Eric Lengyel, Ph.D. Terathon Software

#### About the speaker

Working in game/graphics dev since 1994

i3D 2018

Montréal

• Previously at Sierra, Apple, Naughty Dog

- Current projects:
  - Slug Library, C4 Engine, The 31st, FGED



#### i3D 2018 Montréal

#### About this talk

 Technical details about the "Slug" font rendering algorithm

Paper published in JCGT in June 2017

New developments in past year

### Font Rendering Ubiquity

- Text rendered everywhere in 3D applications
  - GUI: Buttons, checkboxes, lists, menus, ...
  - Games: Score, health, ammo, ...
  - In scene: Signs, labels, computer screens, ...
  - Debug info: Console, stats, timings, ...



# Font Rendering Design Goals

i3D 2018

- Unified approach
  - Same technique used to render all text in all situations
- Runs in shader on GPU
  - Fully dynamic, but also allows caching
  - Can be combined with other materials

# Font Rendering Design Goals

- Total resolution independence
  - No precomputed images or distance fields

i3D 2018

- Ability to render with any transform
  - Arbitrary scale, rotation, perspective
- Must look good at large and small font sizes

# Font Rendering Design Goals

i3D 2018

- Minimal triangulation
  - Don't want lots of small triangles
  - GPUs perform best with large triangles
  - A fixed per-glyph vertex count is desirable
  - Would like to be able to easily clip text
  - Would like to apply text to curved surfaces

#### i3D 2018 Montréal

### **Rendering Algorithm Priorities**

1. Works correctly

2. Looks good

3. Runs fast

# Glyphs in TrueType

Glyph defined by one or more closed contours

 Each contour composed of continuous sequence of quadratic Bézier curves

Each Bézier curve has three control points



#### i3D 2018 Montréal

# Glyph Space

Glyphs are defined on em square

• Coordinates in range [0,1] inside em square

Curves can extend outside em square

i3D 2018 Montréal

#### **Glyph Space**



 To determine whether point inside glyph, calculate winding number with respect to each contour and sum

 Point inside glyph outline if sum of winding numbers is nonzero

 The winding number is the count of complete loops a contour makes around a point

 One direction (arbitrary, either CW or CCW) is considered positive, and opposite direction is then considered negative

- To calculate winding number, fire a ray from point being rendered to infinity
- Direction doesn't matter, so use +x direction for convenience
- Look for contour intersections along the ray

- When contour crosses ray from left to right, increment winding number
- When contour crosses ray from right to left, decrement winding number
- Or other way around, as long as consistent



#### Quadratic Bézier Curve

• Three control points **p**<sub>1</sub>, **p**<sub>2</sub>, **p**<sub>3</sub>

• Parametric curve with  $0 \le t \le 1$ :

$$\mathbf{C}(t) = (1-t)^{2} \mathbf{p}_{1} + 2t (1-t) \mathbf{p}_{2} + t^{2} \mathbf{p}_{3}$$

### **Ray-Curve Intersection**

• Translate control points so ray origin is (0,0)

• Assume ray direction is +x axis

 Solve for values of t where y coordinate of Bézier curve is zero

#### **Ray-Curve Intersection**

• Let 
$$\mathbf{p}_i = (x_i, y_i)$$

Ray intersects curve at roots of polynomial

$$(y_1 - 2y_2 + y_3)t^2 - 2(y_1 - y_2)t + y_1$$
  
 $a = y_1 - 2y_2 + y_3$   $b = y_1 - y_2$   $c = y_1$ 

• Roots  $t_1$  and  $t_2$  given by

$$t_1 = \frac{b - \sqrt{b^2 - ac}}{a} \qquad t_2 = \frac{b + \sqrt{b^2 - ac}}{a}$$

• If a near zero, use root of linear polynomial:

$$t_1 = t_2 = \frac{c}{2b}$$

### **Ray-Curve Intersection**

- Valid intersection at t<sub>i</sub> when:
  - $0 \le t_i \le 1$  (between curve endpoints)
  - $C_x(t_i) \ge 0$  (at positive distance along ray)

- $t_i = 1$  specifically disallowed
  - Corresponds to intersection at t<sub>i</sub> = 0 on next Bézier curve, and don't want to count twice

#### **Ray-Curve Intersection** V $\mathbf{p}_1$ *t* = 0 (0,0)Х t = 10 **p**<sub>2</sub> **p**<sub>3</sub>

# **Ray-Curve Intersection**

- Increment or decrement winding number?
- Look at y values in range  $0 \le t_i \le 1$ 
  - Positive before  $t_i$  or negative after  $t_i$ : increment
  - Negative before  $t_i$  or positive after  $t_i$ : decrement
- Can't rely on derivative
  - Zero if ray tangent to curve

#### Robustness

Sound from purely mathematical standpoint

But plagued by numerical precision errors!

• Floating-point limits cause huge problems for roots near endpoints where  $t_i = 0$  or  $t_i = 1$ 

#### i3D 2018 Montréal

#### Numerical Precision Errors

Produce sparkle and streak artifacts

 Hacks like epsilons and coordinate perturbation just shift problem cases around

Need something that's 100% robust

# Slug Algorithm

- Calculates winding number
  - Input is arbitrary set of closed contours composed of quadratic Bézier curves

- Performs antialiasing
  - Determines fractional coverage at each pixel

i3D 2018

## Priority #1: Works Correctly

i3D 2018

- Robust for all valid inputs
  - Meaning any floating-point coordinates that are not infinity or NaN
- No distortion of glyph outlines
- No sparkle artifacts

### Equivalence Class Algorithm

i3D 2018

- Infinite problem space reduced to a finite number of equivalence classes
- Same procedure followed for all cases in each equivalence class
- Same abstract idea as Marching Cubes

i3D 2018

Montréal

#### Bézier Curve Classification

 Look at y coordinates of the three control points

Each positive, negative, or zero

27 classes based on these states

#### Bézier Curve Classification

• It turns out we can do better than 27 classes

 Classify each control point based on whether y coordinate is nonnegative or negative

• Only 8 equivalence classes

#### i3D 2018 Montréal

#### Bézier Curve Classification

- Roots (ray intersections) always occur in same way for all cases in each class
- We care about places where curve transitions between nonnegative and negative
- Only have to decide how to modify winding number for each root

### Winding Number Modification

Consider derivative of y coordinate:

$$y'(t) = 2at - 2b$$

$$a = y_1 - 2y_2 + y_3$$
  $b = y_1 - y_2$ 

#### i3D 2018 Montréal

### Winding Number Modification

 An observation about the roots for nonzero discriminant D:

$$t_1 = \frac{b - \sqrt{D}}{a} \qquad t_2 = \frac{b + \sqrt{D}}{a} \qquad D = b^2 - ac$$
$$y'(t_1) = -2\sqrt{D} \qquad y'(t_2) = +2\sqrt{D}$$

#### i3D 2018 Montréal

# Winding Number Modification

- Root at  $t_1$  always crosses ray from left to right
  - Going from nonnegative to negative
  - Always increment winding number

- Root at  $t_2$  always crosses ray from right to left
  - Going from negative to nonnegative
  - Always decrement winding number
# Winding Number Modification

- We can also incorporate cases where ray intersects an endpoint tangentially
- Winding number modified only when transition between nonnegative and negative occurs

i3D 2018

Montréal

• x coordinate at transition must be positive

### **GPU-Centered Font Rendering Directly from Glyph Outlines**





#### i3D 2018 Montréal

## Class A: All Nonnegative

Nothing happens to winding number



# Class H: All Negative

Nothing happens to winding number



### Classes B and D: One Transition

• Winding number decremented if  $x(t_2) > 0$ 



#### i3D 2018 Montréal

### Classes E and G: One Transition

• Winding number incremented if  $x(t_1) > 0$ 



### Class C: Two Transitions

• Winding number incremented if  $x(t_1) > 0$ 

i3D 2018

Montréal

• Winding number decremented if  $x(t_2) > 0$ 



## Class F: Two Transitions

• Winding number incremented if  $x(t_1) > 0$ 

i3D 2018

Montréal

• Winding number decremented if  $x(t_2) > 0$ 



# Discriminant Clamping

- In classes C and F, we could have a negative discriminant D
- To handle with uniformity, clamp D to zero

i3D 2018

Montréal

 Always two transitions at same x coordinate, so guaranteed to cancel each other out

### **Root Calculation**

```
float2 SolvePoly(float4 p12, float2 p3)
Ł
    float2 a = p12.xy - p12.zw * 2.0 + p3; // Calculate coefficients.
    float2 b = p12.xy - p12.zw;
    float ra = 1.0 / a.y;
    float rb = 0.5 / b.y;
    float d = sqrt(max(b.y * b.y - a.y * p12.y, 0.0)); // Clamp discriminant to zero.
    float t1 = (b.y - d) * ra;
    float t2 = (b.y + d) * ra;
    if (abs(a.y) < epsilon) t1 = t2 = p12.y * rb; // Handle linear case where |a| \approx 0.
    // Return x coordinates at t1 and t2.
    return (float2((a.x * t1 - b.x * 2.0) * t1 + p12.x,
                   (a.x * t2 - b.x * 2.0) * t2 + p12.x));
}
```

#### i3D 2018 Montréal

# Root Eligibility

We know what to do for each root

Just need to decide whether to actually do it!

Use a lookup table for root eligibility

#### i3D 2018 Montréal

# Root Eligibility

- Nonnegative/negative gives us 3-bit state
  - Just use sign bits of y coordinates

• Look up 2-bit root eligibilities for  $t_1$  and  $t_2$ 

Total LUT size is a tiny 16 bits

## Root Eligibility Lookup Table

Class	<i>y</i> <sub>3</sub> < 0	<i>y</i> <sub>2</sub> < 0	<i>y</i> <sub>1</sub> < 0	Root 2	Root 1
А	0	0	0	0	0
В	0	0	1	1	0
С	0	1	0	1	1
D	0	1	1	1	0
Е	1	0	0	0	1
F	1	0	1	1	1
G	1	1	0	0	1
Н	1	1	1	0	0
				0x2E	0x74

## Calculating Root Codes

```
uint CalcRootCode(float y1, float y2, float y3)
{
    uint i1 = asuint(y1) >> 31U;
    uint i2 = asuint(y2) >> 30U;
    uint i3 = asuint(y3) >> 29U;
    uint shift = (i2 \& 2U) | (i1 \& \sim 2U);
    shift = (i3 & 4U) | (shift & ~4U);
    return ((0x2E74U >> shift) & 0x0101U);
}
```

```
bool TestCurve(uint code)
   return (code != 0U);
}
bool TestRoot1(uint code)
   return ((code & 1U) != 0U);
}
bool TestRoot2(uint code)
   return (code > 1U);
}
```

### Ternary Logic Instruction

- Recent Nvidia GPUs have a ternary logic instruction (LOP3)
- Maps arbitrary 3-bit input to 1-bit output with 8-bit lookup table encoded in the instruction

i3D 2018

Montréal

Perfect fit for our algorithm!

### Ternary Logic Instruction

- Instruction not directly accessible from pixel shader code
- Compiler can recognize arbitrary sequence of AND, OR, XOR, NOT operations and generate single ternary instruction

i3D 2018

Montréal

#### i3D 2018 Montréal

## **Ternary Logic Instruction**

 With LOP3, root code calculation requires only 2 instructions where otherwise need at least 7

• Shader gets about 4% faster in all cases

### Root Codes with Ternary Logic

```
int2 CalcRootCode(float y1, float y2, float y3)
{
   int a = asint(y1);
   int b = asint(y2);
    int c = asint(y3);
   return (int2(~a & (b | c) | (~b & c),
                 a & (~b | ~c) | (b & ~c));
```

```
bool TestCurve(int2 code)
{
    return ((code.x | code.y) < 0);
}
bool TestDest1(ist2 code)</pre>
```

```
bool TestRoot1(int2 code)
{
    return (code.x < 0);
}</pre>
```

```
bool TestRoot2(int2 code)
{
    return (code.y < 0);
}</pre>
```

#### i3D 2018 Montréal

# **Total Winding Number**

```
int winding = 0;
for (all Bézier curves)
{
    float4 p12 = first (.xy) and second (.zw) control points
    float2 p3 = third (.xy) control point
    code = CalcRootCode(p12.y, p12.w, p3.y);
    if (TestCurve(code))
    {
        float2 r = SolvePoly(p12, p3);
        if ((TestRoot1(code)) && (r.x > 0.0)) winding += 1;
        if ((TestRoot2(code)) && (r.y > 0.0)) winding -= 1;
    }
}
```

if (winding != 0) then pixel is inside glyph outline

# Priority #2: Looks Good

Performs accurate antialiasing

Handles arbitrary transforms well

Handles minification well

 Integer winding number produces simple in/out state for each pixel

Correct, but has jagged edges everywhere

• We need fractional pixel coverage values

### **GPU-Centered Font Rendering Directly from Glyph Outlines**



- Ray origin is at pixel center
- Previously, we incremented or decremented winding number when  $x(t_i) > 0$
- Now, we add or subtract the fractional distance ray makes it through pixel before intersection

- Let *u* be number of pixels per em
  - Scales coordinates so that width of pixel = 1 unit

Always change winding number (WN) by

saturate  $\left(u \cdot x(t_i) + \frac{1}{2}\right)$ 

- If  $u \cdot x(t_i) \leq -0.5$ , then no change to WN
- If  $u \cdot x(t_i) \ge 0.5$ , then WN always changed by 1
- In between, WN changed by fractional value
- Accounts for multiple curves per pixel

### **GPU-Centered Font Rendering Directly from Glyph Outlines**





## Fractional Winding Number

```
float coverage = 0.0;
for (all Bézier curves)
{
    float4 p12 = first (.xy) and second (.zw) control points
    float2 p3 = third (.xy) control point
    code = CalcRootCode(p12.y, p12.w, p3.y);
    if (TestCurve(code))
    {
        float2 r = SolvePoly(p12, p3) * pixelsPerEm;
        if (TestRoot1(code)) coverage += saturate(r.x + 0.5);
        if (TestRoot2(code)) coverage -= saturate(r.y + 0.5);
    }
```

# Antialiasing

- This gives us excellent 1D antialiasing
  - Looks great when curves are mostly vertical
- Doesn't work well for mostly horizontal curves
- So fire a ray in the y direction, too
  - Looks great when curves are mostly horizontal

# Antialiasing

- Calculate coverage for two rays at each pixel
  - One in x direction, and one in y direction
  - Note pixels per em could be different in x and y

- Combine two coverage values for good
   2D antialiasing
  - Lots of ways to calculate weighted average

# Antialiasing

Output is linear coverage value

 Works best when blended into sRGB framebuffer

## Bounding Box Dilation

- Draw one quad per glyph coinciding with glyph's bounding box
- GPU fills pixels with centers covered by quad

i3D 2018

Montréal

 Could miss pixels on boundary with up to 50% fractional coverage value

### **GPU-Centered Font Rendering Directly from Glyph Outlines**





# Bounding Box Dilation

Must dilate bounding box by half pixel width

• In em space, dilate by 0.5 / font size

If size dynamically change, need to estimate smallest on-screen pixels per em

## Minification

 At very small font sizes, lots of detail can occur inside each pixel

 Can't be captured by single sample position at pixel center

# Adaptive Supersampling

 As pixels get larger in em space, increase number of samples

 Use screen space derivatives to dynamically calculate sample counts for horizontal and vertical rays

### **GPU-Centered Font Rendering Directly from Glyph Outlines**




## Adaptive Supersampling

- Example sample count calculation
  - 1–4 samples per pixel in each direction

```
float2 emsPerPixel = fwidth(renderCoord);
int2 sampleCount = clamp(int2(emsPerPixel * 32.0 + 1.0), int2(1, 1), int2(4, 4));
```

### Adaptive Supersampling

#### Supersampling

#### Single sample

TOLE TOPONE OF ALL LESSON, AND FOR TO BE AND THE REAL PARTY OF TOTAL ADDRESS OF TOTAL ADDRESS OF TOTAL ADDRESS TOLE TO ADDRESS ADDRESS OF ADDRESS OF ADDRESS OF ADDRESS OF TOTAL ADDRESS OF TOTAL ADDRESS OF TOTAL ADDRESS OF ADDRESS O ামনামান কলেছেনে, এই উপানে, মান সকলেজেনে, মান্য মন্দ্র মন্ত্র সময়ে সময়ে হিলেজ বেজেলে বেজেলে সেনেনে সময়ে। প্রথা বৌধা যে মিনুর হিলেজ বিজে সিমারে উপানে সময়ে বিজেলেজেনে সেনেনে সেনেনে সেনেনে বেজেলে বেজেলে সেনেনে সেনেনে স মান্য মন্দ্র সময়ে সেনে সেনে সেনে সেনে সেনে সেনেনে সেনেনে সেনেনে সেনেনে সেনেনে সেনে বেজেলে বেজেলে সেনেনে সেনেনে የሚገኘ መምግርም ጫገኙነኝ ምም ገኘ መገኘውም 11 የምም 31 የተመውለም በአዋርቱ ምርጭ ምርጫ ውለጥበር በ4 ውስ ላይታን በግለሥር በ74/13/49 ውሸ ዋሪ ሸነት በሚታወሰ ሰው ት ነርስት 13 በባትር በገለዋ ከነበረት ውስት የሆኑ የሚሰሩት ነው ለመሰብ ብዙ ለውስት ለውስት የሆኑ በተመለከት በቀን በመሰብ ለውስት ለውስት የሆኑ የሰላት ለመሰረት የመሰረት የመሰረት የተመሰረት በለውስት በቀንም በአስታውረም ለግጭ በሰብ ለውስት ለውስት የሆኑ የመሰረት በውስት የሆኑ የሆኑ የሆኑ የሆኑ የሆኑ የሆኑ የሆኑ የ ያውነ ሃርሃ የተምራተዋና ላው ፖለት መንስቲካት ላይ የሚሰው እና የቀውረውንር, የአቶ ና ላ ነውን ከውሊ የርቁ ነው ይጠቅም ነው ነው. የርካ ላው ያውው ያውርት ውጤ አውርት ው በመቼ ያውርኛ ምርብ ይገለዋና ምር የድ ውርጅውና ላይ እና በውስት የርቅ ምር ውርጅውና ነው. የድ ውርጅውና ነው የርቅ ብርር ውስጥ ነ ሃርካ ላው ያውው ሃርስ የ ውጤ አውርት ው ምር ለአውር የ የአመር የ አውር ምር ማር የምር በውርጅውና ለምር ምር ላይ የምር ብር ስለ የ A fee a few recends, year room in the ground is to group a subdry. It's minute debest and experie descepts a president for device year. You can be done works without means transmitted and of the new hor. In the law is more balance ગાળ કે દાંગ માંગણકાં, ગુમાં માંગ માંગણ ઉપયોગ છે છે છે. આ ગામ કે પ્રાથમિક કરે પ્રાથમિક છે છે છે છે. આ ગામ પ્રાથમ આ છે છે છે છે છે. દુધ્ધ છે છે. આ ગામ પ્રાથમિક છે છે છે છે છે છે. આ ગામ પ્રાથમિક આ ગામ આ આ ગામ આ છે છે. આ ગામ આ આ છે છે છે છે છે છે છે. આ ગામ પ્રાથમિક અને છે છે છે. ઉપયોગ પ્રાથમિક આ ગામ પ્રાથમિક NR JAG Sensel op 64 der provinsiel takte keit (minist hafte kallen in der Verligt och park had der soner Ander Kreis in stellt genetike, her deren all einer keit pake kontektiver ander der verke belanden ander soner Ander der Verligt der Sense Research einer der Verligt och kontektiver ander der verke bester ander soner konst સ્ટાઈન સ્ટાઈન છે. આઈનું વીકાર્ય સાથ, ઉભ છે. આઈ સાથ સ્ટાઈન સ્ટાઈન સ્ટાઈન્સ પ્રાથમિક પ્રાથમિક પ્રાથમિક પ્રાથમિક પ સ્ટાઈને (એ Auri) અને અંગ અને પ્રાથમિક સ્ટાથમિક સ્ટાથમિક સ્ટાઈન સ્ટાઈને સ્ટાઈને સ્ટાઈને સ્ટાઈને સ્ટાઈને સ્ટાઈને સ સ્ટાઈને આઈને અને અને સ્ટાયમાં સ્ટાઈને સ અર્થમાં છે. આવી કેમ્ટ્રેન્ડ્રેસ્ટ્રેસ્ટ્રેસ્ટ્રેસ્ટ્રેસ્ટ્રેસ્ટ્રેસ્ટ્રેસ્ટ્રેસ્ટ્રેસ્ટ્રેસ્ટ્રેસ્ટ્રેસ્ટ્રેસ્ટ્રેસ્ટ્રેસ્ટ્રેસ્ટ્રેસ્ટ્રેસ્ટ્રેસ્ટ્રેસ્ટ્રેસ્ટ્રેસ્ટ્રેસ્ટ્રેસ્ટ્રેસ્ટ્રેસ્ટ્રેસ્ટ્રેસ્ટ્રેસ્ટ્રેસ્ટ્રેસ્ટ્રેસ્ટ્રેસ્ટ્રેસ્ટ્રેસ્ટ્રેસ્ટ્રેસ્ટ્રેસ્ટ્રેસ્ટ્રિટ્ટ્રેસ્ટ્રિટ્ટ્ સ્ટ્રેસ્ટ્રિસ્ટ્ર્સ્ટ્રેસ્ટ્રિસ્ટ્રેસ્ટ્રિસ્ટ્રિસ્ટ્રિસ્ટ્રિસ્ટ્રિસ્ટ્રિસ્ટ્રિસ્ટ્રિસ્ટ્રિસ્ટ્રિસ્ટ્રિસ્ટ્રિસ્ટ્

Your same in Que Ontents, and you've a findiplier in the small loan of States Valey, where the largest employer is the small loan of States Valey, where the same data was a second and a state where the same data was a second TORT NAME IN GUILS COMMENT, UND FORM IS DESIGNATE IN THE START DURING OF AND THE START DURING PROPERTY IN THE MEDICAL DISTINCTIONS IN TRANSMENT OF BRIDE IN THE COMPARISON AS DOWNLOW AND ADDRESS OF ADDRESS Your crain come you and the trainers. There's a manager takes at the table, training as we enge at low-. The pre- of the first engine as kinds of tables friends and grap relicity of the incredible size of the first but it the strain base of and built water for tables to manage takes to be built at the tables and tables and tables are a strainer to a ef higt familiarity familing through helian in the tweat-yes our annual walk. As you represen the second for a court bold, the walk and flow of the building colleges in anyone walk of designed a familiar in the building walk and w box, the wait and box's of the burning compare to explore a very comparison character areas as some or some are being public lab a binders light at the position of a ginet multiplic tips. The grand begins to bill benefit your later areas and the burning compares to explore a section of a ginet multiplic tips. sens) presed entri is transmit regit et tra content et a grant messato mito, tra grantet segura si tra transmit poy mat, it positif la storte up tra senguento sispa to sociale, foi it i so bak traine pada tanto de securitaria internation menter anternational and ana traine and an traine anternation also material and an excidence international and Source for a source up and assessed and the source of the Adler a few avconds, you shere to the ground in a proced sector of the wave inter in these to resid below product. for abuse you, You area for few revelue defines proceed you and red out of the wave inter in these to resid below product. After a fere accords, you store to the ground in a printy medicit. Ex maning carries and expan through a point ax for above you, You are the first majore failing toreard you and rel and of the may just in time to read being explore. You will remember a set the remember activate a local remember You to and in You have seen arms. You are used THE REVERT VERY THE ADMENT ADMENT ADMENT ADMENT VALUE ADMENT Yee politymenetive of the present and take a local arrays. You're net in Tener Weby ney news, You'rean news, denn been in energy direction, but there net news taken benefit and a work indicated score land, ou war worked to arrays be been below a below war a work taken. Dennis reak a benefit of work being bet, war worked to be बताना उत्ताव आ त्यांत् विकारांत्रम, पेना फिला सत स्वायत दिवा पुण्डीक शिवांत्रमुं प्राय का भाषाकी उत्ताव्यमा उभय शिवां वरणां का स्वीतांत्री रेत्री शिवां शिक्षेत्रीयार्थाः पित्र इत्यांत्री काम दिखाना श्रेर्वात्रम् प्राय्ते का प्राय्ता का भा स्वार्थिक व्यार्थ के स्वार्थ्यात्र स्वार्थ्य स्वार्थ्य स्वार्थ्य का स्वितंत्र अत्यात्र प्रायं भाषा स्वार्थ्य स्व activity by laced industriants. We portal area clears, loaning only a banch of analytication explore, and the remains of sense takerionate MCE, workers believe. You'rit or your const

## Priority #3: Runs Fast

- Minimize raw computation
  - We want to examine as few Bézier curves as possible in the pixel shader

- Promote high GPU resource utilization
  - We want low thread divergence in the pixel shader

#### i3D 2018 Montréal

#### Computation

 Looking for ray intersections with all Bézier curves would be very slow

- Many curves far away from ray and never contribute to coverage (classes A and H)
- Need to reduce active set of curves

 Divide glyph's bounding box into many horizontal and vertical bands





- Bézier curves are sorted into the bands
  - A curve can belong to multiple bands
  - When rendering, band selected based on ray origin
- Doesn't matter how large pixel footprint gets
  - Pixel size only matters in ray direction
  - Band parallel to ray extends forever

- Perfectly horizontal lines are never added to horizontal bands
- Perfectly vertical lines are never added to vertical bands
- Ray intersections with these can't happen

- Further dividing into cells causes problems
  - Pixel could cover multiple cells along ray direction
  - Those cells often won't have disjoint curve sets
  - Can't calculate final winding number without additional per-cell fix-ups that aren't robust
- Bands are a much cleaner and faster solution

- Using large numbers of bands is faster
  - Allows fewer curves per band

- Minimize number of curves in worst band
  - GPU thread coherence makes shader wait for highest number of loop iterations in a group of pixels (32 or 64, hardware dependent)

#### i3D 2018 Montréal

# Banding

 Can merge data for bands containing identical sets of Bézier curves



## Curve Sorting

 Curves in each horizontal band are sorted in descending order by the maximum
 x coordinate of the three control points

 This is an early-exit optimization that makes the shader about twice as fast compared to not sorting curves at all

## Curve Sorting

 Translate control points so that pixel center is at (0,0), and perform test:

if (max(max(p12.x, p12.z), p3.x) \* pixelsPerEm.x < -0.5) break;</pre>

- If true, then it's not possible to hit this curve or any that follow
  - Sorted in descending order, so must also be true for all later curves in the band

### Symmetric Band Optimization

Can do even better at large font sizes

Also sort in ascending order by minimum
 x coordinate of the three control points

 Use a left-pointing ray when pixel x coordinate is less than a band split value

i3D 2018

Montréal

#### Symmetric Band Optimization

i3D 2018

Montréal



## Symmetric Band Optimization

 Sorting and split values also apply to vertical bands

- Splits values introduce divergence in shader
  - Faster for large font sizes where lots of pixels will choose same execution path

i3D 2018

Montréal

Slower for small font sizes due to decoherence

# Bounding Polygons

- Glyphs tend to have empty space near the corners of their bounding boxes
- Clip these corners off to reduce pixels filled

i3D 2018

Montréal

 Adds more triangles, but roughly 10% faster with larger font sizes

### **Bounding Polygons**

i3D 2018

Montréal



# Bounding Polygons

 As with symmetric band splits, bounding polygons increase performance for large font sizes, but can hurt at small font sizes

i3D 2018

Montréal

 Greater number of triangles increase number of pixels double-shaded in 2x2 quads along triangle edges

## **Rectangle Primitives**

 Even when rendering quads, double-shading along the interior edge can be a significant expense at small font sizes

 Expense can be eliminated for text that's aligned to screen axes

## **Rectangle Primitives**

- Most GPUs support rectangle primitives
  - Exposed through GL\_NV\_fill\_rectangle extension
- Specify three vertices, and screen-aligned enclosing rect is drawn without internal edge
- Up to 15% faster for typical font sizes

i3D 2018

Montréal

### Rectangle Primitives

Can be combined with conservative rasterization to handle glyph dilation

 Automatically shades pixels with any amount of coverage

## Multicolor Glyphs

- Microsoft emoji font uses vector artwork
  - Based on same TrueType quadratic Bézier curves

i3D 2018

Montréal

- Multiple layers composited back to front
  - Adds an outer loop to the pixel shader



### Data Stored in Two Texture Maps

i3D 2018

Montréal

- Curve texture
  - 4-channel 16-bit floating-point
  - Stores all Bézier curve control points
- Band texture
  - 4-channel 16-bit integer
  - Stores lists of curves for all bands

#### **Curve Texture**

 Third control point of one curve is always same as first control point of next curve in contour

Red (16 bits)	Green (16 bits)	Blue (16 bits)	Alpha (16 bits)
p1.x (curve 1)	p1.y (curve 1)	p2.x (curve 1)	p2.y (curve 1)
p3.x (curve 1) p1.x (curve 2)	p3.y (curve 1) p1.y (curve 2)	p2.x (curve 2)	p2.y (curve 2)
p3.x (curve 2) p1.x (curve 3)	p3.y (curve 2) p1.y (curve 3)	p2.x (curve 3)	p2.y (curve 3)

## Band Texture

- Each glyph has list of H bands and V bands
- Each band contains list of curves, sorted in both directions



One texel for each curve in each band

#### Results

• 4K display filled with text, timed on NV GeForce 1060

Font	Sample	Complexity	Time (ms)
Arial	ABCDEFG	28	0.70
Minion	ABCDEFG	35	0.71
Times	ABCDEFG	35	0.73
Jokerman	ABCDEFG	60	1.1
Spider	ABORDEF G	500	2.8

i3D 2018 Montréal

### Results

### ABCDEFCHIJK SMNOP 2 PETUVVK YZ ABCDEFGHIJKLMNOPQRSTUVWXYZ ABODEFGHIJELANOPQRSELDDXPZ

ABCDEFGHIJKLMNOPQRSTUVWXYZ

ABQBEF GHIJKLMNOPZRSTVVVXXZ

ABCDEFGHIJKLMNOPQRSTUVWXYZ

i3D 2018 Montréal

#### Results

ብ 👧 🕰 👧 🖉 X 🔝 🥮 💭 🐵 N 🚸 \land 🥠 🗩 🕒 🗨 🍋 🖓 🖓 🖏 - ● 〒 ☆ △ - ♀ ♡ ▲ - ▲ ≫ - ↓ ☆ 🛍 🛢 ※ 🖾 🛪 □ | ● | ● | ▲ - 丶 / / 77 (p 🜙 🗸 (p) 🖉 🍇 🏛 🖄 💞 🌮 🎒 🍲 📥 🐂 💎 🍈 🍆 🍋 🍋 🏷 💫 - A ଟ 🝼 🟉 🚱 🗗 🧬 🔍 📥 🚎 🚔 😁 🔍 👖 😂 💾 🛠 🧏 🗑 🛶 🏠 🏷 i de no se no s / 📰 🛲 🥪 🏇 🎲 🖓 🖉 🗂 🗖 🎧 🐥 📕 🚔 💷 🎬 🍤 🎮 🎯 📖 🕲 🎲 🖑 🖳 🎜 | 🖅 🌃 🌧 🚝 🍜 🛑 🏯 🗄 🏳 📁 🙆 🤣 🏸 🖄 🇐 🌐 🎬 🎬 🎬 🐃 🖘 🖘 - 🗳 🖉 🥶 📶 🐨 🚌 😷 🛠 🐥 👗 ዥ 🐷 🐒 🟠 🐄 🏵 💥 🐺 🐯 💥 🧐 🌰 🌰 🦚 🙆 🛷 🛲 🙈 😂 🎢 🝟 👫 🦨 👙 🍞 🖨 🖾 🛋 👞 👞 📐 👞 🗶 🧐 🕿 🧐 📎 🐼 🙆 🏭 🖬 🖬 🐨 🤮 闷 😳 😳 😳 🌝 🌝 🚱 🚱 🚱 🚱 🚱 🐨 🥵 😳

#### i3D 2018 Montréal

#### Results



i3D 2018

Montréal

### Questions?

lengyel@terathon.com

• Twitter: @EricLengyel