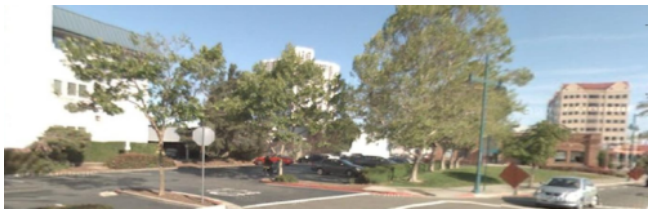
Improving Mobile GeoMaps Applications with Expressive Rendering: A Test Case

Milagro I. Feijoo* John F. Hughes†

Department of Computer Science
Brown University



(a) Example input image

(b) Reference image from Michael Gage's travel journals

Figure 1: *We want to take the Google street view image (a) as input and produce an illustration that resembles (b)*

Abstract

We present a collection of techniques that take a Google Street View image and generates an expressive rendering of vegetation based on the style of Michael Gage's travel journals. This serves as a prototype for the general problem of rendering an expressive representation of an urban enviroment from a photorealistic image. With this prototype we can evaluate the feasibility of generating these images adequately and the usefulness of this approach.

Keywords: expressive rendering, non-photorealistic rendering, image simplification

1 Introduction

It is said that a picture is worth a thousand words, referring to the idea that complex stories can be described using a single still image. This idea is the basis for this project, in which we use illustrations in place of photos in mobile geographic mapping applications. Our ultimate goal is to produce comprehensible imagery of urban environments in place of Google street view images –comprehensible in the sense that it is easy for the viewer to see and understand the essential elements of a scene without having to process less important information. In this particular project, we work on some small parts of this larger goal.

Figure 1(a) shows a photograph of an urban environment. Figure 1(b) shows a watercolor illustration of a similar environment drawn by Michael Gage. It is evident that it is much easier to grasp the essential information of the environment from looking at the watercolor illustration as opposed to the photograph. The problem with photorealism is that it gives equal importance to everything, including information that might be unnecessary in helping someone navigate through an urban environment. Expressive rendering (also called non-photorealistic rendering or NPR) has the advantage of bringing clarity to these types of scenes. In the context of mobile geographic mapping applications, it provides several advantages when compared to photographs:

- Images with less information are viewed more clearly on small displays, such as those on PDAs and smartphones.
- The files of expressive renderings are significantly smaller than the original photographs. This translates into faster download times since it requires less bandwidth.
- Labels and other visual aids can be applied to the image without making it too visually cluttered.

Features

We analyzed both illustrations from Michael Gage's travel journals and many streetview images to identify features which we considered important for providing the viewer with the necessary information to navigate his surroundings. We identified the following features:

- Buildings / Other structures
- Vegetation: Trees, Bushes, Flowers
- Color
- Building number / Traffic signs / Street names
- Walls / Fences
- Doors / Windows / Awnings
- Pools / Parks / Fields
- Decorative features, such as sculptures and flower pots
- Stairs / Ramps
- Bus stops
- Traffic lights / Lamp posts / Power lines
- Mailboxes / Newspaper boxes
- Garbage bins
- Fire hydrants

Goal

The focus of this particular project is to create a prototype that will aid us in evaluating the proposed approach for the larger goal. A fast way to create a proof of concept is to focus on just one of the features listed above. Buildings and other architectural structures, such as bridges, seem like the natural choice since they are essential for urban scene understanding. We believe that combining Google's existing data on building geometry, street view and satellite images, it should be relatively straightforward to identify and render these

*mfeijoo@cs.brown.edu
†jfh@cs.brown.edu

Figure 2: *Sample of the input for the algorithm. Most of the images are screen captures of Google Street View except the last two which are raw images from a modest camera on a day with very good lighting. Notice how the four images on the top are less saturated than the images on the bottom.*

buildings, by drawing the outline and doing a flood fill with their main color. Another feature that dominates most scenes is vegetation. The type, size, height, color or placement of a tree or bush will surely help a person get their bearings. However, when depicted by a photograph, vegetation contains a lot of details, particularly in terms of texture and shape. This provides a good place to examine what the abstraction process will produce.

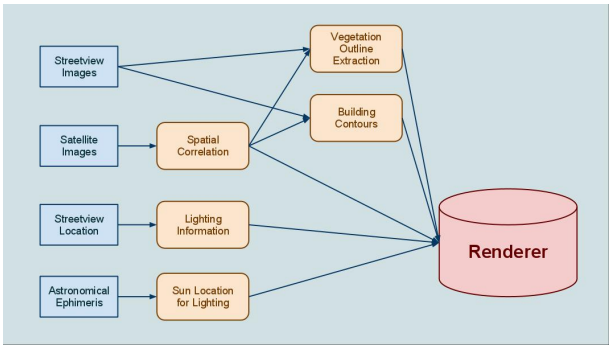


Figure 3: Architecture Diagram.

2 Related Work

The goal of a lot of the work that has been done in expressive rendering has been to take a photorealistic image or a scene description, in either 2D or 3D, and make it look like an image from a particular artist or style.

In general, *"there is ample evidence that non-photorealistic renditions are in fact more effective for communicating specific information than photographs or photorealistic renditions in many situations"* as stated in [Strothotte and Schlechtweg 2002]. In fact *'Over and above this empirical evidence, many studies have been carried out by cognitive and educational psychologists that attest to the superiority of such handmade graphics over photolike images."*

We have a slightly different goal in this project in the sense that we are trying to create a non-photorealistic representation that corresponds to an abstaction of an urban scene. We need to pick out the elements of the image that say a lot about the scene in the simplest way possible, trying to bring out the large scale details while omitting the smaller scale ones which, despite providing a realistic representation of a scene, clutter the image with non-essential information.

Some work has been done in stylization and abstraction of photographs by [DeCarlo and Santella 2002]. They describe a computational approach for clarifying the meaningful structure of an image. They use data collected from a visual perception study to obtain information about the important areas of the picture and use this information to create an abstraction that considers the visual structure of the image.

3 The Vegetation Problem

As described above, we want to create a prototype for expressive rendering of urban scenes. Vegetation is a dominant feature that provides enough varying characteristics to aid navigation in an unfamiliar place. The solution we propose is a two step process. First, we need to process the source image to identify vegetation and obtain the data describing that vegetation. Then, we need to come up with a representation that captures the simplicity, clarity and beauty of our reference illustrations.

The problem of identifying vegetation is a difficult one. Even if we can correctly mark the green pixels as candidates for vegetation, we need to get rid of green pixels that belong to things like buildings and cars. We could do that by looking at the texture of the green area, as cars and buildings generally have smoother and less varying texture. After getting rid of the green pixels which are not vegetation, we still have the issue of determining what type of tree should be drawn: deciduous, evergreen or palm trees, to name a few. This problem remains unsolved, but we have achieved reasonable results using photographs with good lighting and without green cars or buildings. Also, we do not attempt to determine the type of tree; the user selects what type of tree they want drawn in that particular scene. That is, in our prototype, the tree-classification is a transformation step implemented by a human rather than an algorithm.

3.1 Input and Output Description

The input for our algorithm consists of an image taken from Google Street View, examples of which you can see in Figure 2. For the purposes of this project we took screen captures of Google's Street View and cropped out the extra information provided by the application. It seems as though most of these images were taken in bad lighting situations as their saturation values are very low. We can speculate about the reasons why this is so. The images might have been blurred to combine it with others to simulate the environment or down-sampled to display well on browsers. As for the bad lighting, these photographs are probably taken at times when traffic is low, like dawn or dusk, which do not provide good lighting. Perhaps if this application were to be developed within Google it could use the raw images, which would likely provide better results.

Our output image is a modification of the source image where the trees have been replaced by the tree abstraction.

3.2 Tree Identification

In our efforts to generate a tree identification algorithm we attempted several different approaches. None were completely successful, but some yielded interesting results that if investigated further could provide a solution to the given problem. We now proceed to describe these successes and failures, since they may be useful to others who attempt to solve this problem.

3.2.1 Image Segmentation with k-means

For this algorithm we work with the image in L*a*b color space. This color model is designed to approximate human vision, so it can help us determine if two colors are perceptually close to each other. We remove the luminance channel L, since we only need to compare the chromaticity of a pixel to determine whether it is green. We then take these two channels and run a k-means clustering algorithm on them with $k \in \{3, 4, 5\}$. To guarantee that at least one of the clusters is green we initialize the cluster centers at green. For each of the clusters returned we take the mean of the RGB value of the pixels in that cluster and replace the original RGB value of these pixels with this mean RGB. Then we take a predefined set of greens and compare them with the cluster colors. We mark the pixels in the cluster with the smallest distance to this set of greens as tree. This provides the input for the tree representation algorithm that will be described in section [3.3].

3.2.2 Tree Marking Tool

While trying to obtain the tree pixels automatically, to aid our testing in other areas of the project, we implemented a simple selection tool to mark a polygon that defines the outline of the tree crown. This tool allows the user to select as many trees as they want. Upon closing the polygon the tool generates a splined curve from the polygon points to smooth the outline for the tree crown. The tool outputs the outline points, the mask, and the pixels that are contained in the outline.

3.2.3 Naive Vegetation Detection

Before segmenting the image with k-means as described in [3.2.1] we took a naive approach at vegetation detection in which we compared the color value of each pixel to a set of greens to determine whether to mark it as vegetation.

We work with the image in L*u*v color space because it is perceptually uniform, so distances between colors are measured by their euclidean distance. As when working with L*a*b space, we also remove the luminance channel and work with the two channels that represent the pixel's chromaticity. The algorithm consists of going through each pixel in the image and calculating the euclidean distance between its uv values and the uv values of a set of greens, and, if the distance is smaller than a certain threshold, the pixel is marked as green. We expected to obtain a very scattered collection of pixels, given that photographs have a lot of inter-pixel color variation. However, if we obtained good enough results, we could apply a filter to the detected pixels such that we could mark whole sections of pixels as vegetation. These extensions are explained in the next three sections. With this in mind, the results obtained were not very good and we believe this has to do with the fact that the threshold is an arbitrary number. Given that there is a significant amount of inter-scene variation, specifically in terms of lighting, a single value for this threshold does not successfully detect greens in every image. This means that some of the detected pixels in some of the scenes appear to be more brown or gray than green and some pixels that appear to be green were not detected as so.

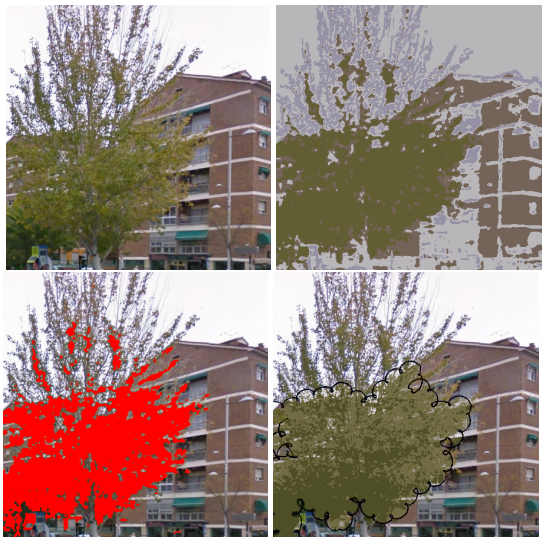


Figure 4: Segmentation with k-means. *The top left image shows the input. The top right image shows the segmented image with k=4. The bottom left image shows the detected vegetation pixels. The bottom right image shows the output of our algorithm.*

Figure 5: *Naive Tree Detection*

3.2.4 Dilation/Erosion Filter

To improve these results we include an additional step in which we dilate and erode the image once. The parameters of the dilation filter used are a line structure element (strel) with length 4 and 0 degrees and another line structure element with length 4 and 90

degrees. The parameter of the erosion filter used is a disk structure element with a radius of 4. In preprocessing we apply this filter to the original source image before running the vegetation detection algorithm.

Figure 6: *Dilation/Erosion Filter in Preprocessing*

In postprocessing we apply this filter to the pixels that were detected by running the vegetation detection algorithm on the original source image.

Figure 7: *Dilation/Erosion Filter in Postprocessing*

We expected, specifically by doing this in postprocessing, that this filter would get rid of very small areas of detected groups of pixels and reduce the holes in the bigger areas. We applied it also in preprocessing to see what would happen in that case. The actual result was worse than expected, since the naive vegetation detection does not actually output big solid areas of detected pixels, but instead areas that appear very porous from the amount of holes in them.

3.2.5 Median Filter

Just as with the dilation and erosion filter, we apply a median filter to the image in preprocessing and to the tree mask in postprocessing. In preprocessing we use a neighborhood of 10x10 and apply the filter twice. In postprocessing we use a neighborhood of 2x2 and apply the filter 10 times. These parameters seemed to provide the best results for vegetation detection. Same as with the dilation and erosion filter we expected this to reduce the amount of holes in the green areas and get rid of small green areas surrounded by other colors, in this case, we thought that by applying the median filter as a preprocessing step we could blur these small green areas and holes out of the original image. We also tried to apply it in postprocessing to see what would happen. This filter does not produce good results either, but in this case the algorithm is overly aggressive and detects areas that are not green.

3.2.6 Multi-scale Naive Green Detection

Another approach at improving these results we took was running the original green detection on an image pyramid. We tried this as an extension of the median filter, so that we could examine the

Figure 8: *Median Filter in Preprocessing*

Figure 9: *Median Filter in Postprocessing*

effect of this filter at various scales. This did not improve results very much. As you can see in Figure 10 this modification makes the algorithm overly aggressive and large sections of the image are marked as green when they are not.

3.2.7 Color Image Segmentation (Matlab File Exchange)

There was also an off-the-shelf solution to a related problem that seemed to have the potential to work well for vegetation detection. Unfortunately, the clustering is overly aggressive, and detects more clusters than necessary and does not separate greens appropriately. We found it on the Matlab File Exchange website http://www.mathworks.com/matlabcentral/fileexchange/25257-color-image-segmentation

3.2.8 Summary

We found that the best option for vegetation detection is Image Segmentation with k-means, given that the detected regions were the most consistent across the spectrum of images we tested.

3.3 Tree representation

From examining several reference illustrations of trees we identified a few distinct qualities that we considered were important in conveying the essential information necessary for a human to understand that they are looking at a tree. On one hand we need to provide an outline for the tree. Typically humans draw a loopy outline to describe a tree in line-drawing style. In Figure 11 we can see some examples of line drawings of trees as humans typically draw them.

On the other hand we would like to create an abstraction for the color of the trees. The original photograph contains a lot of texture that visually clutters the scene with information that is unnecessary to convey the essential scene elements. So simplifying the pixels contained in the tree outline to only a few greens seems like the natural solution. Additionally, to give it a more artistic look, we imitate our reference illustrations for providing shading and depth

Figure 10: Multi-scale Naive Green Detection. *At left the source image. At right, in order of iteration number, are the multi-scale results for a section of one of the images.*

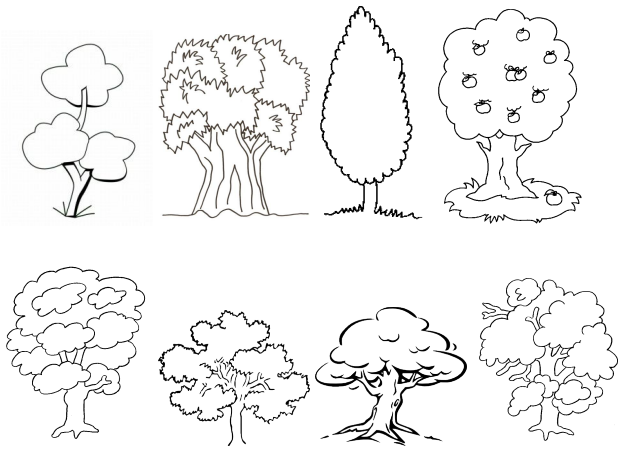


Figure 11: *Examples of line drawings of trees*

to the scene, by using a watercolor style. Deciduous trees and evergreens can be depicted very similarly by defining a bushy outline and filling it in with color. As for the palm trees we thought it would be best to use the position and size of the palm tree and draw a general palm tree shape.

3.3.1 Drawing a bushy tree outline

Given the set of pixels that got marked as tree pixels we obtain the boundaries of each disconnected section, represented as a list of points. We then get rid of boundaries with less than 30 points, since we consider these to be too small to draw. These outlines are defined by a polygon. After we obtain each of the polygons we generate a spline curve with the points in the polygons as the spline control points. We then generate the loopy outline from this spline with a function based on a mfeijoo: cycloid function with a radius of 20 pixels. There are a few tuning parameters to generate different variations of the loopy outline. We can alter the height, the width and overlap of each loop. There are a few randomization steps throughout the generation of this curve, so as to introduce variations in the roundness of the loop, or small variations in height, width and overlap based on the set values for these parameters. This is intended to reduce the regularity of a computer generated curve. We also introduce a smooth variation to the thickness of the line, so that it simulates the difference in the pressure that a human can exert on the paper while drawing a curve.

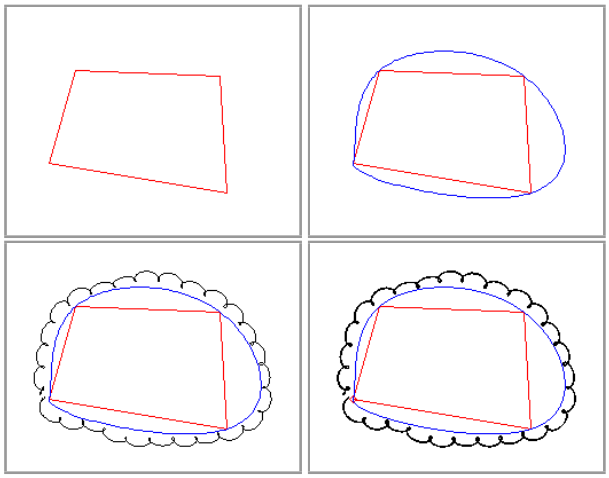


Figure 12: Drawing a bushy tree outline. *The image on the top left shows, in red, the polygon drawn by a user. The image on the top right shows, in blue, the spline generated by the points of the polygon. The image on the bottom left shows, in black, the loopy outline generated. The image on the bottom right incorporates the thickness variation to simulate a pen stroke.*

3.3.2 Coloring in the bushy tree crown

As mentioned before, we need to incorporate abstraction to the texture of the color fill of the tree crown. To do this we run k-means to obtain the 3 most popular greens. We follow a similar approach as when color segmenting the image where we set each pixel to its cluster color. This creates a particular shading to the trees that is very much influenced by the original image structure. So the shading is very close to what is desired. However, the edges between

cluster sections are very sharp and they end up looking very pixelated, so it still feels computer generated. You can see results of this algorithm in Figure 12.

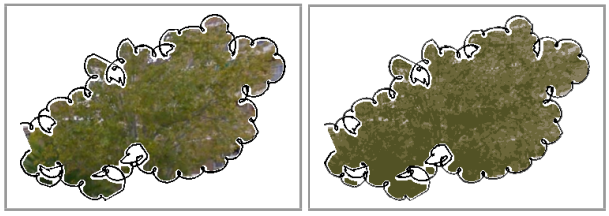


Figure 13: Coloring in the bushy tree crown. *On the left are the original pixel values of the marked tree crown. On the right the colors have been reduced to only three greens that are calculated by clustering the existing colors and assigning the mean color to each cluster.*

Another member of the research group applied watercolor to this output and produced the results in Figure 13.

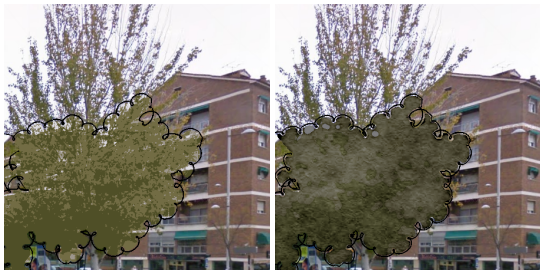


Figure 14: *Watercolor modification of color fill. Project by Patrick Doran.*

3.3.3 Drawing Palm Fronds

The input of this algorithm is the region where the palm is to be drawn defined by a center point and its size defined by a width and a height. We then define a radius as the maximum of height and width and that will determine the length of each frond on the palm tree. A real palm tree has about 15 or 20 fronds, however an artists' depiction of a palm tree usually shows between 5 and 9 fronds, so we pick the number we draw at random from this range. We set the position guide of each frond at $i*(2\pi/lq)$ where i indicates which frond and lq is the frond quantity. We then compute the position guide points for each of the fronds. We take these points and add a small random rotation to break up some of the uniformity that the guide lines originally have. We then proceed to alter their shape by sampling three points: the first and last points of the guide line and one in the middle. We compute a curvature based on the length of the leaf and use that to alter the middle point by that amount in the direction of the normal of the frond. We then proceed to create a spline based on these 3 control points, which will constitute the frond midrib. We replicate this procedure to determine two other curves that will serve as boundaries for the leaf length in each frond. To draw each leaf on the frond we sample points on the midrib line such that the distance between them is given by $ceil(max(w,h)/5)$, where w and h correspond to the width and height of the desired palm tree. This means that the sampling rate for the leaves will vary according to the size of the palm tree. We then draw a line from each of these points to the boundary line at a 45° from the midrib line. To make it seem a bit more palm-like, we draw a black 1 pixel width line through our midrib and leaf points and also a green line, whose

Figure 15: Drawing Palm Fronds. *An example of drawing palm fronds.*

width is given by $2*ceil(max(w,h)/100)$, as before w and h correspond to the width and height of the desired palm tree. Sampling the root of leaves and the width of the green line this way aids in maintaining a sense of scale for the palms.

4 Results and Future Work

Our current algorithm runs in an average of 20 seconds and the bulk of processing time occurs in the vegetation detection algorithm. Figure 16 shows several results.

There are many ways in which this algorithm could be further developed to generate better results, both in the identification step and in the representation step.

For instance, the vegetation detection could be improved by using object detection algorithms or image segmentation algorithms, such as the porous classifier of Hoiem et al. [Hoiem et al. 2005] or even their Scene Interpretation framework [Hoiem et al. 2008]

The tree outline representation has room for improvement. The boundary of areas marked as tree in the vegetation identification step is jagged at a very fine scale. This causes mfeijoo: some - does it say red? artifacts when drawing the loopy outline of the tree. If we apply a smoothing algorithm to these areas, we could generate a smoother and better looking tree outline. An abstraction algorithm such as the one proposed in [Mi et al. 2009], might be useful in obtaining an outline that maintains the global shape of the tree crown and smooths out the local smaller features of the shape boundary. The current continuous manner in which strokes are drawn makes these drawings appear computer generated. Even if that is the case, we would like to make them appear more like a human drew them. It is possible to do this by cutting out the outline at random places by a small amount, in this way simulating the fact that people lift the pen when drawing.

Similarly, the tree outline stroke can be improved. Our suggestion to those trying to further this research is to look at earlier research on stroke synthesis, such as that of Northrup and Markosian in [Northrup and Markosian 2000].

A natural progression towards creating an application that generates expressive renderings of urban scenes is to find solutions to the different problems that arise when attempting to identify and represent each of the other features that compose these scenes. Each of

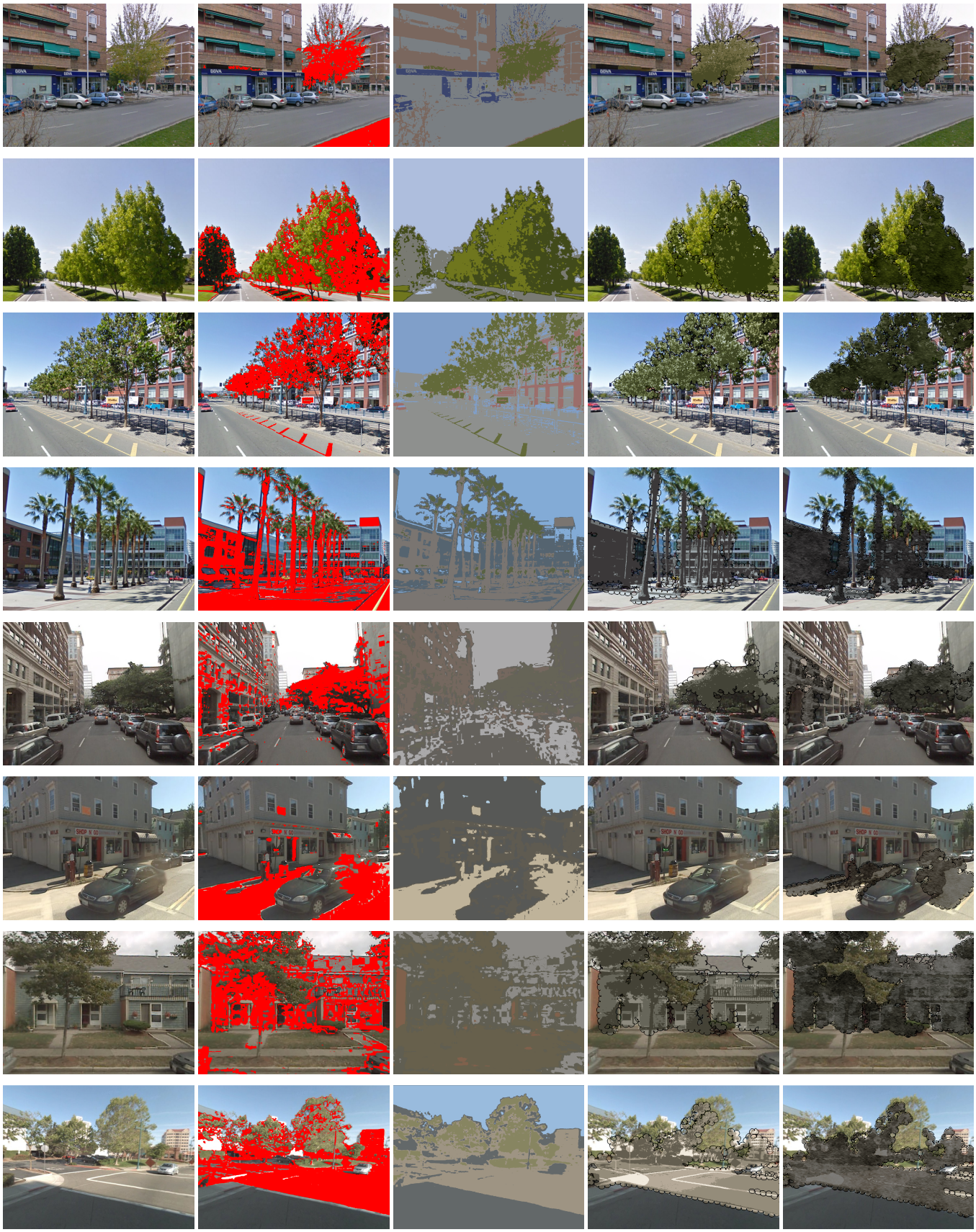


Figure 16: *The first three rows show somewhat good results, the fourth row is not very good, and the last four rows are very bad.*

these features in turn has the potential to be a very active research topic in other areas, such as text recognition in Computer Vision.

We suggest to the reader to read [DeCarlo and Santella 2002] as a source of alternative ideas as to how to reduce visual clutter from the scene by removing texture information.

5 Conclusion

In this paper, we present a collection of techniques that were put together to render an expressive rendering of vegetation from a Google Street View photograph. The main goal was to create a prototype for the general problem of rendering an expressive representation of an urban environment, to examine the feasibility and usefulness of generating such imagery, in particular for mobile applications. For this goal, we chose a specific representation with an abstraction level that includes the essential information for environment navigation and a very picturesque feel.

Our process is modular in that we can combine several simple techniques to produce these expressive renderings. Each of these techniques could potentially be replaced by a more complex technique such that better results are produced for that particular subproblem, which in turn means a better overall solution to the vegetation identification and representation problem.

References

DECARLO, D., AND SANTELLA, A. 2002. Stylization and abstraction of photographs. *ACM Trans. Graph. 21*, 3, 769–776.

HOIEM, D., EFROS, A. A., AND HEBERT, M. 2005. Geometric context from a single image. In *ICCV '05: Proceedings of the Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, IEEE Computer Society, Washington, DC, USA, 654–661.

HOIEM, D., EFROS, A. A., AND HEBERT, M. 2008. Closing the loop on scene interpretation. In *Proc. Computer Vision and Pattern Recognition (CVPR)*.

MI, X., DECARLO, D., AND STONE, M. 2009. Abstraction of 2d shapes in terms of parts. In *NPAR '09: Proceedings of the 7th International Symposium on Non-Photorealistic Animation and Rendering*, ACM, New York, NY, USA, 15–24.

NORTHRUP, J. D., AND MARKOSIAN, L. 2000. Artistic silhouettes: a hybrid approach. In *NPAR '00: Proceedings of the 1st international symposium on Non-photorealistic animation and rendering*, ACM, New York, NY, USA, 31–37.

STROTHOTTE, T., AND SCHLECHTWEG, S. 2002. *Non-photorealistic computer graphics: modeling, rendering, and animation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.