Towards Automatic Band-Limited Procedural Shaders

Jonathan Dorn, Connelly Barnes, Jason Lawrence, Westley Weimer
University of Virginia
7 October 2015
Rendering Textures

\[ \int \cos(s)k(s, w) \, ds \]
Rendering Textures

\[ \int \cos(s) k(s, w) \, ds \]

Texture function
Rendering Textures

\[ \int \cos(s) k(s, w) \, ds \]

Pixel footprint
Rendering Textures

\[ \int \cos(s) k(s, w) \, ds \]
Single-Sample Approximation

\[ \cos(s)k(s, w) \]
Multi-Sample Approximation

\[ \frac{1}{n} \sum_{i=1}^{n} \cos(s_i) k(s_i, w) \]
Band-Limited Functions

\[ \hat{\cos_k}(s, w) \]
Band-Limited Procedural Shaders

Given a procedural shader, generate a new shader that is:

- Visually faithful to original,
- A band-limited function of sampling rate,
- Efficient to compute.
Band-Limited Primitives

1. Solve by hand.
   - See paper and supplemental material for examples.

2. Published solutions.
   - E.g. Gabor noise, summed area tables.

<table>
<thead>
<tr>
<th>$f(x)$</th>
<th>$\hat{f}(x,w)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>$x$</td>
</tr>
<tr>
<td>$x^2$</td>
<td>$x^2 + w^2$</td>
</tr>
<tr>
<td>$\text{fract}_1(x)$</td>
<td>$\frac{1}{2} - \sum_{n=1}^{\infty} \frac{\sin(2\pi nx)}{\pi n} e^{-2w^2\pi^2 n^2}$</td>
</tr>
<tr>
<td>$\text{fract}_2(x)$</td>
<td>$\frac{1}{2w^2} \left( \frac{\text{fract}^2 \left( x + \frac{w}{2} \right)}{} + \left[ x + \frac{w}{2} \right] - \text{fract}^2 \left( x - \frac{w}{2} \right) - \left[ x - \frac{w}{2} \right] \right)$</td>
</tr>
<tr>
<td>$\text{fract}_3(x)$</td>
<td>$\frac{1}{12w^2} \left( f'(x - w) + f'(x + w) - 2f'(x) \right)$</td>
</tr>
<tr>
<td>$</td>
<td>x</td>
</tr>
<tr>
<td>$</td>
<td>x</td>
</tr>
<tr>
<td>$</td>
<td>x</td>
</tr>
<tr>
<td>$\cos x$</td>
<td>$\cos x e^{-\frac{x^2}{2}}$</td>
</tr>
<tr>
<td>$\text{saturate}(x)$</td>
<td>$\frac{1}{2} \left( \text{erf} \left( \frac{x}{w\sqrt{2}} \right) - (x - 1) \text{erf} \left( \frac{x - 1}{w\sqrt{2}} \right) + w\sqrt{\frac{2}{\pi}} \left( e^{-\frac{x^2}{2w^2}} - e^{-\frac{(x-1)^2}{2w^2}} \right) + 1 \right)$</td>
</tr>
<tr>
<td>$\sin x$</td>
<td>$\sin x e^{-\frac{x^2}{2}}$</td>
</tr>
<tr>
<td>$\text{step}(a,x)$</td>
<td>$\frac{1}{2} \left( 1 + \text{erf} \left( \frac{x - a}{w\sqrt{2}} \right) \right)$</td>
</tr>
<tr>
<td>$\text{ trunc}(x)$</td>
<td>$\text{floor}(x,w) - \text{step}(x,w) + 1$</td>
</tr>
</tbody>
</table>
Linear Combinations

\[ [s + 0.5] - [s] \]
Linear Combinations

\[ [s + 0.5] - [s] \]

**Linearity of integration:**
Can band-limit each term separately
Linear Combinations

\[ \lfloor s + 0.5 \rfloor - \lfloor s \rfloor \]
Linear Combinations

\[
\widehat{\text{floor}}_k(s + 0.5, w) - \widehat{\text{floor}}_k(s, w)
\]
color stripe(float2 p) {
    return (color)(floor(p.s + 0.5) – floor(p.s));
}
color stripe(float2 p) {
    return (color)(floor(p.s + 0.5) - floor(p.s));
}

Identity function:
Identity transformation
color stripe(float2 p) {
    return (color)(floor(p.s + 0.5) - floor(p.s));
}

Linearity: Identity transformation
color stripe(float2 p) {
    return (color)(floor(p.s + 0.5) - floor(p.s));
}
Abstract Syntax Trees (ASTs)

- Transform AST nodes locally.
- Replace as child of parent.
- Replace as parent of children.

![Diagram showing AST transformation]

- floor
- floor_bl
- W.S
color stripe_bl(float2 p, float2 w) {
    return (color)(floor_bl(p.s + 0.5, w.s) - floor_bl(p.s, w.s));
}
If shader and pixel footprint are multiplicatively separable, the separate parts can be band-limited separately.

We ensure separable pixel footprint with a bounding box approximation.
Multiplicative Combinations

\[(\lfloor s + 0.5 \rfloor - \lfloor s \rfloor)(\lfloor t + 0.5 \rfloor - \lfloor t \rfloor)\]
Multiplicative Combinations

\[
\left( \hat{\text{floor}}_k(s + 0.5, w_s) - \hat{\text{floor}}_k(s, w_s) \right) \left( \hat{\text{floor}}_k(t + 0.5, w_t) - \hat{\text{floor}}_k(t, w_t) \right)
\]
Function Composition

\[ \int f(g(x))k(x, w) \, dx = \hat{f} \circ g(x, w) \]

- Our approach: transform subsets of functions.
  - E.g., \( \hat{f}_k(g(x), w) \)
  - Genetic search.

- Our approach: linear combination of sampling rates.
  - \( w = c_0 + c_1w_x + c_2w_y \)
  - Simplex search.
Approach Overview

1. Identify AST nodes that may be transformed.

2. Select subset with genetic search.

3. Fit sampling rate coefficients with simplex search.

4. Replace selected nodes using fitted sampling rates.
Results: Checkerboard

<table>
<thead>
<tr>
<th>Target Image</th>
<th>No Antialiasing</th>
<th>16x Multisampling</th>
<th>Our Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Target Image" /></td>
<td><img src="image2.png" alt="No Antialiasing" /></td>
<td><img src="image3.png" alt="16x Multisampling" /></td>
<td><img src="image4.png" alt="Our Approach" /></td>
</tr>
</tbody>
</table>
Results: Checkerboard

Target Image  
No Antialiasing  
16x Multisampling  
Our Approach

Error heatmap
$L^2$ in RGB
Results: Checkerboard

- **4x faster** than multisampling.
- **2x less \( L^2 \) (RGB) error** than multisampling.
  - Error is due to separable kernel approximation.
### Results: Brick and Wood

<table>
<thead>
<tr>
<th>Target Image</th>
<th>No Antialiasing</th>
<th>16x Multisampling</th>
<th>Our Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="brick.png" alt="Brick Image" /></td>
<td><img src="bricks_no_antialiasing.png" alt="Brick Image" /></td>
<td><img src="bricks_16x_multisampling.png" alt="Brick Image" /></td>
<td><img src="bricks_our_approach.png" alt="Brick Image" /></td>
</tr>
</tbody>
</table>

6x faster, 2x less $L^2$ error than multisampling.

<table>
<thead>
<tr>
<th>Wood Image</th>
<th>No Antialiasing</th>
<th>16x Multisampling</th>
<th>Our Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="wood.png" alt="Wood Image" /></td>
<td><img src="wood_no_antialiasing.png" alt="Wood Image" /></td>
<td><img src="wood_16x_multisampling.png" alt="Wood Image" /></td>
<td><img src="wood_our_approach.png" alt="Wood Image" /></td>
</tr>
</tbody>
</table>

5x faster, 3x more $L^2$ error than multisampling.
Results: Procedural Noise

- Target Image
- No Antialiasing
- 16x Multisampling
- Our Approach

6x faster, equivalent $L^2$ error to multisampling.

6x faster, equivalent $L^2$ error to multisampling.
Results: More Complex Procedures

Target Image

No Antialiasing

16x Multisampling

Our Approach

20x faster, 2x more $L^2$ error than multisampling.

17x faster, 2x more $L^2$ error than multisampling.
Future Work

- More sophisticated fitness functions.
- Spatially varying loop bounds.
- Merging conditional branches.
- Investigate alternative transformations.
- Language support.
Conclusion

- Explore problem of automatically band-limiting general texture shaders.

- Band-limited shaders are faster than multisampling.

- Work remains to scale up.
Questions?