Efficient Rendering of Linear Brush Strokes

Apoorva Joshi

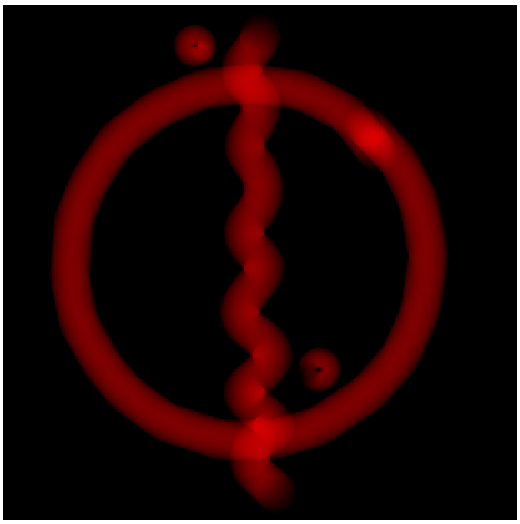


Figure 1. Compound multi-line strokes rendered with our technique.

Abstract

We introduce a fast approach to rendering brush strokes with variable hardness, diameter, and flow for raster image-editing applications. Given an N-pixel-long linear brush stroke with diameter M, our approach reduces the time complexity from $O(NM^2)$ to $O(NM)$, while enabling the stroke to be rendered in a single GPU draw call and while avoiding overdraw.

1. Introduction

In an image editor, when the brush tool is being clicked or dragged, we draw a brush stroke along the mouse's drag path. If the image editor is running at interactive frame rates, you can chain line segments between mouse drag positions to draw arbitrary-shaped brush strokes. In any given frame, the brush is stamped along the straight line connecting the last mouse position and the current mouse position. The proposed

 ISSN 2331-7418

method renders the brush stroke along such a line segment—it cannot deal with curves at infinite precision.

We propose a hybrid method between stamping and sweeping, as defined in Diverdi's paper [DiVerdi 2013]; we use the analytical approach of sweeping, but for stamping purposes instead of for physically simulating brush bristles [Xu et al. 2004] or wet paint [Vandoren et al. 2009].

In most image editors, if a brush is only clicked (not dragged), a single impression is created on the image being edited. This single impression is referred to here as a *stamp*. Image editors often provide multiple parameters to customize the brush tool:

- *Diameter*: Controls the diameter of each stamp, usually defined in terms of pixels;
- *Hardness*: Controls how the stamp alpha falls off with distance from the center;
- *Opacity*: Determines the maximum alpha of the brush stroke;
- *Flow*: Controls how much each stamp adds to the stroke. Multiple intersecting stamps sum up their contributions, with the maximum alpha clamped to opacity.

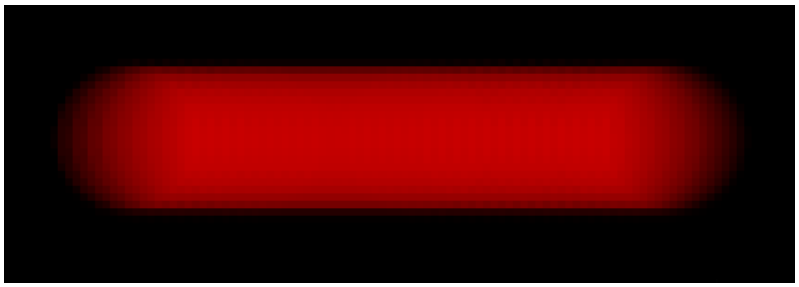


Figure 2. A brush stroke in GIMP. Note how the sides are stamped by fewer circles and, hence, have less intensity.

In the case of tablet-based pressure-sensitivity, the start- and the end-points may have different values for these parameters. The editor interpolates these values as it draws stamps along the line. To sum up, a brush stroke is a series of parametric stamps along a straight line, additively blended together (see Figure 2).

We need to model a stroke as a continuous process, instead of something like a distance function, because not doing so leads to visible boundary artifacts between adjacent stroke segments which share beginning/end points. We also lose the subtler falloff effects at the ends of the line segment. This is why we cannot use a distance-based prefiltering technique [Chan and Durand 2005].

2. Related Work

Based on our tests, Photoshop (CS5) and GIMP (2.8) create brush strokes by additively stamping along the line. That means they follow the Bresenham-esque pixel line joining the stroke start point and the end point, stamping the stamp centrally at each point along it.

3. Our Approach

In a brush stroke, the alpha value at any pixel is proportional to the number of circles that pass through it. Although existing image editors are treating this as a discrete problem, it can be modeled as a continuous one and then integrated numerically for faster results with fewer artifacts.

We treat a stroke as the result of continuously sliding a circle along the stroke axis, integrating the stamp function at each point along the way. If we compute this integral for each point in the stroke in parallel in a shader, we can render the stroke in a single pass and with a single quad.

We render a brush stroke, given a quad (see Figure 3) along with the uniforms N, $R_{\max}$, $R_{\min}$, opacity, H_1, H_2, F_1 and F_2 as defined in Section 3.1. The derivations for the below formulae can be found in the appendices.

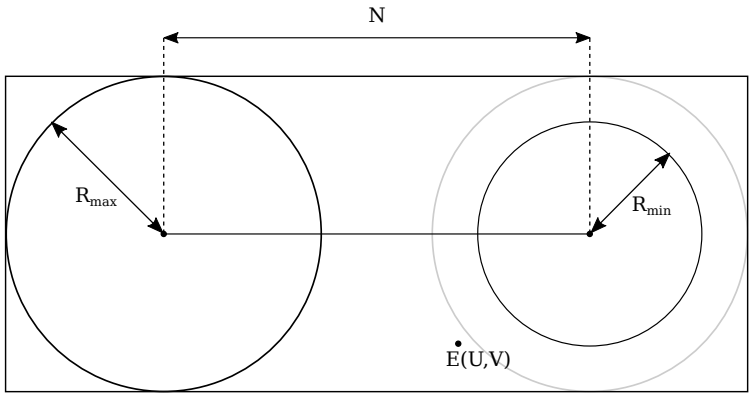


Figure 3. The quad

3.1. Constants

These are the symbolic constants we use:

U, V	UV coordinates of current pixel
N	Length of the stroke in pixels
$R_{\max}, R_{\min}$	Max, min radii in pixels at the ends of the stroke
opacity	Opacity of the stroke
H_1, H_2	Hardness at the ends of the stroke
F_1, F_2	Flows at the ends of the stroke

3.2. Formulae

$$\begin{aligned} \text{aspect} &= \frac{N}{2R_{\max}} + 1 \\ X &= U \times \text{aspect} \\ Y &= V \end{aligned} \qquad \text{(Appendix A)}$$

$$\begin{aligned} \psi(x) &= \frac{R_{\max}(2x-1)}{N} \\ \text{flow}(x) &= \text{lerp}(F_1, F_2, \psi(x)) \\ \text{hardness}(x) &= \text{lerp}(H_1, H_2, \psi(x)) \end{aligned} \qquad \text{(Appendix E)}$$

$$\begin{aligned} \text{radius}(x) &= \text{lerp}\left(0.5, \frac{R_{\min}}{2R_{\max}}, \psi(x)\right) \\ \text{dist}(x, X, Y) &= \sqrt{(X-x)^2 + (Y-0.5)^2} \\ \phi(x, X, Y) &= \frac{\text{dist}(x, X, Y)}{\text{radius}(x)} \end{aligned} \qquad \text{(Appendix D)}$$

You can use your own falloff function. Here's ours:

$$\text{falloff}(x, X, Y) = \begin{cases} 1 & \text{if } \phi(x, X, Y) < \text{hardness}(x), \\ \cos\left(\frac{\pi\phi(x,X,Y)}{2(1-\text{hardness}(x))} + \text{phase}(x)\right)^2, & \text{otherwise;} \end{cases}$$

$$\text{phase}(x) = \frac{\pi}{2}\left(1 - \frac{1}{1 - \text{hardness}(x)}\right).$$

Formulae for finding limits:

$$
\begin{aligned}
BE &= |Y - 0.5| && \text{(Appendix C)}\\
BD &= \text{radius}(X)\\
BO &= \frac{N \times BD}{R_{\max} - R_{\min}}\\
\lambda &= DB^2 - \frac{DB^2.BE^2}{BO^2}\\
\epsilon &= \frac{4DB^4}{BO^2}\\
r_1^2 &= \begin{cases} 0.25 & \text{if } R_{\min} = R_{\max} \\ \frac{(2\lambda+\epsilon)+\sqrt{(-2\lambda-\epsilon)^2-4(\lambda^2+\epsilon BE^2)}}{2} & \text{if } R_{\min} \neq R_{\max} \end{cases}\\
r_2^2 &= \begin{cases} 0.25 & \text{if } R_{\min} = R_{\max} \\ \frac{(2\lambda+\epsilon)-\sqrt{(-2\lambda-\epsilon)^2-4(\lambda^2+\epsilon BE^2)}}{2} & \text{if } R_{\min} \neq R_{\max} \end{cases}\\
X_1 &= \text{clamp}(X - \sqrt{r_1^2 - BE^2}, 0.5, \text{aspect} - 0.5)\\
X_2 &= \text{clamp}(X + \sqrt{r_2^2 - BE^2}, 0.5, \text{aspect} - 0.5)
\end{aligned}
$$

Finally, the alpha for the point (X, Y) is given by

$$\alpha(X,Y) = \min\left(\text{opacity}, 2R_{\max}\int_{X_1}^{X_2} \text{flow}(x)\text{falloff}(x,X,Y)dx\right).$$

We compute the integral numerically using the trapezoidal rule.

4. Blending

While the brush is being dragged and before it is released, we accumulate the stroke into a separate overlay texture. When the mouse is released, this overlay is blended with the image below in a process called *merging*. Merging follows the standard compositing rules [Porter and Duff 1984], but accumulating the overlay does not. Let's see why:

When we're blending subsequent strokes together, the starting point of the second stroke overlaps with the end point of the first one. Let's say for a given pixel, the first stroke contributes 0.5 intensity (alpha) to that pixel, and the second stroke contributes another 0.5 intensity. Because of the additive nature of the process, the pixel should now have an alpha of 1.0.

However, in a traditional alpha-compositing setup, we use `glBlendEquation(GL_FUNC_ADD)` and `glBlendFunc(GL_ONE, GL_ONE_MINUS_SRC_ALPHA)`. Thus, given

source alpha A_s and destination alpha A_d, final resultant alpha A_f is given by $A_f = A_s + A_d(1 - A_s)$. Substituting $A_s = A_d = 0.5$, we get $A_f = 0.75$ instead of the desired value of 1.0.

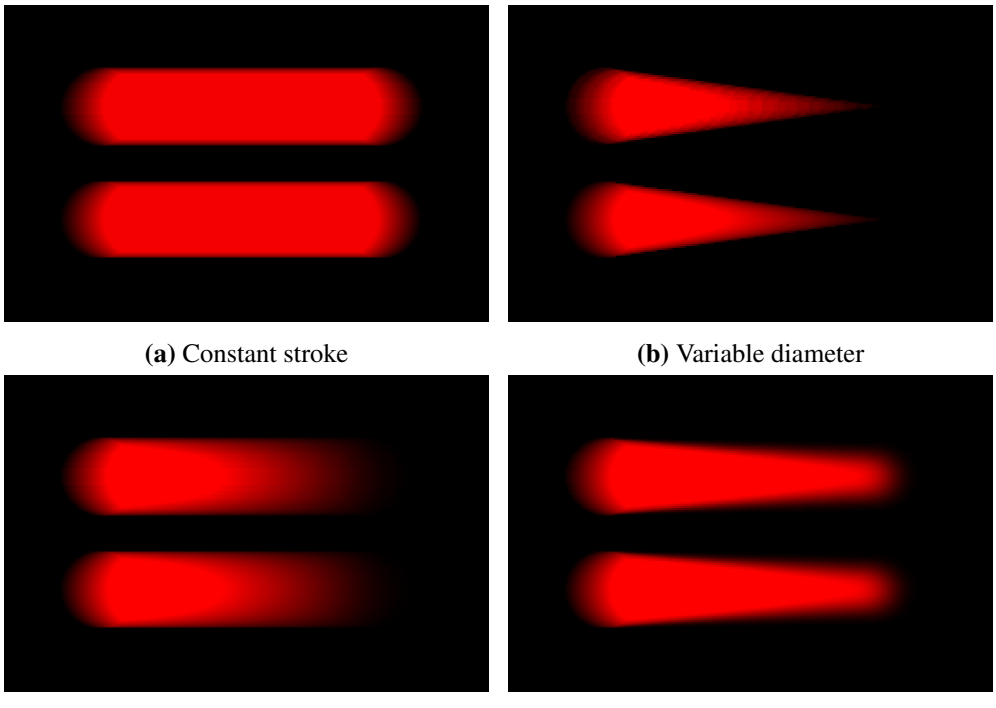
This is why we have to switch to additive blending, but only on the alpha channel. We do this by changing the blending function to `glBlendFuncSeparate(GL_ONE, GL_ZERO, GL_ONE, GL_ONE)`, which means that alpha is additive and RGB is picked from the source (foreground). Thus $A_f = A_s + A_d$, which is what we want.

5. Results

Figure 4 contains single-line stroke comparisons between the discrete approach and our approach.

Figures 5–8 contain comparisons of compound strokes—made up of multiple line strokes—rendered with the discrete approach and with our approach.

Note that as intended, the strokes largely look the same. However, our approach tends to be smoother because we model stamps as a continuous function and hence avoid pixel artifacts present in the discrete approach.



(a) Constant stroke **(b)** Variable diameter

(c) Variable flow **(d)** Variable hardness

Figure 4. Single-line strokes—discrete method (top) vs our method (bottom).

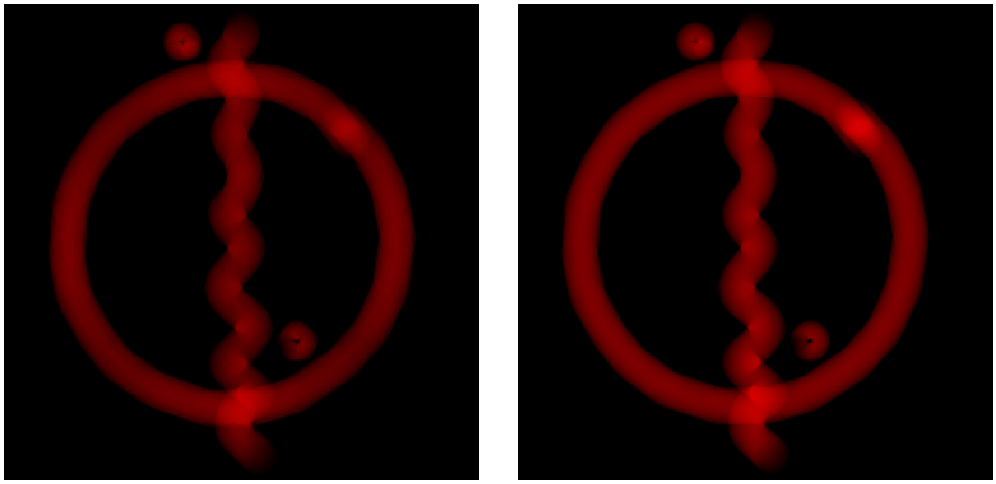


Figure 5. Constant stroke—discrete method (left) vs our method (right).

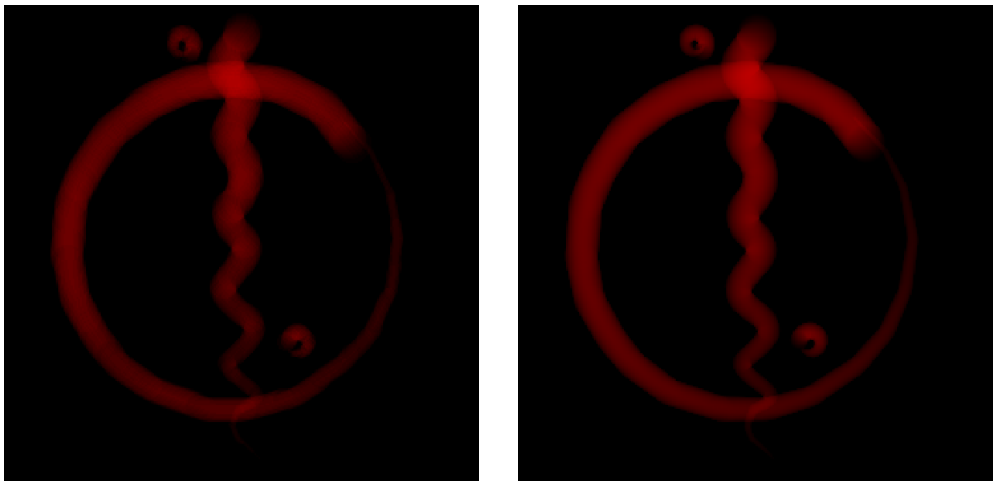


Figure 6. Variable diameter—discrete method (left) vs our method (right).

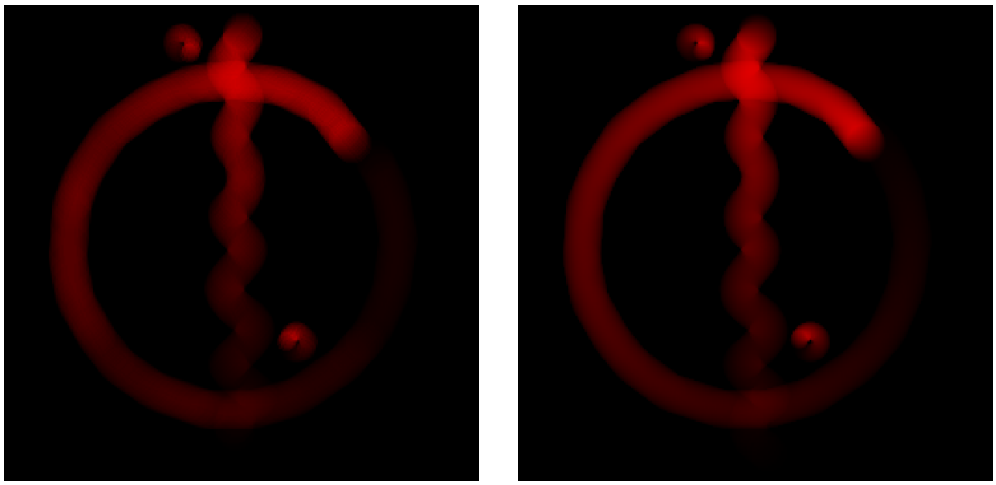


Figure 7. Variable flow—discrete method (left) vs our method (right).

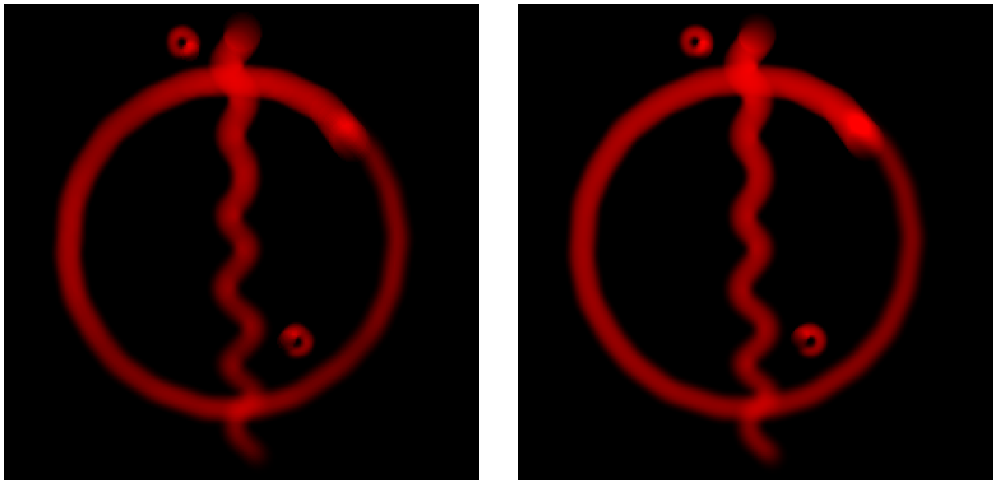


Figure 8. Variable hardness—discrete method (left) vs our method (right).

Our approach can lead to slightly more intense strokes than the discrete approach. This is because in diagonal strokes, the discrete approach spaces out stamps a little bit more along pixel diagonals, as compared to horizontal or vertical strokes. Our approach does not cause lighter diagonal strokes, since the stamp motion is modeled as a continuous slide and not as discrete stamps along a Bresenham-esque pixel line. Even though our approach is arguably more accurate in this regard, the difference is inconsequential for all practical purposes.

6. Limitations

Our technique fundamentally models stamps as a continuous process. In order for this to work, the stamps must be defined as a continuous isotropic function. However, many painting-focused image editors provide the option to use textures as stamps.

Our approach does not handle this use-case. A naive approach would be to approximate this using the following technique: During initialization or brush-loading, we create an accumulated texture of twice the stamp-texture width, and of the same height. In this texture, we compute a sliding sum of the alpha values along the stamp texture's horizontal axis. We can now UV-map this accumulated texture on an arbitrary brush stroke consisting of three quads connected end-to-end to form a rectangle, where we stretch the texture along the middle quad (see Figure 9).

This naive implementation, however, has several flaws. Tapered strokes will not work, since we only accumulate along the horizontal axis. If the stroke is not horizontal, this approach rotates the stamp itself, and since the stamp is not isotropic, strokes will not join up seamlessly end-to-end. Additionally, if the length of the stroke is less than the length of the accumulated texture, we will get incorrect results. This is why our technique cannot be used to render textured brushes. On the other hand, the

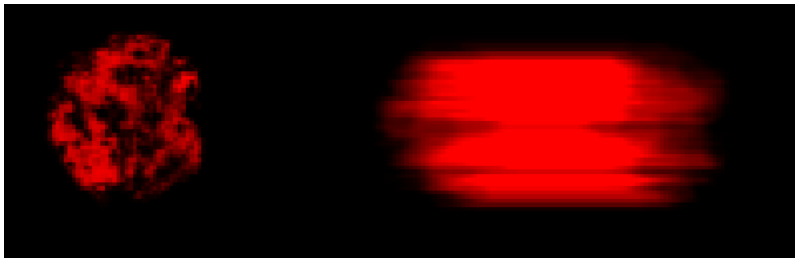


Figure 9. Left: Stamp texture; Right: Accumulated texture.

application can degrade gracefully to rendering multiple quads using the traditional technique of multiple overlapping quads. Another limitation of our approach is that it does not support Bézier curves and it is limited to straight lines.

7. Conclusion

Our implementation thus uses a single quad to render a brush stroke; this allows us to get rid of overdraw, avoid artifacts, and achieve a high degree of parametric stamp control using opacity, flow, hardness and diameter. Our implementation is parallelizable and GPU-friendly.

We allow for varying parameters along the line. Our approach supports a tradeoff between performance and stroke flexibility. Supporting only constant-width strokes will allow us to skip solving for the two radii. Supporting only hard strokes will allow us to skip numerical integration.

Our approach is useful for traditional raster image-editors, where we can support both the pencil tool and the brush tool for non-textured brushes.

Since we can store a stroke as two points and a set of parameters, we can rasterize the strokes at arbitrary resolutions. If we use straight lines to approximate Bézier curves, we can use this technique to enable soft brushes in vector-based illustration applications.

Acknowledgements

We would like to thank Nathan Reed and Vinay Deshpande for their invaluable guidance while researching this subject and Oliver Wang for helping edit this paper on behalf of the Journal of Computer Graphics Techniques.

Appendix A - Coordinate-space Normalization

We do this whole calculation in a pixel shader over a single quad, whose UV coordinates range from $[0, 0]$ to $[1, 1]$. The height of the quad is $2R_{\max}$ pixels, and the width of the stroke is $2R_{\max} + N$ pixels as seen in Figure 3.

To keep our calculation coordinate space spatially uniform, we multiply the UV at the pixel by the aspect ratio of the quad. Thus, after normalization, the X-axis ranges from $[0, \text{aspect}]$, where aspect $= \frac{N}{2R_{\max}} + 1$ and the Y-axis ranges from $[0, 1]$.

Thus, after normalization, the length of the side of one pixel in normalized coordinates is

$$\text{pix\_len} = \frac{1}{2R_{\max}.}$$

Appendix B - Constant Stroke

Before we attempt to fully generalize the problem, we derive the equation for a stroke with constant diameter, hardness, and flow.

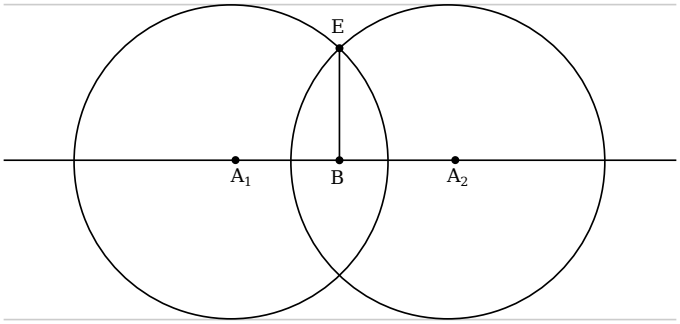


Figure 10. Constant stroke.

Take a stroke of constant radius R and flow F. We will fully generalize to variable-diameter strokes later. We want to derive the alpha value at every point E.

As the circular brush stamp slides continuously from left to right, the alpha at point E will be proportional to the number of stamps that pass through it. The circle with center A_1 is the first circle that overlaps point E, and the the circle with center A_2 is the last. This means that E was inside a stamp for the length of A_1A_2.

We can find out locations of the points A_1 and A_2 easily using the Pythagorean theorem, since we know BE, and since $A_1E = A_2E = R$ for any point E.

In the traditional discrete stamp model, each stamp contributes flow F amount of alpha to a pixel. In our continuous model, we can say that after sliding the circle over the width of a pixel, the resultant alpha is F.

Therefore, the instantaneous alpha any one circle contributes is

$$\frac{F}{\text{pix\_len}} = F \times 2R.$$

Putting the above equations together, the alpha at any point $E(X, Y)$ is equal to

$$\alpha(X, Y) = \min\left(\text{opacity}, \int_{X_1}^{X_2} F \times 2R.dx\right),$$

where X_1 and X_2 are the normalized X-coordinates of A_1 and A_2.

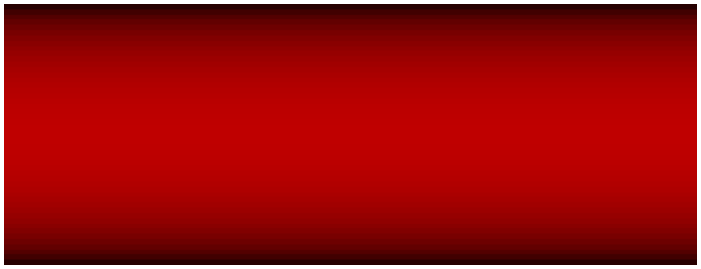


Figure 11. Unclamped constant stroke.

We haven't clamped X_1 and X_2 yet, so our calculations result in the stroke in Figure 11.

If we clamp X_1 and X_2 to the range $[0.5, \text{aspect} - 0.5]$, we get the result we want, as seen in Figure 12.

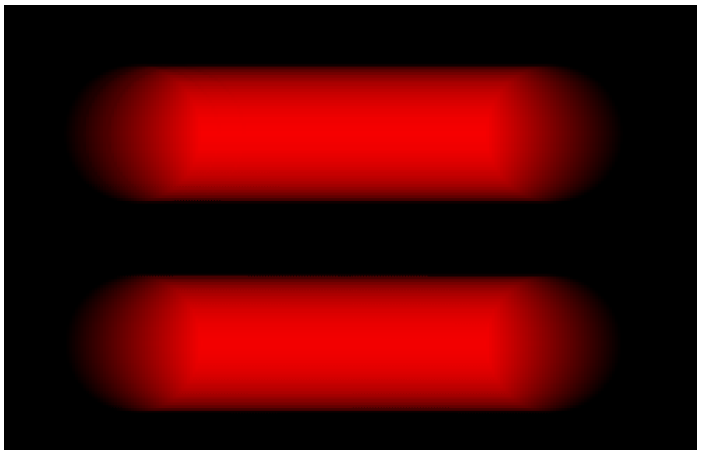


Figure 12. Top: Discrete method (150 quads); Bottom: Our method (Single quad, no overdraw), hardness=1.0, flow=0.02, opacity=1.0.

Appendix C - Variable-diameter Stroke

When brushing with a pressure-sensitive device, pressure can influence brush diameter—the harder you press, the larger the brush size. To support this, we want strokes with linearly interpolated diameters. Just as before, we need to find A_1 and A_2 when given any point E (see Figure 13).

This calculation isn't as simple as the previous case, because the radii of the two circles are also unknown and need to be determined. A_1 and A_2 are the centers of the first and the last circles that contain the point E. Therefore, we need to find two circles that are tangential to the taper line and pass through point E.

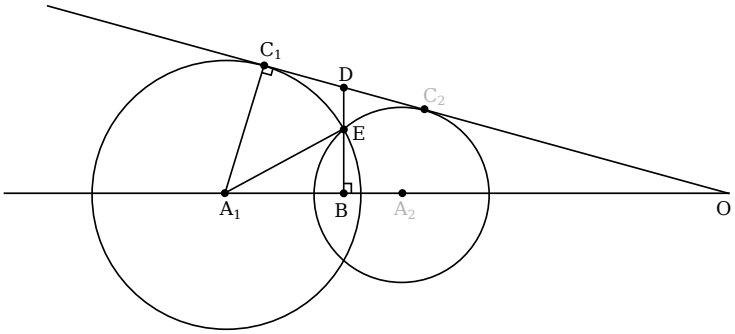


Figure 13. Variable-diameter stroke.

Let's first locate A_1:

$$\begin{aligned}
&\Delta DBO \sim \Delta ACO, \\
&\therefore \frac{DB}{BO} = \frac{AC}{CO}, \\
&\therefore \frac{DB}{BO} = \frac{r}{\sqrt{AO^2 - r^2}}.
\end{aligned}$$

Now,

$$\begin{aligned}
AO &= BO + AB \\
&= BO + \sqrt{r^2 - BE^2}.
\end{aligned} \tag{1}$$

Substituting AO,

$$\frac{DB}{BO} = \frac{r}{\sqrt{(BO + \sqrt{r^2 - BE^2})^2 - r^2}}.$$

After simplifying the equation and introducing a couple of intermediate variables for brevity, we are left with

$$r^4 + (-2\lambda - \epsilon)r^2 + (\lambda^2 + \epsilon BE^2) = 0, \tag{2}$$

where

$$\begin{aligned}
\lambda &= DB^2 - \frac{DB^2.BE^2}{BO^2}, \\
\epsilon &= \frac{4DB^4}{BO^2}.
\end{aligned}$$

This is solvable as a quadratic equation of r^2 because all three constants—DB, BE and BO—are easily evaluated for any point E. The full formulation is provided in Section 3.

For point A_2, we can go back and replace Equation (1) with $AO = BO - AB$. After the full simplification process, we end up with Equation (2) again.

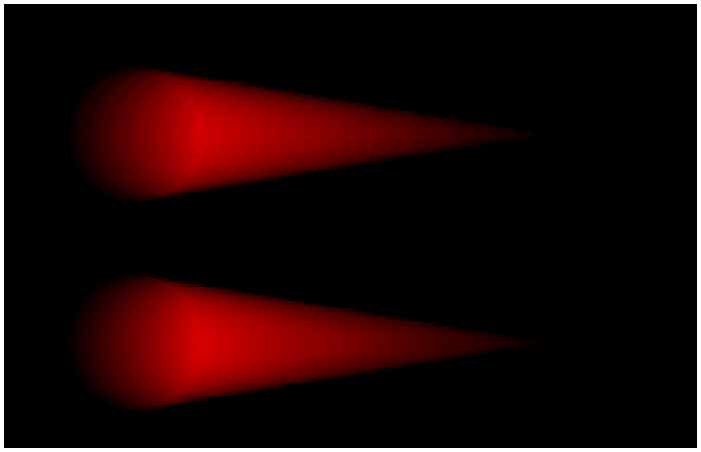


Figure 14. Top: Discrete method (150 quads); Bottom: Our method (Single quad, no overdraw), hardness=1.0, flow=0.02, opacity=1.0, tapering to zero.

If you solve the quadratic above, the two obtained values of r^2 give us the radii of the two circles we want. Take a square root, and we're done:

$$r_1^2 = \frac{(2\lambda + \epsilon) + \sqrt{(-2\lambda - \epsilon)^2 - 4(\lambda^2 + \epsilon BE^2)}}{2}$$
$$r_2^2 = \frac{(2\lambda + \epsilon) - \sqrt{(-2\lambda - \epsilon)^2 - 4(\lambda^2 + \epsilon BE^2)}}{2}$$

Once we get the two radii, we can get A_1 and A_2 using the Pythagorean theorem. See Figure 14 for a comparison.

Appendix D - Falloff

Until now, we were only talking about stamps with a distant-invariant function, but brush tools also offer a distance-based falloff using a parameter called hardness.

To compute falloff for any given stamp, we first compute $\phi(x, X, Y)$, the normalized distance from the center. Here, X, Y are the coordinates of the given point E, and $(x, 0.5)$ represents the coordinates of the sliding center; ϕ is 0 at the center, and it is 1 at the circumference of the stamp:

$$\phi(x, X, Y) = \frac{\text{dist}(x, X, Y)}{\text{radius}(x)}.$$

The distance function gives the Euclidean distance from any point $E(X, Y)$ to any circle center $(x, 0.5)$:

$$\text{dist}(\text{x}, \text{X}, \text{Y}) = \sqrt{(\text{X} - \text{x})^2 + (\text{Y} - 0.5)^2}.$$

The radius function gives the radius of any circle centered at $(x, 0.5)$:

$$\text{radius}(\text{x}) = \text{lerp}\left(0.5, \frac{\text{R}_{\text{min}}}{2\text{R}_{\text{max}}}, \psi(\text{x})\right).$$

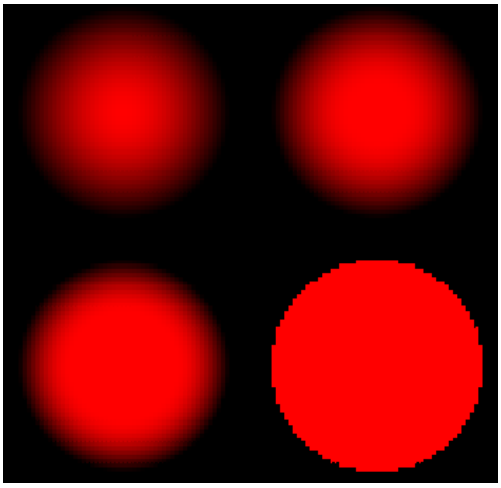


Figure 15. Stamps for brushes with 0, 0.25, 0.50, and 1.0 hardness.

$R_{\min}, R_{\max}$ are the minimum and the maximum radii of the stroke in pixels, and N is the length of the stroke. The term $\psi(x)$ represents the interpolation factor. It is 0 at the center of the left-most circle of the stroke and 1 at the center of the right-most circle of the stroke:

$$\psi(x) = \frac{2R_{\max}(x - 0.5)}{N}.$$

Now that we have obtained a normalized distance ϕ in the range $[0, 1]$ (for affected points), we can calculate the falloff. In this implementation, we use a squared-cosine falloff, but any continuous function can be used instead.

Thus, given a hardness in the range $[0, 1]$,

$$\text{falloff}(x, X, Y) = \begin{cases} 1 & \text{if } \phi(x, X, Y) < \text{hardness} \\ \cos\left(\frac{\pi\phi(x,X,Y)}{2(1-\text{hardness})} + \text{phase}\right)^2, & \text{otherwise;} \end{cases}$$

$$\text{phase} = \frac{\pi}{2}\left(1 - \frac{1}{1 - \text{hardness}}\right).$$

To find the alpha at any point, we need to integrate over all of the falloff values of the circle-centers that touch that point:

$$\alpha(X, Y) = \min\left(\text{opacity}, F \times 2R_{\max} \int_{X_1}^{X_2} \text{falloff}(x, X, Y).dx\right).$$

We could not find a function that acted as a falloff function that we could also analytically integrate over. Thus, we'll be using numerical integration methods such as Simpson's rule or the trapezoidal rule. There is potential for optimization here, if we can find a better way to numerically integrate, requiring fewer samples, or a falloff function that is more amenable to integration.

Appendix E - Interpolating Parameters

When dealing with variable parameters, we have to linearly interpolate between them. The parameters $R_{\max}$, H_1, and F_1 are associated with the left-most circle of our brush stroke, which is centered at the normalized coordinates $(0.5, 0.5)$. The parameters $R_{\min}$, H_2, and F_2 are associated with the right-most circle of our brush stroke, which is centered at the normalized coordinates $(\frac{R_{\max}+N}{2R_{\max}}, 0.5)$.

We want a function $\psi(x)$ that ranges linearly between [0,1] between the two end-points of our stroke, given a normalized coordinate x. We use this function to generate our linear interpolation coefficient for any given pixel:

$$\psi(x) = \frac{R_{\max}(2x-1)}{N}.$$

We can use this function to compute a parameter for a circle centered at x. For instance, hardness for a circle with center $(x, 0.5)$ is

$$\text{hardness}(x) = \text{lerp}(H_1, H_2, \psi(x)).$$

Because flow, radius and hardness are interpolated within limits X_1 and X_2, this guarantees that ψ will be in the range $[0, 1]$.

References

CHAN, E., AND DURAND, F. 2005. Fast prefiltered lines. *GPU Gems 2*. URL: https://developer.nvidia.com/gpugems/GPUGems2/gpugems2_chapter22.html. 2

DIVERDI, S., 2013. A brush stroke synthesis toolbox, Jan. 2

PORTER, T., AND DUFF, T. 1984. Compositing digital images. *SIGGRAPH Comput. Graph. 18*, 3 (Jan.), 253–259. URL: http://doi.acm.org/10.1145/964965.808606, doi:10.1145/964965.808606. 5

VANDOREN, P., CLAESEN, L., VAN LAERHOVEN, T., TAELMAN, J., RAYMAEKERS, C., FLERACKERS, E., AND VAN REETH, F., 2009. Fluidpaint: An interactive digital painting system using real wet brushes. URL: http://doi.acm.org/10.1145/1731903.1731914, doi:10.1145/1731903.1731914. 2

XU, S., TANG, M., LAU, F. C. M., AND PAN, Y. 2004. Virtual hairy brush for painterly rendering. *Graph. Models 66*, 5 (Sept.), 263–302. doi:10.1016/j.gmod.2004.05.006. 2

Author Contact Information

Apoorva Joshi
apoorvaj@apoorvaj.io

Apoorva Joshi, Efficient Rendering of Linear Brush Strokes, *Journal of Computer Graphics Techniques (JCGT)*, vol. 7, no. 1, 1–17, 2018
http://jcgt.org/published/0007/01/01/

Received:	2017-11-07		
Recommended:	2018-01-08	Corresponding Editor:	Oliver Wang
Published:	2018-02-14	Editor-in-Chief:	Marc Olano