student at Columbia University, New York, NY 10027 USA (e-mail: mk4178@columbia.edu). [↑](#footnote-ref-1)