Learning and Producing 2-D Artistic Shadings of 3-D Models

John Turgeson
Georgia Tech
john.turgeson@gatech.edu

Sehoon Ha
Georgia Tech
sehoon.ha@gmail.com

Karthik Raveendran
Georgia Tech
kraveendran@gatech.edu

Abstract

Cartoon shading can be a long and tedious task when creating an animated work of even minute length. In this paper, we attempt to automate the process by having an artist shade a 2-D image of a small portion or limited view of an object. We then attempt to learn the stylistic elements, or characteristics of the artist, using supervised learning algorithms and then apply that knowledge to shade in 2-D the rest of the object, the object from another view, and / or other objects altogether. We present renderings of a variety of 3D models, produced using learned models of artist style. We show that neural networks are a effective means of learning such styles and provide a comparative analysis against another supervised learning algorithm, k-Nearest Neighbors.

1 Introduction

Over the last two decades, advances in rendering have enabled the creation of photorealistic images and animations that are used extensively in film and television. At the other end of the spectrum, non-photorealistic renderings (NPR) are desirable for cartoons and certain stylized films. There has been extensive research on generating non-photorealistic renderings (NPR) of 3D models ([9, 5]). The majority of the work in this area focuses on creating algorithms that render certain features of the shape (such as silhouette edges and contours) with a defined style [[6, 4]]. For instance, the work by Eden et. al ([3]) automatically stylizes water by using bold outlines to stress depth discontinuities and patches of constant color to highlight near-silhouettes and areas of thinness. However, such models need to be re-programmed (by modifying the shaders) if a different look is desired. Other researchers have studied the extraction of brush stroke information from artist sketches using image processing as well as machine learning techniques such as Hidden Markov Models ([8, 11, 10, 7]). These approaches allow artists to apply an extracted style to new models or images. Cole and colleagues ([1]) present an analysis of how lines drawn by artists correlate with lines drawn by computer algorithms for a given model.

We believe that application of machine learning can significantly reduce the effort of creating custom NPR shaders and algorithms by learning a shading model automatically from artist drawn examples. Further, subtleties of the artist style may be captured and used to synthesize a more natural looking image. In this paper, we present a method to learn artist style from sketches of a known 3D model. To our knowledge, there has been no research that directly attempts to correlate artist style with features on a 3D computer model of the scene.

2 Artist Shading: 2-D Image Features

We first describe the features we extract from the 2-D sketches of the artist, our rationale for their inclusion, and the process by which we extract them. These become the classification vectors for the machine learning algorithm.

2.1 Feature Description

- **Intensity** - Intensity is used to emphasize regions of an image and is one of the key components of any shading style.
- **Brush Type** - Artists use solid lines as well as crosshatching and stippling which are common techniques used to fool the eye into thinking that pure black marks on a white surface are really shades of gray. There are many ways to crosshatch but the most general are included in the sketching program (Line, Backwards diagonal, Forwards diagonal, Cross diagonal, Cross, Horizontal stripes, Vertical stripes, Solid).
- **Brush Width** - The size of the brush is important because it encapsulates size and / or importance.
- **Brush Direction** - The brush direction is used when inferring the crosshatching type when using the line brush. It is included in the feature vector because it is relevant when using non-uniform brushes and is additional data that aids in clustering pixels in a patch. We plan on implementing the actual velocities in future work.
- **Stroke Length (not currently implemented)** - We would like to capture the stroke length or how long the tool has been in contact with the surface. This would help differentiate short quick strokes with long smooth ones.
- **Angle of Brush (not currently implemented)** - The stylus allows us measure the angle it is being used at relative to the surface. We believe this could add additional data to enhance shading and help distinguish different artists.

2.2 Feature Extraction

There are three steps in our classification vector extraction pipeline.

2.2.1 Scene Generation

The first step in the feature extraction process is to create the scene that the artist will shade. One or several objects are loaded into a scene and rendered using a perspective projection. The artist is then allowed to manipulate the camera (translate and rotate) to generate the desired view. The scene is then rendered using only flat shading and this 2-dimensional representation (an image) is sent to the next stage.

2.2.2 Sketch acquisition

Shading a line drawing is not a trivial task and artists usually employ a variety of brushes with varying parameters such as size and color in order to produce a particular look. We use the wxWidget library to load the image generated earlier and add our shading and other stylistic elements. Behind the scenes, we are capture the user input (type of brush, position of stroke etc.) that will be used to generate our classification features. Currently, the artist can sketch an image using several brushes with varying thicknesses and grayscale intensity. To guide the sketch, the reference image (3D rendered scene) can be loaded and be overlayed on the screen at 50% transparency. A grid can be also toggled to aid the artist.

2.2.3 Feature Extraction

We convert all brush strokes of the artist into classification vectors by sampling them on a grid. Whenever the stylus is used the current patch it is editing is found. The current brush and intensity values are recorded into that patch. If a patch is edited again, the brush type and size currently in use it will overwrite the old ones stored for that patch. The direction vector is also overwritten. Intensity

values are alpha blended between the current patch value and the brush color. The patches are saved to a comma delimited file which is used in the next step.

3 Object Shading: 3-D Model Features

In order to use machine learning to shade 3-D models, we need to first represent them using a finite set of features. In this section, we describe the features that we extract, our reasoning behind these choices and the algorithmic details of extracting these features. We combine these to form a mesh feature vector which can be used in our machine learning algorithm.

3.1 Feature Description

- **Diffuse lighting component** - If we assume that a simple lighting model where all light is diffuse (and with no global illumination or inter-reflection of light), the amount of light received by a point on the surface of an object can be computing easily by taking the dot product between the vector from point of interest to the light source and the surface normal at that point. Light is the single most important element when shading and this feature captures how the light is bouncing off the surface at our point of interest.
- **Distance from the camera** - The distance from the camera to the point on the surface is trivial to compute and this is done by using the distance formula. This accounts for the effect of perspective shortening. It captures the fact that identical objects are not perceived as being of the same size if they are not situated at the same distance from the viewer. The farther away as object is, the smaller it looks. Artists tend to use different brush sizes to create this illusion and this feature attempts to model this.
- **Curvature** - This determines the sharpness of a feature on the mesh. Bumps and furrows have high curvature, while flat regions have zero curvature. It is vital to account for this because these features add richness and visual detail to what would otherwise be a flat surface. We use the discrete mean curvature at the point of interest using Desbrun's cotangent based formula ([2]).
- **Distance to closest edge** - The distance from the point of interest to the closest edge is equivalent to the problem of finding the shortest distance between a point and all line segments around near it. This can be done by finding the triangle that the point belongs to and computing the distance from it to each side. Specifically, we project the point onto each edge and compute the distance to the projected point. This feature might be able to capture distinct shading styles used by an artist to indicate boundaries or discontinuities in the 2D image.
- **Is the closest edge a silhouette edge (to camera)?** - A silhouette edge is an edge between two triangles where one triangle is forward facing (faces the camera) and the other is backward facing (faces away from the camera). Silhouette edge detection is an active area of research. We use an approximate but fast silhouette edge test. This is performed by computing the dot product between the vector from the camera to the midpoint of the common edge, and the normal of the triangle opposite of the triangle containing the point of interest. We believe this is an important feature because an artist may shade a boundary or outline edge differently from internal edges on an object.

The curvature and silhouette edge algorithms require connectivity information from the mesh. To do so, we build a corner table that enables us to move to adjacent triangles. We also precompute curvature values at the vertices of a mesh and interpolate them linearly across the face of triangles.

3.2 Pixel to Mesh Mapping

From the edited portions in our sketched image, we need find the actual coordinates of the points on the surface of the model (in 3-dimensional space) that correspond to the pixels in the sketch that were drawn on (2-dimensional space). We do this by first acquiring the OpenGL projection and modelview matrices and then unprojecting the point in screenspace (by inverting the product of these matrices). We super sample each patch (typically 8 by 8 pixels) by looking at 16 pixels inside it. Once we map our pixels to points on the mesh, we compute the relevant features (listed above) from

the triangulated mesh and average them out to produce a feature vector per patch. These features are saved to a comma delimited file, paired with their classification features (from the sketch), and are ready to be used to train a machine learning algorithm.

An overview of the entire process is shown in Figure 1.

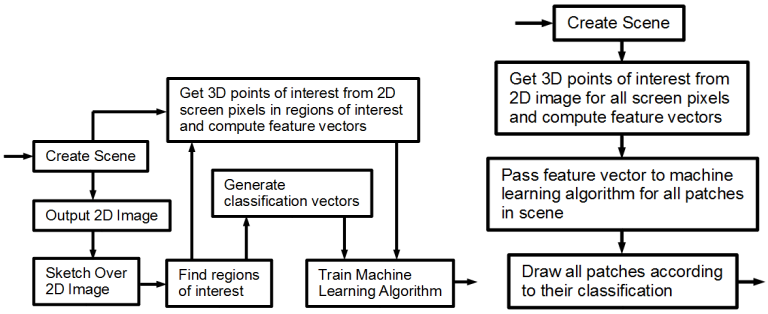

Figure 1: Overview of process

4 Machine Learning

4.1 Data Set Constraints

There are several important considerations we took into account when selecting our machine learning algorithm:

1. Both our feature vector and classification vector are a mix of continuous and discrete variables.
2. We have a classification vector, not just a single variable.
3. We will be generating the data ourselves.

Our training data set contains instances that are concatenations of the mesh feature vectors and the sketch classification vectors for each patch. Once we acquire a learned model, we feed in the mesh feature vector for each patch in the scene and obtain a classification vector (or brush) that we use to shade the patch in the final image. We applied two machine learning techniques on our data.

4.2 Neural Network

A neural network is a natural fit for out data. It handles both discrete, continuous data or a mix of the two without any modifications. We simply set up our input nodes to our input vector and the output nodes to the classification vector. We use the FANN toolkit to set up our neural networks and backpropagation to train them.

4.3 K-Nearest Neighbor

A kNN algorithm can handle a mix of discrete and continuous input variables and match to a vector. The only downside is that we need to generate a large amount of (quality) data. We use the Approximated Nearest Neighbor library to run this algorithm on our data.

4.4 Comparison on Single Classification Features

We ran the two algorithms on a test data set to determine optimal parameters for each. We plotted the accuracy of finding the correct brush type and intensity in Figures 1 and 2. The general trend is that the error rate decreases for kNN if we increase the number of neighbors. This is likely due to noisy input data. We were able to obtain a reasonable brush size accuracy of 89 percent using 9 hidden nodes for the neural network.

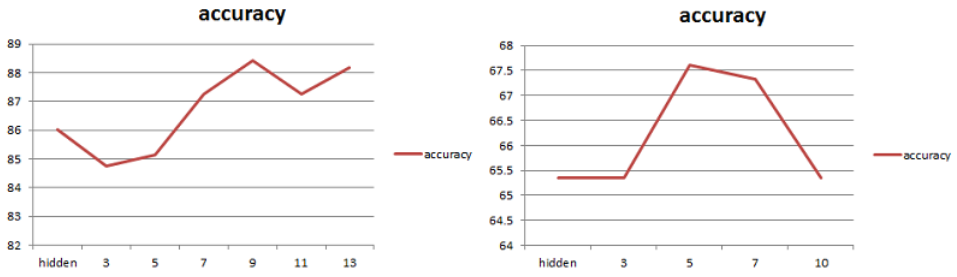

Figure 2: Plot of accuracy of neural networks against number of hidden neurons (left: based on brush size, right: based on brush type)

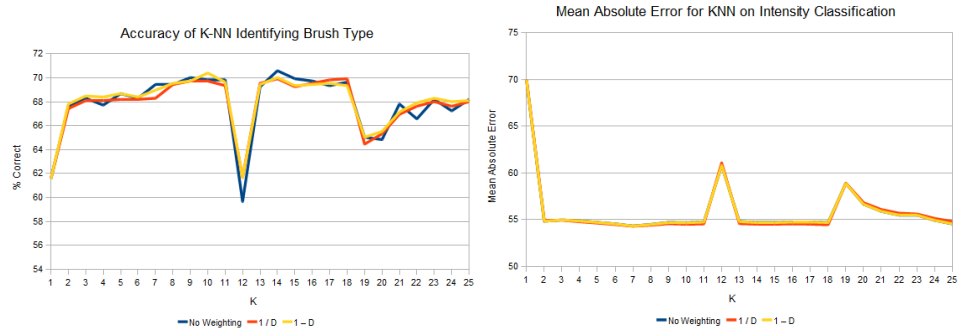

Figure 3: Plot of kNN accuracy for different values of k

5 Results

We present a number of images synthesized by our algorithms. We ran four distinct comparisons to gauge the quality of our output.

1. **Cross-comparison:** We asked an artist to create two reference sketches of two objects (one of each). We then generated an image of an object from a different view, using training data from a sketch of itself. We also generated an image from the reference sketch of the other object. This cross comparison helps us understand if our shading style transfers well to different, unseen objects. We used neural networks to learn the model for this test. Our test results are listed in Figure 4. The style learned from the bunny generalizes well to other views of itself as well as the sphere. The silhouettes are perfectly captured. Further, it appears to detect the bumps on the bunny (near the hind legs) and also shades regions of low curvature with a cross brush. However, the style learned from the sphere does not give a satisfactory result when used to shade the bunny. This is likely due to the fact that the sphere has constant mean curvature and our algorithm is unable to learn how to deal with non-uniform curvature. The body of the bunny is full of bumps and furrows and these effectively appear as unseen features that are not shaded with a very limited knowledge. In general, we believe that our machine learning framework produces a reasonable result for this test case.

2. **Artist style comparison:** We requested two artists to produce their stylized version of the Stanford bunny model. We then synthesized a number of other shapes using their styles and compared the renderings. From Figure 4, the algorithm clearly produces distinct styles based on the two input datasets. Note the use of dark silhouette edges in the images of artist 1 while those produced by artist 2 use a thinner, light brush. In particular, the horse demonstrates the success of our technique by capturing illumination and silhouettes perfectly in two different artistic styles. The hand model is a much tougher test case that

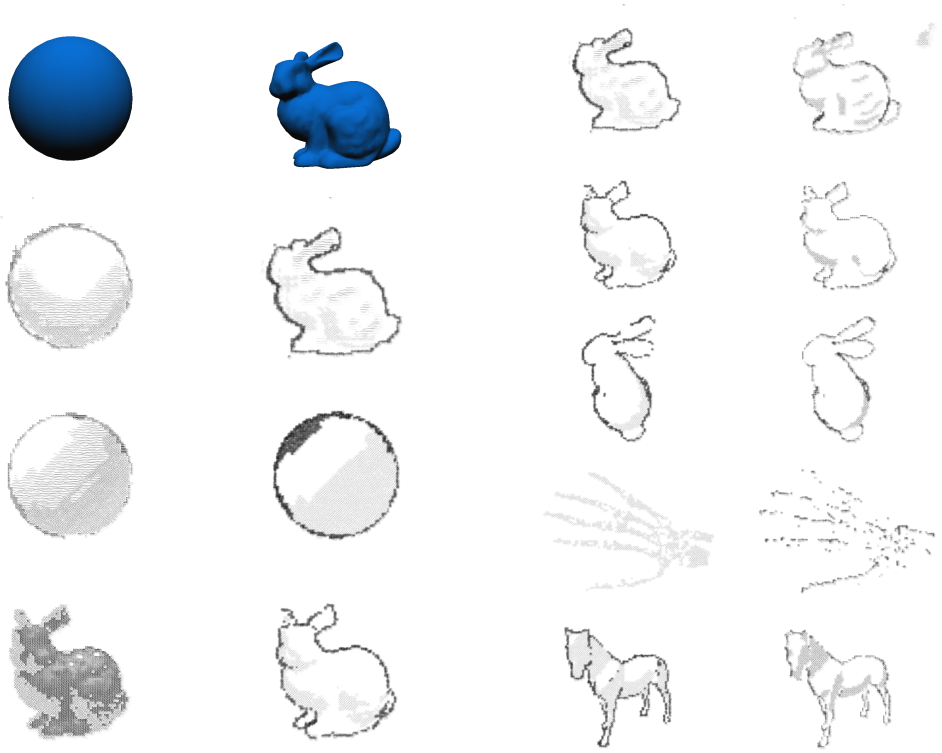

Figure 4: (Left): Comparison of sketch styles learned on two models: The first row shows the reference 3D model and camera view. The second row contains the corresponding artist sketch. The third row shows the reference shape being shaded by the machine learning algorithm. The fourth row shows the other reference shape being shaded in the same style (Right): Models rendered using two artist styles (first row shows the two reference sketches)

appears to be limited more by the resolution of our patches because it has thin features in screenspace and our average feature vector is too coarse to be of much use.

3. **Machine learning method comparison:** We used a reference sketch of the Stanford bunny and generated results for a new view of the bunny using both neural networks and kNN. In figure 5, we see significant differences in the images produced by neural networks and kNN. The latter appears to be noisy with inconsistent shading of the silhouettes (alternates between brush types). The neural network produces an cleaner sketch of the bunny by clearly outlining the object and then filling in the darker regions with backward hatching.
4. **kNN parameter comparison:** We varied the number of neighbors used by kNN and generated sketches of the Stanford bunny. Figure 6 shows that by increasing the number of neighbors, we obtain smoother and more consistent shading. The bottom-right figure (using 15 neighbors) eliminates the dark splotches seen in the top-left image (using 1 neighbor). kNN also appears to capture curvature and silhouette edges.

In summary, our preliminary results using this approach suggest that there is potential in using machine learning for automatic non-photorealistic shading. While the fidelity of the reconstructed images is still low (due to the relatively large patch size and aliasing), the images indicate that most noticeable 3D features are shaded with the expected style.

6 Conclusions and Future Work

In this paper we introduced the idea of using machine learning to map 2-D shading techniques to 3-D mesh information. Using a simple sketching program we recorded stylus input as well as the current brush settings to compute several 2-D features such as intensity and brush type which we

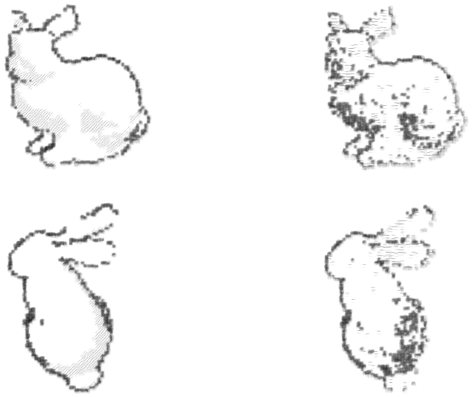

Figure 5: Comparison of sketched generated using neural networks (left) with those using kNN (right)

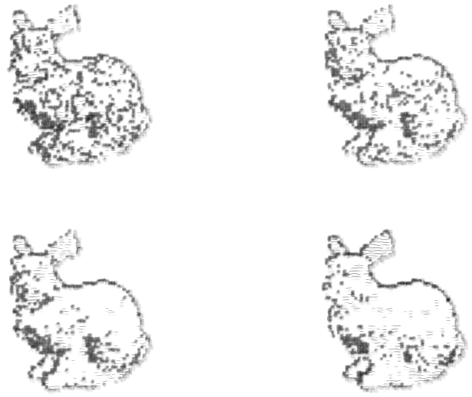

Figure 6: Comparison of sketches of a bunny using different values of k for kNN (left to right, top to bottom: 1, 3, 6, 15)

used as our classification vector. We found the 3-D points on the mesh that corresponded to our 2-D edited points and computed several features about the mesh there, including distance from camera, incident of light and curvature. These 3-D features were used to construct a feature vector. We trained two machine learning algorithms on our data set and used this model to shade additional 3-D meshes. Neural networks currently offer more promise compared to kNN on our test datasets, however larger training datasets may favor the latter due to its lazy evaluation.

Future work includes adding additional shading tools and options to the shading program including additional brush types and possibly color. We plan to include the not yet implemented 2-D features (stroke velocity, stroke length, and angle of stylus) and hope to think of several more. We are prioritizing improving the shading feature selection and generation but also plan on adding additional information or features about the mesh at the points of interest. Several possible features could be surface material, angle between light source and camera, and shadows. We are currently using a single non-attenuating light source and plan on adding additional lights and light source types. A more streamlined process, perhaps scripts that open the multiple programs would reduce the time to generate and test data. Reconstruction of the image could also be significantly improved by generating larger strokes (by connecting similar patches) instead of simply alpha blending square elements. This would likely remove the aliasing observed in our current results.

Acknowledgments

We would like to thank Yan Xu for providing us with a Wacom tablet for our sketches.

References

[1] Forrester Cole, Aleksey Golovinskiy, Alex Limpaecher, Heather Stoddart Barros, Adam Finkelstein, Thomas Funkhouser, and Szymon Rusinkiewicz. Where do people draw lines? *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 27(3), August 2008.

[2] Mathieu Desbrun, Mark Meyer, Peter Schröder, and Alan H. Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '99, pages 317–324, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[3] Ashley M. Eden, Adam W. Bargteil, Tolga G. Goktekin, Sarah Beth Eisinger, and James F. O'Brien. A method for cartoon-style rendering of liquid animations. In *Proceedings of Graphics Interface 2007 (to appear)*, pages 51–55, May 2007.

[4] OHASHI Gosuke, MOCHIZUKI Keita, NAGASHIMA Yasutake, and SHIMODAIRA Yoshifumi. Edge-based feature extraction method for image retrieval. *IEICE technical report. Image engineering*, 101(567):1–4, 2002-01-11.

[5] Aaron Hertzmann. Tutorial: A survey of stroke-based rendering. *IEEE Comput. Graph. Appl.*, 23:70–81, July 2003.

[6] Tilke Judd, Frédo Durand, and Edward H. Adelson. Apparent ridges for line drawing. *ACM Trans. Graph.*, 26(3):19, 2007.

[7] Christopher D. Kulla, James D. Tucek, Reynold J. Bailey, and Cindy M. Grimm. Using texture synthesis for non-photorealistic shading from paint samples. In *Proceedings of the 11th Pacific Conference on Computer Graphics and Applications*, PG '03, pages 477–, Washington, DC, USA, 2003. IEEE Computer Society.

[8] Jia Li and James Z. Wang. Studying digital imagery of ancient paintings by mixtures of stochastic models. *IEEE Transactions on Image Processing*, 13:340–353, 2003.

[9] Rezwan Sayeed and Toby Howard. State of the art non-photorealistic rendering (npr) techniques. 2006.

[10] Shlomo Saul Simhon. *A machine learning framework for the classification and refinement of hand drawn curves*. PhD thesis, Montreal, Que., Canada, Canada, 2006. AAINR25255.

[11] Georges Winkenbach and David H. Salesin. Computer-generated pen-and-ink illustration. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, SIGGRAPH '94, pages 91–100, New York, NY, USA, 1994. ACM.