Style Spheres: Defining Style in NPR

Samir Roy* Greg Humphreys†

University of Virginia

Abstract

Although a large number of systems have been developed for Non-Photorealistic Rendering, many of them remain specific to a set of styles, without the ability to easily create new styles. We present a framework for artistic rendering, called "*Style Spheres*," that can learn new styles by example. A source image is first presented to the system, consisting of a sphere drawn or rendered in the desired style. The style of rendering in the sphere is learnt by the system and applied to complex 3D models and scenes.

The framework is based on a targeted, deterministic search process. Pixels in the output image are mapped to those in the input sphere by searching for the closest match. The images resulting from this synthesis have styles, perceived similar to those of the input spheres. Thus, *Style Spheres* provides an easy and intuitive method for design and specification of rendering styles.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation—Display Algorithms; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture; I.2.6 [Artificial Intelligence]: Learning—Analogies; J.5 [Computer Applications]: Arts and Humanities—Fine arts

Keywords: non-photorealistic rendering, image based rendering, texture synthesis, texture transfer, machine learning

1 Introduction

When providing tools to artists, it is important to allow them to define their own styles. In particular, artists should be limited only by their own creativity, not the tools they use. Many previous NPR techniques have perfected a specific set of styles, rather than allowing users to create new ones. In this paper, we present a system that allows artists to specify and create new, unique styles that can be used to render 3D models.

Our system, called *Style Spheres*, allows the user to specify a particular rendering style by providing a drawing of a sphere in the desired style. This sort of by-example interface keeps the style creation process intuitive and user-friendly by hiding the technical details of the underlying rendering system. In addition, a pictorial example is particularly effective, since it is often impossible to capture the subtleties and uniqueness of an artist's desired drawing style with equations or words.

Figure 1: A sample rendering from our system. The artistic style applied to the model is transferred automatically from the hand-painted sphere shown in the upper-right.

In order to apply the given style to a rendering of arbitrary 3D geometry, we use a targeted, deterministic search process inspired by previous work in texture synthesis. Each pixel in the output image is mapped to a pixel on the sphere, ensuring that the neighborhoods of the pixels are similar. Our search process is accelerated by targeting it to the area on the sphere that has the same normal vector as that of the pixel being rendered. We also transfer the drawing style of the object's silhouettes, which can be crucial to achieve the illusion of a hand-drawn model.

The remainder of the paper is organized as follows. Section 3 describes the algorithm used for rendering in detail. We also discuss different parameters that affect the synthesis process. We conclude by discussing the advantages and limitations of our approach in Section 4 and propose some directions for future work in Section 5.

2 Previous Work

An enormous amount of related research already exists in various different areas of nonphotorealistic rendering, including painterly rendering [Meier 1996]; sketching [Curtis 1998]; hatching[Praun

*sr4z@cs.virginia.edu

†humper@cs.virginia.edu

et al. 2001]; pen and ink illustration [Winkenbach and Salesin 1994]; technical illustration [Gooch et al. 1998]; and interactive techniques [Daniels 1999; Kalnins et al. 2002]. For a more exhaustive overview of NPR techniques, refer to [Gooch and Gooch 2001] or to [Strothotte and Schlechtweg 2002].

Much of the previous work in NPR has focused on providing a particular look, rather than allowing users to create new styles of rendering. Even though interactive techniques help the user to adjust parameters, the final renderings are still typically restricted to some specific rendering styles. Halper et al. propose a rendering pipeline that allows designers to iteratively modify the rendering style using elementary operations [Halper et al. 2002]. Their work focuses on the user interface and how to make it intuitive and user-friendly for designers. There have been a few notable exceptions that allow the user to define the style by example, Hertzmann et al. described a system where curves could be stylized using examples [Hertzmann et al. 2002]. Jodoin et al. developed a system to do hatching by example [Jodoin et al. 2002]. Note that in this case, although the system does use examples to define the style, those styles are still limited to hatching and could not capture the effects of a watercolor painting, for example.

Sloan et al. captured style from works of art using *Lit Spheres* [Sloan et al. 2001]. Our technique is similar in spirit; We use spheres to define the rendering style for 3D objects. Our system differes from Lit Spheres in two main ways. First, we are able to correctly capture the look of the object's silhouettes. Second, and perhaps more importantly, Sloan et al. concentrated on providing methods to create the Lit Spheres, and then simply used them as environment maps applied directly to geometry. When using environment maps, the detail in the sphere maps can be lost due to blurring and stretching. Our method preserves the fine details of the texture by using texture synthesis.

The algorithm underlying Style Spheres is primarily based upon research in texture synthesis by Wei and Levoy [2000]. Their algorithm takes an example texture patch and produces new images of different sizes that look like the given sample. Ashikhmin built upon this work and increased the quality and efficiency of the algorithm [Ashikhmin 2001]. Hertzmann et al. used these concepts to apply filters to 2D images [Hertzmann et al. 2001]. Their framework takes two images, one of which is assumed to be the filtered version of the other, and applies a similar filter to a third image.

Style Spheres extends this idea to three dimensions. Using a similar filter, a 3D model can be rendered to look like a given texture. In place of using regular textures, we use as input an image of a sphere as it would look when rendered in the desired style. In Hertzmann's method, if the first image is the normal map of the sphere, and the third image is the normal map of the 3D model, the result would be a 3D model rendered in a similar style as the input sphere. Since the normal maps are known to us, we only need the textured sphere as input, along with the 3D geometry. Further, the pixels in the source and target images should have similar normals, so we can reduce the search space from the entire image to just a few pixels around the normals. By making this assumption about the similarity of the surface normals, we were able to reduce the rendering time from hours to seconds, while actually improving the final image quality.

3 Algorithm

In this section, we describe in detail the data structures and algorithms for Style Spheres. We will use a similar notation to previous work.

3.1 Definitions and Data Structures

As input, our algorithm takes an image of the Style Sphere S, along with the 3D geometry G. It produces image O as output.

Each image consists of an array of pixels. In addition to the standard (r,g,b) triple, we also require luminance values y, which can be easily computed as $y = 0.299(r) + 0.587(g) + 0.114(b)$. Together, these four channels comprise the feature vector for each pixel. We denote the complete feature vector of a pixel p in an image I as $I(p)$.

The luminance values alone are used to compare pixels. This serves two purposes. First, because the human eye is more sensitive to changes in luminance than changes in color, this technique has been shown to yield perceptually better results. In addition, using only luminance speeds up image synthesis, as we only compare one value for each pixel in place of three.

3.2 Initialization and Main Loop

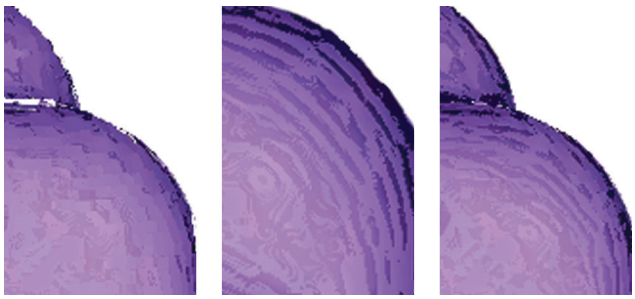

Figure 2: Rendering without environment map (left), Style Sphere (middle), Rendering with environment map (right)

Two images of the 3D scene are first rendered using the hardware. The normal map for the 3D scene is rendered and stored in an image N. The scene is also rendered using the input sphere S as an environment map, as in the Lit Spheres technique. This image is stored as the initial output buffer O in place of the random noise image used by Wei and Levoy. The environment map gives an approximate structure to the strokes, so we use the environment map as a starting point in order to capture the orientation of larger strokes. This yields better results in styles that have strokes oriented in certain directions along the sphere. Figure 2 shows an image rendered without (left) and with (right) an environment map. In the image rendered without the environment map, the continuity and orientation of the strokes is lost, and the style looks different from that of the Style Sphere. In order to use the data from the environment map, we need to compare the complete region around the pixel in consideration, as opposed to previous techniques that only compare the region that has already been rendered (Figure 3).

Next, we loop through the output buffer in scanline order, and replace each pixel with a copy of the "best match" pixel from the Style Sphere. The best match will be a pixel on the Style Sphere that has a similar neighborhood to the target pixel in the output image. Section 3.3 describes the method of finding the best match in detail. In place of using the scan-line order to loop through the pixels, we also experimented with other orders such as fixed interval and random. Changing the order did not have a significant effect on the output, so the scan-line order was used for efficiency and simplicity.

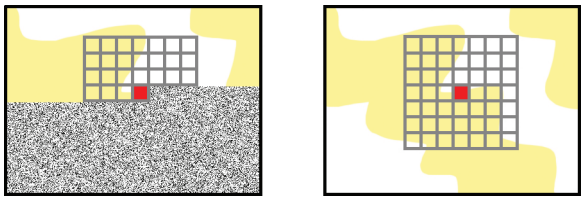

Figure 3: Search regions used by previous techniques (left) and our technique (right)

The following pseudocode implements the above algorithm:

function StyleSpheres(S, G):
  Render image N, as normal map of G
  Render image O, as G sphere mapped with S
  for each pixel q in O in scan-line order
    p = BestMatch($q, S, O, N(q)$)
    $O(q) = S(p)$
  return O

3.3 Finding the Best Match

The BestMatch function is the key to our algorithm's success. Its role is to find a pixel in the Style Sphere to replace the one in the output image. The function takes the current pixel in the output image q, the Style Sphere S, the output image O, and the normal n for the geometry at q. It loops through all the pixels in the search region around the normal n, and comparing the local neighborhood with the neighborhood around q. Points are weighted by a Gaussian centered at the search center so that points close to the center contribute more to the match. The optimal size of the Gaussian will depend on the texture of the substrate (such as paper, canvas, etc.). For fine textures, a smaller sized filter is sufficient. In practice, we found that a 5×5 filter was adequate for most textures.

Unlike some previous techniques, we do not need to search though the whole texture, since it is likely that the pixels in both images will have similar surface normals. Since the normal information is known to us, we can use it to target the search process. The normal n at pixel q is mapped to a pixel on the Style Sphere, and the search area is reduced to a small region around it. We found that a search area of 5×5 performed best in practiec; increasing the search area further did not result in noticable improvements in the quality. Previous techniques used multi-resolution techniques in order to accelerate the search process, but this step is not necessary for our algorithm, as our search area is already small enough. All of the examples shown in this paper were rendered using a search area of 5×5 with Gaussian weighting.

Since we are operating at the pixel level, the size of the texture of the substrate remains consistent over the different areas of the image. Further, the size of the 3D geometry does not affect the size of the texture, which remains consistent with that of the Style Sphere. Figure 4 shows the texture at different parts of a rendered image, as well as that on the Style Sphere.

Following is the pseudocode for the "best match" step:

function BestMatch(q, S, O, n)
  Start with a best match b = pixel in S at normal n
  for each pixel k in search region around n in S
    for each pixel l in neighborhood of k
      Find corresponding pixel m in neighborhood of q
      Compute weighted diff between $S(l)$ & $O(m)$
    Compute the average weighted difference
    if k is a better match than b
      $b = k$
  return b

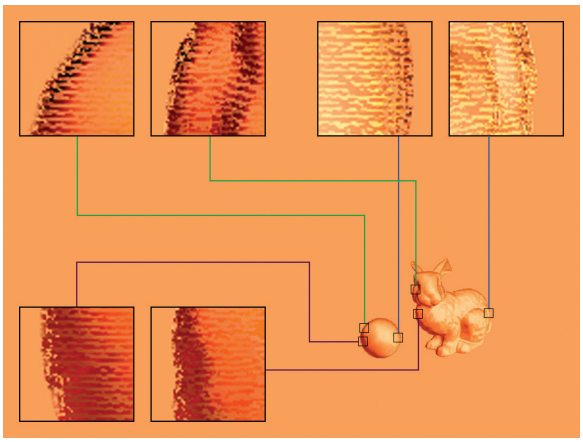

Figure 4: Different parts of a rendered image and corresponding parts on the Style Sphere. Notice that the texture remains consistent in size as well as style.

3.4 Silhouettes

Figure 5: An edge of a model, without rendering the silhouette (left) and with the silhouette (right)

Properly capturing the style of the silhouettes is a crucial component to accurately reproducing the intent of the artist. Without special handling, the object silhouettes will appear perfectly crisp and will tend to look synthetic and unnatural. Figure 5 shows model edges with and without the silhouette rendered from the Style Sphere. The edge where the silhouette is rendered artistically (right) looks much more natural.

In order to render the silhouettes, we map the pixels that lie near them to a point on the Style Sphere's silhouette. We first determine the point on the edge of the geometry that lies closest to the pixel on the silhouette, and compute the vector from the edge point to the pixel being drawn. Using the normal of this edge point, a corresponding point on the edge of the Style Sphere is calculated. We then perturb this Style Sphere edge point using the vector from the previous step, yielding a new point inside the Style Sphere. The BestMatch function is then used to determine the color of the pixel. We render the silhouettes upto a distance of 10 pixels from the edge of the geometry. In place of using the scan-line order, they are drawn during a separate rendering pass, in increasing order of their distance from the edge of the geometry. This helps improve the quality of the edges, as they have not been sphere mapped, and thus there is less noise than that in the scan-line order.

In cases where the silhouette is thin, it need not be explictly rendered. If the normals are mapped to the Style Sphere such that the silhouette edges lie just inside the sphere, they will be rendered automatically. This also helps in rendering the feature edges that lie inside the models. There is not always an easy solution, however, as the Style Sphere's silhouette edges may not have a consistent width. We obtained the best results by mapping the normals such that the silhouettes lied close to the edge of the Style Sphere.

4 Discussion

We have described a framework that allows users to intuitively create and specify a rendering style by example. In addition to improving on the quality of previous methods, we have also improved rendering speed by three orders of magnitude (hours to seconds). When using a 256×256 input sphere and rendering an 800×800 output image, Hertzmann et al.'s method took three hours, as compared to our framework, which took 11 seconds on the same computer. Since the normal map and the environment map are rendered very quickly using graphics hardware, our rendering time is proportional to the size of the output image and not the geometry.

The spheres used for testing were derived from mainly two sources. Some of the spheres were drawn by art students using different physical media and then scanned, while others were created in Adobe Photoshop. The system was successfully able render the 3D geometry in similar styles for most cases.

It is important to note that the results produced by our system may not reproduce a particular effect as faithfully as a special algorithm for producing the specific style. However, our results are qualitatively quite good, and we allow users to easily experiment with new styles without adjusting any parameters. We feel that this extremely intuitive interface to style-by-example makes our texture synthesis based approach attractive for novice users or for users who desire a wide variety of rendering styles.

4.1 Limitations

Wei and Levoy's texture synthesis works best on natural textures, and can fail on textures with very regular structure. Because we rely directly on this algorithm to find matching pixels in the style sphere, our system can also fail in these cases. Figure 6 shows two such cases. On the left, the small dots on the Style Sphere are not reproduced. This failure can not be completely attributed to the texture synthesis application, however; the environment mapped rendering that we use as a starting point enlarges and stretches the dots, making it even harder for the texture synthesis algorithm to find a good match. On the right, the style sphere is made up of strokes aligned along the sphere in multiple directions. In places where there are strokes only in one direction, the framework can detect the strokes and render them effectively, but in places where there are strokes in multiple directions, the framework can not detect the structure and thus the strokes are lost. Although the final rendering looks reasonable, it is not a faithful reproduction of the original style.

Figure 6: Failure cases for Style Spheres. These drawbacks mainly come from limitations of the texture synthesis method, which has difficulty capturing fine local structure.

5 Future Work

Style Spheres are quite effective for reproducing many different styles of drawing, but there are many areas for improvement:

Highly curved or sharp areas: In areas of high curvature, the rapid change in surface normal can lead to a poor match in the style sphere. Even more fundamentally, the artist might want to draw sharp edges in the model with a certain style, but there is no opportunity to show that style with only a sphere as input. We would like to provide additional sample drawings of simple objects with different local surface properties, such as cubes (for edges and corners).

Brush strokes: Our system currently operates only in image space and might therefore miss higher-level attributes of the brush strokes. If the user were to use the computer to paint the Style Sphere itself, we could extract stroke features from the sample drawing and make more intelligent choices about how to render the 3D model.

Structured samples: In the last section, we showed examples of styles where our technique failed to capture the local structure of the input. We would like to experiment with newer texture synthesis algorithms that better support these kinds of sample inputs. In particular, we would like to be able to accurately capture the fine stroke detail in pen-and-ink drawings.

Mesh deformation: In cases where the input sphere is not perfectly round, the rendered model also appears deformed. Although this can be the desired effect, it can leaves gaps in the rendered geometry. A system that deforms the geometry before rendering would help remove this artifact.

Animation Currently, the system produces only still images, and will not maintain frame to frame coherence in animation. A natural extension to the system would be to match features between two frames to maintain frame to frame coherence. If the brush strokes can be separated from the texture of the substrate, as discussed earlier, the system could also maintain coherence by using a fixed texture for the substrate, and applying the brush strokes on the geometry.

6 Acknowledgements

We thank Prof. Achimescu and his art class for providing valuable input in interpreting the results, as well as providing us with various different Style Spheres.

Figure 7: Photoshop Dry Brush Filter

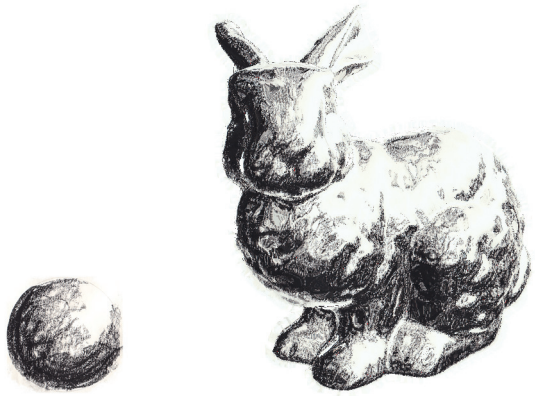

Figure 8: Charcoal

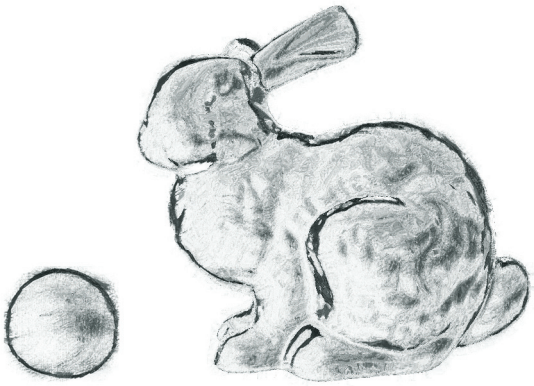

Figure 9: Ink

References

ASHIKHMIN, M. 2001. Synthesizing natural textures. In *Symposium on Interactive 3D Graphics*, 217–226.

CURTIS, C. J. 1998. Loose and sketchy animation. In *ACM SIGGRAPH 98 Electronic art and animation catalog*, ACM Press, 145.

DANIELS, E. 1999. Deep canvas in disney's tarzan. In *ACM SIGGRAPH 99 Conference abstracts and applications*, ACM Press, 200.

GOOCH, B., AND GOOCH, A. A. 2001. *Non-Photorealistic Rendering*. A. K. Peters Ltd.

GOOCH, A., GOOCH, B., SHIRLEY, P., AND COHEN, E. 1998. A non-photorealistic lighting model for automatic technical illustration. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, ACM Press, 447–452.

HALPER, N., SCHLECHTWEG, S., AND STROTHOTTE, T. 2002. Creating non-photorealistic images the designer's way. In *Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering*, ACM Press, 97–104.

HERTZMANN, A., JACOBS, C. E., OLIVER, N., CURLESS, B., AND SALESIN, D. H. 2001. Image analogies. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, ACM Press, 327–340.

HERTZMANN, A., OLIVER, N., CURLESS, B., AND SEITZ, S. M. 2002. Curve analogies. In *Proceedings of the 13th Eurographics workshop on Rendering*, Eurographics Association, 233–246.

JODOIN, P., EPSTEIN, E., GRANGER-PICH, M., AND OSTROMOUKHOV, V. 2002. Hatching by example: a statistical approach. In *Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering*, ACM Press, 29–36.

KALNINS, R. D., MARKOSIAN, L., MEIER, B. J., KOWALSKI, M. A., LEE, J. C., DAVIDSON, P. L., WEBB, M., HUGHES, J. F., AND FINKELSTEIN, A. 2002. WYSIWYG NPR: drawing strokes directly on 3d models. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, ACM Press, 755–762.

MEIER, B. J. 1996. Painterly rendering for animation. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, ACM Press, 477–484.

PRAUN, E., FINKELSTEIN, A., AND HOPPE, H. 2000. Lapped textures. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., 465–470.

PRAUN, E., HOPPE, H., WEBB, M., AND FINKELSTEIN, A. 2001. Real-time hatching. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, ACM Press, 579–584.

SLOAN, P., MARTIN, W., GOOCH, A., AND GOOCH, B. 2001. The lit sphere: A model for capturing NPR shading from art. In *Proceedings of Graphics Interface 2001*, B. Watson and J. W. Buchanan, Eds., 143–150.

STROTHOTTE, T., AND SCHLECHTWEG, S. 2002. *Non-Photorealistic Computer Graphics: Modeling, Rendering and Animation*. Morgan Kaufmann.

Figure 10: Pen

WEI, L., AND LEVOY, M. 2000. Fast texture synthesis using tree-structured vector quantization. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., 479–488.

WEI, L., AND LEVOY, M. 2001. Texture synthesis over arbitrary manifold surfaces. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, ACM Press, 355–360.

WINKENBACH, G., AND SALESIN, D. H. 1994. Computer-generated pen-and-ink illustration. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, ACM Press, 91–100.

Figure 11: Oil Paint

Figure 12: Photoshop Fresco Filter

Figure 13: Pencil