# Artistic Rendering of Digital Images

## Implementation of Texture Transfer Algorithm

Allison Card
Graduate School of Business
Stanford University
Stanford, CA
acard@stanford.edu

***Abstract*—Artistic rendering is a subset of non-photorealistic rendering with the intention of using artistic effects in rendered images. This paper looks at texture transfer as a generalized method to render digital images in artistic styles. Texture transfer algorithms transfer the source texture to a target image. Since artistic style is subjective, this algorithm has four adjustable parameters that allow a user to effect the style of the resulting image. This paper also suggests some starting values for the parameters that have been found to work best for most images. Finally, texture transfer works well when the source texture has high frequency components, is similar to target image, and has the desired artistic style.**

***Keywords—image processing; non-photorealistic rendering; artistic rendering; texture transfer; texture synthesis; image quilting; drawing; painting***

## I. Introduction

Historically, Computer Graphics research has focused on creating realist renderings of objects and scenes. This is called photo realism. However, at the turn of the millennium a branch of research was created focusing on rendering images with stylistic effects like brush strokes and crosshatching [1]. This is known as non-photorealistic rendering. Artistic rendering, a subset of non-photorealistic rendering, allows for a greater range of stylist presentations and dramatic effects and a greater ability to focus the attention of the viewer [1]. In the modern day and age, artist rendering can be used to turn everyday photos into works of art.

While there are many different methods for artistic rendering, most of them are specialized to a specific artistic effect. For example, Hong and Liu developed an algorithm that replicated the impressionist style of pointillism [2]. However, the emphasis of this paper is on creating artistic rendering of all types, so a texture transfer algorithm is used due to its generality.

Texture transfer algorithms take in an image to be used as the source texture and a target image. The target image is transformed in such a way as to replicate the high frequency texture components of the source image, but still keep the general shapes in the target image. This paper includes Section II, discussing related work around texture transfer, Section III, detailing the algorithm implemented, Section IV, showing results, and Section V, discussing future considerations.

Identify applicable sponsor/s here. If no sponsors, delete this text box (*sponsors*).

## II. Related Work

Hertzmann et. al. [5] and Efros and Freeman [6] were some of the first to use texture synthesis algorithms to transfer an artistic style to an image. Hertzmann et. al. used an image analogy to train their program to replicate an artistic style. This multiscale auto regression takes in A, A', and B and returns B', where A:A'::B:B'. Efros and Freeman's algorithm, on the other hand, stitches together small pieces of the source texture to create an image that looks similar to the target image. They refer to this process as image quilting.

The difficulty with these algorithms is that they are quite slow [3]. Creating the desired artistic effect may require many iterations of the algorithm with a wide variety of parameters. Ashikhmin developed a fast texture transfer algorithm which performed significantly faster than these previous algorithms [3]. He used a method called the coherent synthesis technique, which examines a single pixel during the texture transfer process [4] rather than looking at chunks of the image. Unlike Hertzmann et. al., he does not need to train before the transfer process, and unlike Efros and Freeman, he does not need to consider such a large number of candidate tiles.

## III. Algorithm

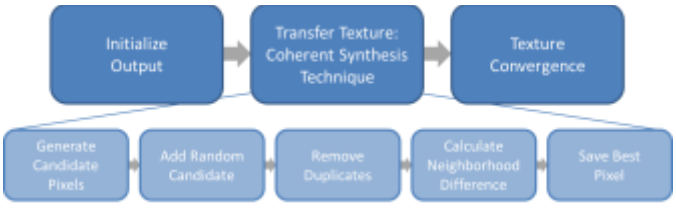



Fig. 1. Overview of algorithm.

This paper implements the fast texture transfer algorithm [3] with some modifications. As seen in Fig. 1, there are three main parts to this algorithm. The meat of the texture transfer process and where most of the processing comes into play is step two. This is the pixel by pixel coherent synthesis technique [4]. The coherent synthesis technique is detailed in the first subsection to facilitate better understanding of steps one and three. Detailed next is the first step of the algorithm, setting up the initial output such that the coherent synthesis technique can run smoothly. The final part of the algorithm adds additional iterations to eliminate harsh edges and allow the texture to converge.

### A. *Texture Transfer*

This algorithm synthesizes texture from the source image in a pixel by pixel process. The output array is first initialized (see Section III. B.) and then the algorithm moves through the output array in scanline order. For each pixel, there are five steps (also pictured in Fig. 1):

- Generate candidate pixels.
- Add random candidate pixel with probability *p*.
- Remove duplicate candidates.
- Calculate neighborhood difference for all candidate pixels.
- Save candidate pixel with smallest difference.

The algorithm considers a n-by-n neighborhood surrounding the current pixel to generate new candidate pixels. Fig. 2 shows an example of this using a 3x3 neighborhood. The resulting image contains the current pixel in black, 4 completed pixels in the 3x3 neighborhood in blue, and 4 uncompleted pixels in the 3x3 neighborhood in white.

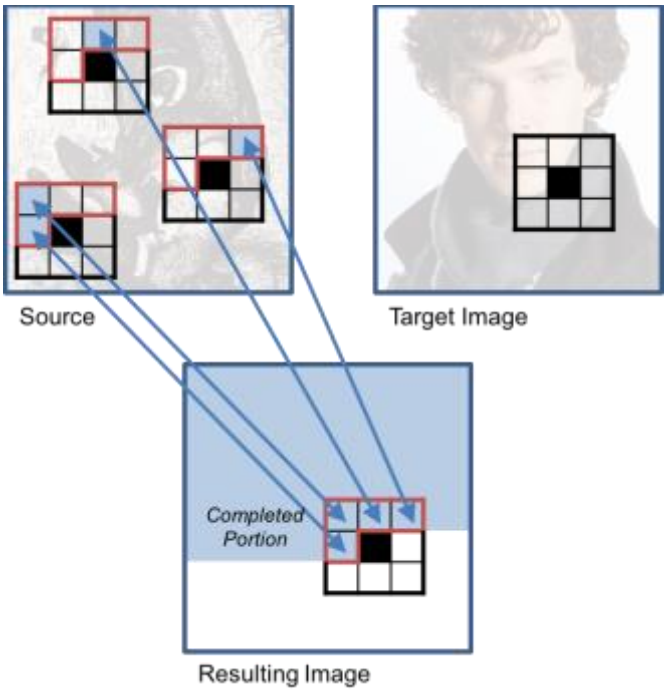



Fig. 2. Coherent synthesis technique. Diagram modified from [3]. Image sources [1] and [2].

The previously completed pixels are looked up in the source image. The new candidate pixels are chosen based on the location of the previous pixel in relation to the pixel currently under consideration. Fig. 2 shows the previously chosen pixels in the source image. The new candidate pixels are in black.

Finding candidate pixels in this way allows the algorithm to grow areas of texture in the resulting image. However, it can also lead to harsh edges in the resulting image when the algorithm runs into the boundaries of the source image or changes texture areas. Therefore, a random candidate pixel from anywhere in the source image is added to the candidate list. This reduces the size of the texture growth areas allowing for smoother transitions. A probability of 0.05 is generally sufficient to reduce these edge effects [3], though greater probabilities may create superior artistic effects.

Since the algorithm chooses new pixels based on the location of the pervious pixels in the source image. There will likely be duplicate candidates in the candidate list. An example of this can be seen in Fig. 2. To increase the speed of the algorithm, duplicate candidates are removed. While this isn't really necessary when considering a neighborhood of 3x3, it becomes much more necessary when considering larger neighborhoods as the number of candidates grows quadratically.

To find the neighborhood difference for every candidate pixel, this paper used a similar algorithm to Ashikhmin's [3] with some changes. Ashikhmin kept the colors of the target image in his algorithm, but the stylistic effects of using the source colors, as Efros and Freeman [6] did, are preferable for dramatic renditions of everyday photos. The neighborhood difference is calculated using the difference in intensity between the source and the target image and the L2 distance between the completed portion of the resulting image and the L-shaped neighborhood of the candidate pixel:

$$D^2 = w(\overline{N_s}-\overline{N_t})^2 + (1/n)^2 L_2(N_{rL}, N_{sL}) \quad (1)$$

$N_s$ refers to the neighborhood of the candidate pixel in the source image, $N_t$ refers to the neighborhood of the pixel under consideration in the target image, and $N_r$ refers to the neighborhood of the candidate pixel in the resulting image. In Fig. 2, the neighborhood size is 3x3, so 9 pixels are considered. The average intensity value (indicated by the bar over $N_s$ and $N_t$) of this neighborhood is used such that no one pixel bears too much weight. This allows the algorithm to pull textures with large intensity variations into a section of the resulting image. This part of the neighborhood difference is weighted with parameter *w*. A value of one tends to work well for most use cases [3]. However, it can be changed for different stylistic effects.

In the second half of (1), *n* refers to the number of pixels in the L-shaped neighborhood (e.g., 4 in Fig. 2). This normalizes the importance of the distance between the source and the result. The L2 distance, the Euclidian distance between all rgb values, is taken between the L-shaped neighborhood in the resulting image and the L-shaped neighborhood in the source image. In Fig. 2, these L-shaped neighborhoods are highlighted with a pink boarder.

When the candidate with the smallest neighborhood difference is found, the pixel rgb values and the location in the source image are stored and the algorithm moves on to the next pixel in scanline order.

### B. *Initialization of Output*

The coherent synthesis technique requires that an L-shaped neighborhood of completed pixels be available for every new pixel synthesized in the resulting image. Ashikhmin copies the target image into the result and uses these values to compute the candidate pixels and the neighborhood difference [3]. However,

this can create harsh edges in the resulting image especially around the image boarder. To avoid this issue, create an extra boarder of width N/2 around the resulting image. The pixels in this boarder are randomly assigned from the source image. This allows the first couple of pixels and edges of the resulting image to be easily computed, see Fig. 3. For a 3x3 neighborhood, the border width would be one pixel on three sides of the image.

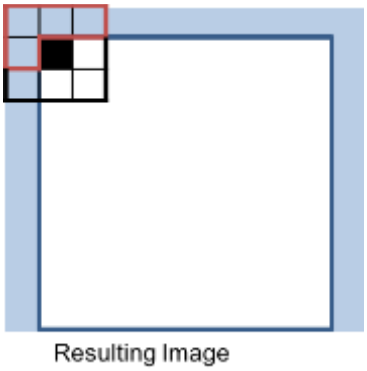



Fig. 3. Initialization of output image.

### C. Texture Convergence

While adding in random candidate pixels and initializing the output image help reduce harsh edges in the resulting image, these are not always sufficient. To ensure that the edges in the image converge, the texture transfer portion of the algorithm can be iterated over several times [4] [6]. The algorithm used in the second plus iteration is mostly the same as that used in the first iteration. However, there are changes to two steps detailed in Section III. A.:

- Generate candidate pixels.
- Calculate neighborhood difference for all candidate pixels.

Instead of only looking at the previously completed pixels in the L-shaped neighborhood to generate candidates, the algorithm now generates candidates from the entire surrounding neighborhood. Fig. 4 shows the second iteration over the resulting image. The L-shaped neighborhood of resulting pixels from the second iteration are used to generate candidate pixels, and the bottom L-shaped neighborhood of resulting pixels from the first iteration are also used to generate candidate pixels.

The neighborhood difference between the source and the target remains the same. However, the L2 difference between the resulting image and the source image now uses pixels from the entire neighborhood:

$$D^2 = w(\overline{N_s} - \overline{N_t})^2 + (1/n)^2 L_2(N_r, N_s) \quad (2)$$

As seen above, (2) is very similar to (1). *n* now refers to the number of pixels in the total neighborhood (e.g., 9 in Fig. 4) instead of the L-shaped neighborhood.

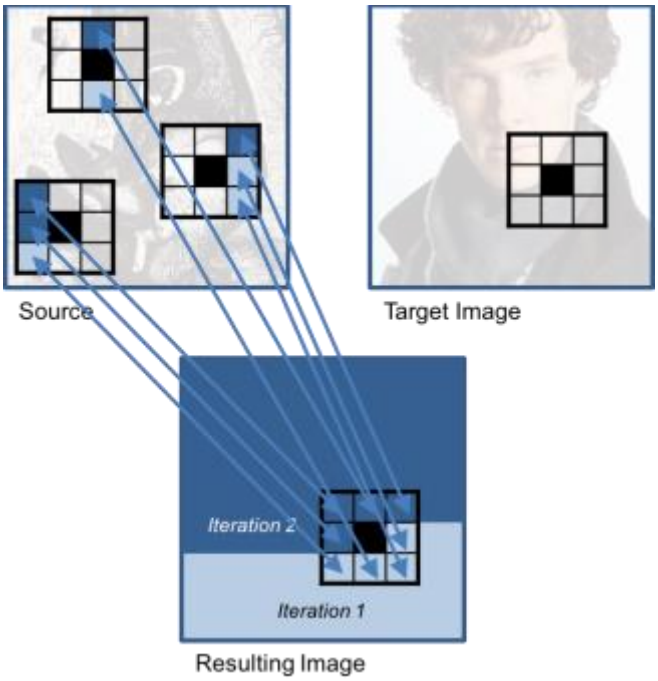



Fig. 4. Texture convergence. The second iteration over the resulting image. Diagram modified from [3]. Image sources [1] and [2].

## IV. Results

### A. Parameters

There are four parameters the can be adjusted within this algorithm:

- Neighborhood size (n).
- Probability of adding a new pixel (p).
- Weight on average intensity difference between the source and target (w).
- Number of iterations (i).

The neighborhood size affects the ability of the algorithm to add textures of varying frequencies to the resulting image. The smaller the neighborhood the higher the texture frequency must be. The neighborhood size also affects the convergence of the resulting image as larger neighborhoods will provide a greater number of candidate pixels to be considered. The effects of neighborhood size can be seen in Fig. 5. A 5x5 neighborhood seems to work well for most source images. The 3x3 neighborhood has too many sharp edges, and the 7x7 loses some of the high frequency texture.

The probability of adding a new pixel affects the smoothness of the resulting image. Considering a random candidate pixel causes smaller areas of texture growth. Higher probabilities increase smoothness because the smaller texture areas fit together better, but smaller texture areas mean fewer lower frequency texture components in the resulting image. Fig. 5 shows the effects of different probabilities. A probability of 0.2 seems to work well for most images. The 0.05 probability causes the resulting image to have too many sharp edges, while the


Fig. 5. Variation of output to parameters. Image sources [1] [2] [3].

resulting image with the 0.5 probability starts to lose the low frequency texture components.

The weight on the average intensity difference between the source and the target affects the amount of detail from the target image that is shown. The effects of the weight can be seen in Fig. 5. A weight of 1 seems to work well for most images. The 0.5 weight causes the resulting image to appear blurry, while the weight of 2 reduces the use of low frequency texture components.

Depending on the source image, the resulting image may converge with only one iteration. However, some source textures have different color components with similar intensities. This causes obvious edges in the resulting image. The number of iterations required to converge the image depend greatly on both the source texture and the target image. In Fig. 5, it takes five iterations for the resulting image to converge, though it has mostly converged by iteration 3.

### *B. Evaluation of Technique*

This algorithm works well for certain types and combinations of source textures and target images. When the intensity in the foreground/background of the source texture and target image are similar, the resulting image contains similar dramatic effects to the source. Fig. 6 (a) shows an example of this. The algorithm also works when there are high frequency texture components in the source texture. Fig. 6 (a) has small line segments, and Fig. 6 (b) has small brush strokes. Finally, the source texture should contain the artistic components desired in the resulting image.

The algorithm doesn't work well for a number of source textures and target image combinations. Smooth source textures almost never work as the algorithm cannot pick up on such low frequency changes. Source images that have both smooth and high frequency components also don't work well. Fig. 6 (c) shows the discontinuities in the resulting image. While a larger number of iterations will make this image look better, it is impossible to fully remove these edges. Fig. 6 (c) also shows the undesirability of strong directional lines in the source image.

The resulting image will look best if the target image has standard components of artistic style. For example, a single strong light source, good composition, and a clear separation of foreground and background. Fig. 6 (d) shows an example of this problem. Finally, the algorithm doesn't take into account the types of objects in the image. For example, the resulting image in Fig. 6 (b) doesn't show the night sky in Starry Night as the background as one might desire.

## V. Future Considerations

As mentioned in Section IV. B., there are a number of source textures and resulting images where this algorithm doesn't work as well as desired. Further work on shapes [8] and directionality of gradients [7] would help follow the lines in the target image.

Some preprocessing of the target image could enhance the artistic qualities of the result. In particular, cropping the image to create a better composition, sharpening the contrast of the image to create the illusion of a stronger light source, and blurring the background would likely improve the quality of the result.

Finally, the resulting image would be much more interesting if the objects in the source image could be aligned with the objects in the target image. E.g., if there is sky in both images, then the resulting image should have a sky similar to that of the source image. Gatys, Ecker, and Bethge use neural networks to create better symmetry between the source texture and resulting image [9].


Fig. 6. Results. (a) Parameters: n=5x5, p=0.2, w=1, i=1. Image sources [1] [2]. (b) Parameters: n=5x5, p=0.2, w=1, i=5. Image sources [2] [3]. (c) Parameters: n=5x5, p=0.2, w=1, i=1. Image sources [2] [4]. (d) Parameters: n=5x5, p=0.2, w=1, i=1. Image sources [5] [6]. See Appendix for larger resulting images.

## ACKNOWLEDGMENT

Thank you to Professor Gordon Wetzstein for providing input and direction for this project.

## REFERENCES

[1] J. Romero and P. Machado, "Evolutionary search for the artistic rendering of photographs," in *The Art of Artificial Evolution: A Handbook on Evolutionary Art and Music*. Berlin: Springer, 2008. 39-62.

[2] Y. Hong and T. Liu, "Create pointillism art from digital images," unpublished.

[3] M. Ashikhmin, "Fast texture transfer," in *IEEE Computer Graphics and Applications*, vol. 23, no. 4, pp. 38-43, July-Aug. 2003.

[4] M. Ashikhmin, "Synthesizing natural textures," in *Proceedings of the 2001 symposium on Interactive 3D graphics* (I3D '01). ACM, New York, NY, USA, 217-226.

[5] A. Hertzmann, C. Jacobs, N. Oliver, B. Curless, and D. Salesin, "Image analogies," in *Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (SIGGRAPH '01). ACM, New York, NY, USA, 327-340.

[6] A. Efros and W. Freeman, "Image quilting for texture synthesis and transfer," in *Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (SIGGRAPH '01). ACM, New York, NY, USA, 341-346.

[7] H. Lee, S. Seo, S. Ryoo, and K. Yoon, "Directional texture transfer," in *Proceedings of the 8th International Symposium on Non-Photorealistic Animation and Rendering* (NPAR '10). ACM, New York, NY, USA, 43-48.

[8] T. Mertens, J. Kautz, J. Chen, P. Bekaert, and F. Durand, "Texture Transfer Using Geometry Correlation," in *Rendering Techniques* 273. 2006.

[9] L. Gatys, A. Ecker, and M. Bethge., "A neural algorithm of artistic style," preprint. 2015.

## IMAGE SOURCES

[1] *Untitled* by Enrico Donati. http://picclick.co.uk/Enrico-DONATI-original-lithograph-381762876134.html

[2] Benedict Cumberbatch. http://www.bbcamerica.com/anglophenia/2013/09/benedict-cumberbatch-as-an-actor-youre-looking-for-the-infinite

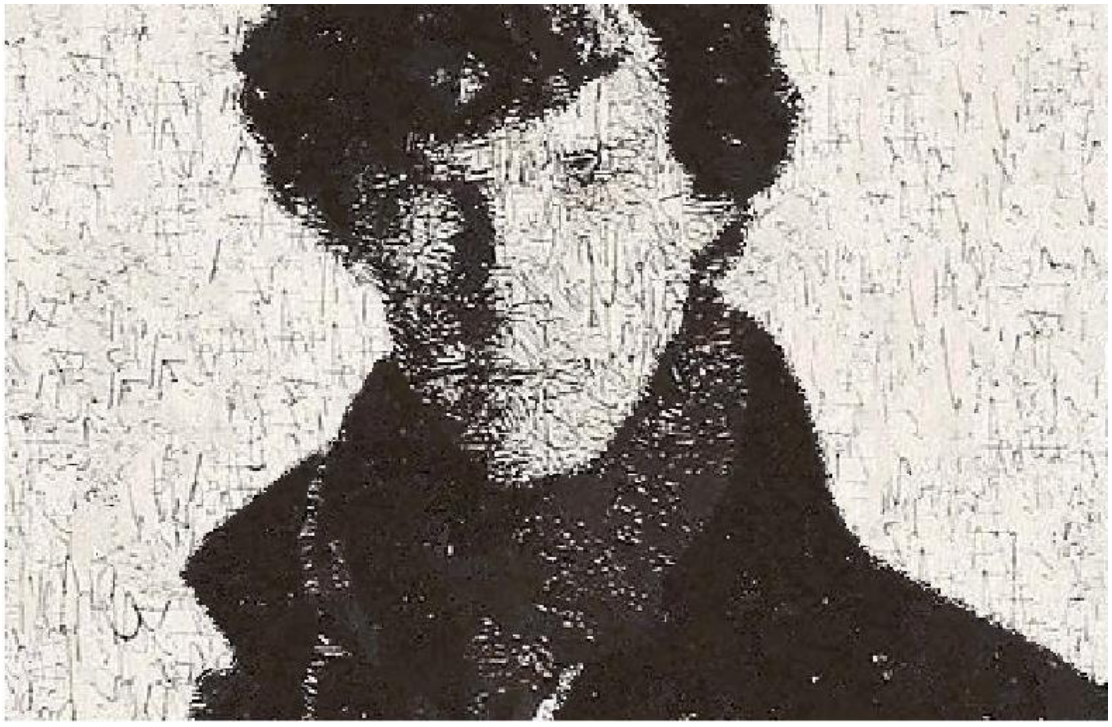
[3] *Starry Night* by Vincent van Gogh. http://wallpaperswide.com/the_starry_night-wallpapers.html

[4] *After Rembrandt* by BrokenUmbrella. http://brokenumbrella.deviantart.com/art/After-Rembrandt-19437939

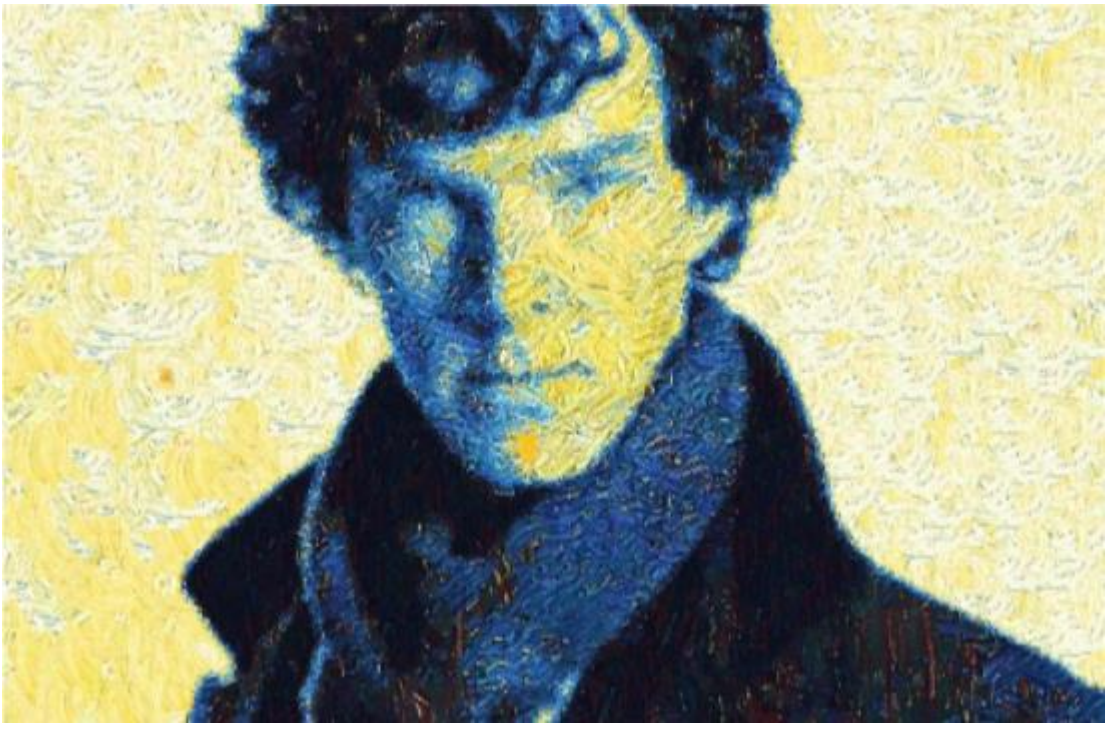
[5] *Autumn Rhythm no 30* by Jackson Pollock. https://dimensionviva.wordpress.com/tag/jackson-pollock/

[6] Unpublished

## APPENDIX



Resulting image from Fig. 6 (a).



Resulting image from Fig. 6 (b).

Resulting image from Fig. 6 (c).

Resulting image from Fig. 6 (d).