Implicit Surfaces & Independent Research

Matt Keeter March 7, 2025



2025









Embedded & systems software Oxide Computer Company





2013



Electrical engineering Formlabs

A brief timeline 2011 2013

Grad school *MIT Center for Bits & Atoms*









Undergrad Harvey Mudd College







2013





Massively Parallel Rendering of Complex Closed-Form Implicit Surfaces

Matthew J. Keeter, independent researcher ACM Transactions on Graphics (Proceedings of SIGGRAPH), 2020





Graphics research!



1	<pre>let r_outer = 1;</pre>		
2	<pre>let r_inner = 0.35;</pre>		
3			
4	let $r = sqrt(square(x) + square(y));$		
5	<pre>let cross = union(</pre>		
6	sqrt(square(x - r outer) + square(z)) - r inner,		
7	intersection(x - r outer.		
8	intersection(z - r innerr inner - z))):		
9	<pre>let puck = cross.remap xvz(r, v, z):</pre>		
10			
11	let three = 10000.0:		_
12	let PI = 3.14159:		_
13	<pre>let offset = 2 * r outer + r inner;</pre>		
14	for i in 03 {		
15	<pre>let angle = i / 3.0 * 2 * PI;</pre>		X
16	<pre>let shifted = puck.remap_xyz(</pre>		
17	<pre>x + offset * cos(angle),</pre>		
18	<pre>y + offset * sin(angle),</pre>		
19	z);		
20	<pre>three = union(three, shifted);</pre>		
21	}		
22			
23	// smooth blend		
24	let $k = 0.3;$		
25	<pre>let v = three - puck;</pre>		
26	<pre>let out = 0.5 * (puck + three - sqrt(square(v) + k*k));</pre>		
27			
28	// clip to Z bounds		
	0k()	3D (normals)	Rendere





What's so interesting about implicit surfaces?

What does independent research look like?

What are implicit surfaces?

f(x, y, z) = 0 at the surface

$f(x, y, z) \rightarrow \mathbb{R}$

f(x, y, z) < 0 inside the shape

- f(x, y, z) > 0 outside the shape



$f(x, y, z) = x^4 - 5x^2 + y^4 - 5y^2 + z^4 - 5z^2 + 10$

Complex closed-form implicit surfaces

Complex closed-form implicit surfaces

Complex closed-form implicit surfaces

Compact representation ...with arbitrary resolution



(this image is 250633 bytes, 69× larger)

680 math operations 3612 bytes





Solid modeling and CSG (that's Constructive Solid Geometry)



Solid modeling and CSG (that's Constructive Solid Geometry)



A microcosm of CS topics

Graphics programming

Algorithms

Compilers

Data structures

Numerical programming

GPU

programming

A simple example

 $x^2 + y^2$ 1



Constructive Solid Geometry A small example





$$f_2(x, y) = \sqrt{(x-1)^2 + y^2} - 0.5$$



Constructive Solid Geometry Union



 $\min(f_1(x, y), f_2(x, y))$



Constructive Solid Geometry Intersection



 $\max(f_1(x, y), f_2(x, y))$



Constructive Solid Geometry Difference



 $\max(f_1(x, y), -f_2(x, y))$



How do we render these shapes?

RenderingThe naive strategy

 $\max(f_1(x, y), -f_2(x, y))$

Evaluate at every pixel!

 $O(N^2 \times E)$

N is image size E is expression size



Rendering in 3D The naive strategy

$O(N^3 \times E)$

N is image sizeE is expression size

This is not feasible!



One weird trick Interval arithmetic

struct Interval { lower: f32, upper: f32,

$X \in [0, 1]$ $Y \in [2, 4]$ $X + Y \in [2, 5]$

One weird trick Interval arithmetic

 $f(x, y) = \max\left(\sqrt{x^2 + y^2} - 1, \ 0.5 - \sqrt{(x - 1)^2 + y^2}\right)$





1.5

One weird trick Interval arithmetic



$X \in [1, 1.5]$ $Y \in [0.5, 1]$

$f(X, Y) \in [0.11, 0.80]$ f(X, Y) > 0

Interval arithmetic lets us prove regions empty or full





Evaluation complexity Reduced dimensionality





Work is concentrated at the model's edges
Evaluation complexity Amortization over pixels





$M \times M$ pixels

The expression is evaluated once for this region

Interval evaluation cost is amortized over pixels



Amortized evaluation count













Expression simplification The second weird trick

$$\max\left(\sqrt{x^2 + y^2} - 1, \ 0.5 - \sqrt{(x - 1)^2}\right)$$



Expression simplification The second weird trick

$$\max\left(\sqrt{x^2 + y^2} - 1, \ 0.5 - \sqrt{(x - 1)^2}\right)$$

 $X \in [-1, -0.5]$ $Y \in [0.5, 1]$

max ([-0.3, 0.4], [-1.7, -1.1])





Expression simplification The second weird trick

$$\max\left(\sqrt{x^2 + y^2} - 1, \ 0.5 - \sqrt{(x - 1)}\right)$$

max ([-0.3, 0.4], [-1.7, -1.1])

Within this region, we can simplify the expression to $\sqrt{x^2 + y^2} - 1$





Interval arithmetic lets us skip entire chunks of computation

Modified render loop





math operations, 129 of which are CSG





11 math operations are relevant within this region







Amortized evaluation count



From 2D to 3D

3D rendering It's basically the same!





3D rendering Side view





3D rendering Side view





Heightmaps and shading





Heightmaps and shading



Finding surface normals



At the surface of the model, the normal is given by $\left(\frac{\partial f(x, y, z)}{\partial x}, \frac{\partial f(x, y, z)}{\partial y}, \frac{\partial f(x, y, z)}{\partial z}\right)$

 dr	_,	dv	_,	27	
∂X		Оy		0χ	



Partial derivatives in 2D



Partial derivatives in 2D



Partial derivatives in 2D



Gradient operator overloading

struct Grad { value: f32, dx: f32, dy: f32, dz: f32,

$$a = 1, \frac{\partial a}{\partial x} = 0.1$$
$$b = 3, \frac{\partial b}{\partial x} = 0.4$$
$$a + b = 4, \frac{\partial (a + b)}{\partial x} = 0.5$$



Deferred rendering





$1024 \times 1024 \times 1024$ render region

Amortized evaluation count







Fast evaluation of math trees

Our roadmap





```
fn eval(ops, reg count, vars) {
regs = vec![0; reg count]
for (out, op) in ops {
   regs[out] = match op {
      Op::X => vars.x,
      Op::Y => vars.y,
      Op::Const(c) => c,
      Op::Sqrt(arg) => regs[arg].sqrt(),
      Op::Square(arg) => regs[arg].square(),
      Op::Sub(lhs, rhs) => regs[lhs] - regs[rhs],
regs[0]
```

- Op::Add(lhs, rhs) => regs[lhs] + regs[rhs], Op::Max(lhs, rhs) => max(regs[lhs], regs[rhs]),





Equation -> Tree

 $\max\left(\sqrt{x^2 + y^2} - 1, \ 0.5 - \sqrt{(x - 1)^2 + y^2}\right)$





Deduplication



Tree \rightarrow **Graph**



Tree \rightarrow **Graph**





Flattening


We want an instruction ordering such that arguments are defined before they are used

- DFS walk through the graph
- Emit a node once all of its arguments have been emitted



\$0 = var-x

0 = var - x1 = const 1

- 0 = var x
- 1 = const 1
- 2 = sub

1 = const 12 = sub\$3 = square \$24 = var - y\$5 = square \$4 \$6 = add \$3 \$5\$7 = sqrt \$6\$8 = const 0.59 = sub10 = square11 = add 1012 = sqrt13 = sub 1214 = max 9 13

\$0 = var-x

Register allocation

Reducing memory usage

\$0 = var-x1 = const 12 = sub**\$3** = square **\$2** 4 = var-y5 = square\$6 = add \$3 \$5\$7 = sqrt \$6\$8 = const 0.59 = sub10 = square11 = add 1012 = sqrt13 = sub 12 1 $14 = \max 9 13$

r1	=	var-x	
r2	=	const 1	
r4	=	sub r1	r2
r4	=	square	r4
r3	=	var-y	
r3	=	square	r3
r4	=	add r4	r5
r4	=	sqrt r4	
r0	=	const 0	. 5
r0	=	sub r0	гЛ
			14
r1	=	square	r 4 r 1
r1 r1	=	square add r1	r4 r1 r3
r1 r1 r1	=	square add r1 sqrt r1	r4 r1 r3
r1 r1 r1 r1		square add r1 sqrt r1 sub r1	r4 r1 r3

Single static assignment form This is now a compilers talk!

```
\$0 = var-x
1 = const 1
2 = sub 
$3 = square $2
4 = var-y
$5 = square $4
$6 = add $3 $5
$7 = sqrt $6
\$8 = const 0.5
9 = sub 
10 = square
11 = add 10 
12 = sqrt 
13 = sub 12 1
14 = \max 9 13
```

Register allocation Liveness ranges

0 = var - x
1 = const 1
\$2 = sub \$0 \$1
\$3 = square \$2
4 = var-y
\$5 = square \$4
\$6 = add \$3 \$5
\$7 = sqrt \$6
\$8 = const 0.5
\$9 = sub \$8 \$7
\$10 = square \$0
11 = add 10
\$12 = sqrt \$11
\$13 = sub \$12 \$1
14 = max $9 $ 13

Begins when a value is written

Ends when the value is used for the last time

Register allocation Liveness ranges

0 = var - x
1 = const 1
\$2 = sub \$0 \$1
3 = square
4 = var - y
5 = square 4
6 = add 53
\$7 = sqrt \$6
8 = const 0.5
\$9 = sub \$8 \$7
\$10 = square \$0
11 = add 10
12 = sqrt 11
\$13 = sub \$12 \$1
14 = max $9 $ 13

\$10 \$11 \$12 \$13 \$14

Ends when a value is written

Begins when the value is used for the first time

\$0 = var-x1 = const 12 = sub\$3 = square \$24 = var-y\$5 = square \$4\$6 = add \$3 \$5\$7 = sqrt \$6\$8 = const 0.59 = sub10 = square11 = add 1012 = sqrt13 = sub 12 1 $14 = \max 9 13$

- Maintain a value → register mapping
- Bind the output value to r0
- Walk through the instructions in reverse
 - When an value becomes live, bind it to an unused register
 - When a value is no longer live, release its register binding

```
\$0 = var-x
1 = const 1
2 = sub 
\$3 = square \$2
4 = var-y
$5 = square $4
$6 = add $3 $5
$7 = sqrt $6
\$8 = const 0.5
9 = sub 
10 = square 
11 = add 10 
12 = sqrt 
13 = sub 12 1
14 = \max 9 13
```

r0 = max r? r?

SSA value	Register
\$14	r0
_	r1
-	r2
-	r3
-	r4

```
\$0 = var-x
1 = const 1
2 = sub 
\$3 = square \$2
4 = var-y
$5 = square $4
$6 = add $3 $5
$7 = sqrt $6
\$8 = const 0.5
9 = sub 
10 = square 
11 = add 10 
12 = sqrt 
13 = sub 12 1
14 = \max 9 13
```

r0 = max r? r?

SSA value	Register
_	r0
_	r1
_	r2
_	r3
_	r4

```
\$0 = var-x
1 = const 1
2 = sub 
\$3 = square \$2
4 = var-y
$5 = square $4
$6 = add $3 $5
$7 = sqrt $6
\$8 = const 0.5
9 = sub 
10 = square 
11 = add 10 
12 = sqrt 
13 = sub 12 1
14 = \max 9 13
```

r0 = max r0 r?

SSA value	Register
\$9	r0
_	r1
_	r2
_	r3
-	r4

```
\$0 = var-x
1 = const 1
2 = sub 
\$3 = square \$2
4 = var-y
$5 = square $4
$6 = add $3 $5
$7 = sqrt $6
\$8 = const 0.5
9 = sub 
10 = square 
11 = add 10 
12 = sqrt 
13 = sub 12 1
14 = \max 9 13
```

r0 = max r0 r1

SSA value	Register
\$9	r0
\$13	r1
-	r2
_	r3
-	r4

SSA value	Register
\$9	r0
\$13	r1
_	r2
_	r3
_	r4

b r? r? x r0 r1

SSA value	Register
\$9	r0
_	r1
_	r2
_	r3
_	r4

b r? r? x r0 r1

SSA value	Register
\$9	r0
\$12	r1
_	r2
_	r3
_	r4

b r1 r? x r0 r1

SSA value	Register
\$9	r0
\$12	r1
\$1	r2
_	r3
_	r4

b r1 r2 x r0 r1

SSA value	Register
\$9	r0
\$12	r1
\$1	r2
-	r3
_	r4

SSA value	Register
\$9	r0
_	r1
\$1	r2
_	r3
_	r4

SSA value	Register
\$9	r0
\$11	r1
\$1	r2
_	r3
_	r4

qrt r1 ub r1 r2 ax r0 r1

```
\$0 = var-x
1 = const 1
2 = sub 
$3 = square $2
4 = var-y
$5 = square $4
$6 = add $3 $5
$7 = sqrt $6
\$8 = const 0.5
9 = sub 
10 = square 
11 = add 10  10  r1 = add r1  r3
12 = sqrt 11
                   r1 = sqrt r1
13 = sub 12 1
                   r1 = sub r1 r2
14 = \max 9 13
                   r0 = max r0 r1
```

SSA value	Register
\$9	r0
\$10	r1
\$1	r2
\$5	r3
-	r4

```
\$0 = var-x
1 = const 1
2 = sub 
$3 = square $2
4 = var-y
$5 = square $4
$6 = add $3 $5
$7 = sqrt $6
\$8 = const 0.5
9 = sub 
10 = square  1 = square  r1 = square  r1 = square 
11 = add 10  10  r1 = add r1  r3
12 = sqrt 11
                    r1 = sqrt r1
13 = sub 12 1
                    r1 = sub r1 r2
14 = \max 9 13
                    r0 = max r0 r1
```

SSA value	Register
\$9	r0
\$O	r1
\$1	r2
\$5	r3
_	r4

```
0 = var - x
1 = const 1
2 = sub 
$3 = square $2
4 = var-y
$5 = square $4
$6 = add $3 $5
$7 = sqrt $6
\$8 = const 0.5
9 = sub  8  7  r0 = sub  r0  r4
10 = square  1 = square  r1 = square  r1 = square 
11 = add 10  10  r1 = add r1  r3
12 = sqrt 11
                    r1 = sqrt r1
13 = sub 12 1
                    r1 = sub r1 r2
14 = \max 9 13
                    r0 = max r0 r1
```

SSA value	Register
\$8	r0
\$O	r1
\$1	r2
\$5	r3
\$7	r4

```
0 = var - x
1 = const 1
2 = sub 
\$3 = square \$2
4 = var-y
$5 = square $4
$6 = add $3 $5
$7 = sqrt $6
\$8 = const 0.5 r0 = const 0.5
9 = sub 
10 = square  1 = square  r1 = square  r1 = square 
11 = add 10  10  r1 = add r1  r3
12 = sqrt 11
13 = sub 12 1
14 = \max 9 13
```

- r0 = sub r0 r4
- r1 = sqrt r1r1 = sub r1 r2

SSA value	Register
_	r0
\$O	r1
\$1	r2
\$5	r3
\$7	r4

r0 = max r0 r1

```
0 = var - x
1 = const 1
2 = sub 
$3 = square $2
4 = var-y
$5 = square $4
$6 = add $3 $5
$7 = sqrt $6
\$8 = const 0.5
9 = sub 
10 = square 
11 = add 10 
12 = sqrt 11
13 = sub 12 1
14 = \max 9 13
```

- r4 = sqrt r4
- r0 = sub r0 r4r1 = square r1
- r1 = add r1 r3r1 = sqrt r1
- r1 = sub r1 r2r0 = max r0 r1

SSA value	Register
_	r0
\$O	r1
\$1	r2
\$5	r3
\$6	r4

r0 = const 0.5

```
0 = var - x
1 = const 1
2 = sub 
\$3 = square \$2
4 = var-y
$5 = square $4
$6 = add $3 $5
$7 = sqrt $6
\$8 = const 0.5
9 = sub 
10 = square 
11 = add 10 
12 = sqrt 11
13 = sub 12 1
14 = \max 9 13
```

- r4 = add r4 r5r4 = sqrt r4r0 = const 0.5r0 = sub r0 r4
- r1 = square r1r1 = add r1 r3r1 = sqrt r1
- r1 = sub r1 r2r0 = max r0 r1

SSA value	Register
_	r0
\$O	r1
\$1	r2
\$5	r3
\$3	r4

```
0 = var - x
1 = const 1
2 = sub 
\$3 = square \$2
4 = var-y
s_5 = square square r_3 = square sq
 $6 = add $3 $5
$7 = sqrt $6
 \$8 = const 0.5
 $9 = sub $8 $7
10 = square 
11 = add 10 
12 = sqrt 11
13 = sub 12 1
14 = \max 9 13
```

- r4 = add
- r4 = sqrr0 = con
- r0 = subr1 = squ
- r1 = addr1 = sqr
- r1 = subr0 = max

lare	ן כ	r	3
r r	1	r	5
t r	-4		
st	0	•	5
re		r	4
lare	ן כ	r	1
r1	L	r	3
t r	-1		
r1	LI	r	2
re		r	1

SSA value	Register
_	r0
\$O	r1
\$1	r2
\$4	r3
\$3	r4

```
0 = var - x
1 = const 1
2 = sub 
\$3 = square \$2
4 = var-y
$5 = square $4
$6 = add $3 $5
$7 = sqrt $6
\$8 = const 0.5
9 = sub 
10 = square 
11 = add 10 
12 = sqrt 11
13 = sub 12 1
14 = \max 9 13
```

r3 = var-yr3 = square r3r4 = add r4 r5r4 = sqrt r4r0 = const 0.5r0 = sub r0 r4r1 = square r1r1 = add r1 r3r1 = sqrt r1

r1 = sub r1 r2r0 = max r0 r1

SSA value	Register
_	r0
\$O	r1
\$1	r2
_	r3
\$3	r4

- 0 = var x1 = const 1\$2 = sub \$0 \$13 = square4 = var-y5 = square 4\$6 = add \$3 \$5\$7 = sqrt \$6 \$8 = const 0.59 = sub10 = square11 = add 1012 = sqrt 1113 = sub 12 1 $14 = \max 9 13$
 - r4 = square r4
 - r3 = var-y r3 = square r3r4 = add r4 r5
 - r4 = sqrt r4 r0 = const 0.5r0 = sub r0 r4
 - r1 = square r1
 r1 = add r1 r3
 - r1 = sqrt r1
 r1 = sub r1 r2
 - r0 = max r0 r1

SSA value	Register
_	r0
\$O	r1
\$1	r2
_	r3
\$2	r4

0 = var - x
1 = const 1
\$2 = sub \$0 \$1
3 = square
4 = var-y
\$5 = square \$4
\$6 = add \$3 \$5
\$7 = sqrt \$6
\$8 = const 0.5
\$9 = sub \$8 \$7
\$10 = square \$0
11 = add 10
\$12 = sqrt \$11
\$13 = sub \$12 \$1
$14 = \max 9 13$

- r4 = sub r1 r2
- r4 = square r4
- r3 = var-yr3 = square r3
- r4 = sqrt r4
- r0 = sub r0 r4r1 = square r1
- r1 = add r1 r3r1 = sqrt r1
- r1 = sub r1 r2

```
r4 = add r4 r5
r0 = const 0.5
r0 = max r0 r1
```

SSA value	Register
_	r0
\$O	r1
\$1	r2
_	r3
_	r4

0 = var - x
1 = const 1
\$2 = sub \$0 \$1
\$3 = square \$2
4 = var - y
\$5 = square \$4
\$6 = add \$3 \$5
\$7 = sqrt \$6
\$8 = const 0.5
\$9 = sub \$8 \$7
10 = square
11 = add 10
\$12 = sqrt \$11
\$13 = sub \$12 \$1
$14 = \max 9 13$

- $r_2 = const_1$
- r4 = sub r1 r2
- r4 = square r4
- r3 = var-yr3 = square r3r4 = add r4 r5
- r4 = sqrt r4
- r0 = sub r0 r4
- r1 = add r1 r3
- r1 = sqrt r1r1 = sub r1 r2
- r0 = max r0 r1

```
r0 = const 0.5
r1 = square r1
```

SSA value	Register
_	r0
\$O	r1
-	r2
_	r3
_	r4

0 = var - x	ľ
1 = const 1	ľ
\$2 = sub \$0 \$1	ľ
\$3 = square \$2	ľ
4 = var - y	ľ
\$5 = square \$4	ľ
\$6 = add \$3 \$5	ľ
\$7 = sqrt \$6	ľ
8 = const 0.5	ľ
\$9 = sub \$8 \$7	ľ
\$10 = square \$0	ľ
11 = add 10	ľ
\$12 = sqrt \$11	ľ
\$13 = sub \$12 \$1	ľ
14 = max $9 $ 13	ľ

- r1 = var-x
- r4 = sub r1 r2
- r4 = square r4
- r3 = var-yr3 = square r3
- r4 = add r4 r5r4 = sqrt r4
- r0 = const 0.5
- r1 = square r1
- r1 = sqrt r1r1 = sub r1 r2
- r0 = max r0 r1

```
r_2 = const_1
r0 = sub r0 r4
r1 = add r1 r3
```

SSA value	Register
_	r0
_	r1
-	r2
_	r3
_	r4

Register allocation Liveness ranges + simplification

```
\$0 = var-x
1 = const 1
2 = sub 
$3 = square $2
4 = var-y
$5 = square $4
$6 = add $3 $5
7 = sqrt 
\$8 = const 0.5
9 = sub 
10 = square 
11 = add 10 
12 = sqrt 
13 = sub 12 1
14 = \max 9 13
```

A value the val If min one or

- A value's liveness range begins when the value is used for the first time
- If min or max can be simplified, then one or the other argument is **not used**
Register allocation Liveness ranges + simplification

```
\$0 = var-x
1 = const 1
2 = sub 
$3 = square $2
4 = var-y
$5 = square $4
$6 = add $3 $5
7 = sqrt 
\$8 = const 0.5
9 = sub 
10 = square
11 = add 10 
12 = sqrt 
13 = sub 12 1
14 = \max 9 13
```

A value the value If min one or

- A value's liveness range begins when the value is used for the first time
- If min or max can be simplified, then one or the other argument is **not used**

Register allocation Reverse linear scan with simplification

\$0 = var-x1 = const 12 = sub\$3 = square \$24 = var-y5 = square\$6 = add \$3 \$57 = sqrt\$8 = const 0.59 = sub10 = square11 = add 1012 = sqrt13 = sub 12 1 $14 = \max 9 13$

- Maintain a value \rightarrow register mapping
- Bind the output value to r0
- Walk through the instructions in reverse

 - When an value becomes live, bind it to an unused register
 - When a value is no longer live, release its register binding

If the instruction's output value is not live, then skip it!

Register allocation Reverse linear scan with simplification

r0 = var
r1 = con
r2 = var
r2 = squ
r0 = squ
r0 = add
r0 = sqr
r0 = sub
r0 = cop

-x st 1

-y are r2

are r0 r0 r2 t r0 r0 r1 y r0



Deduplication



Flattening

Register allocation

Bytecode interpreter

r1 = var-xr2 = const 1r4 = sub r1 r2r4 = square r4r3 = var-yr3 = square r3r4 = add r4 r5r4 = sqrt r4r0 = const 0.5r0 = sub r0 r4r1 = square r1 r1 = add r1 r3r1 = sqrt r1r1 = sub r1 r2r0 = max r0 r1

- regs[0]

```
fn eval(ops, reg_count, vars) {
 regs = vec![0; reg count]
  for (out, op) in ops {
   regs[out] = match op {
     Op::X => vars.x,
     Op::Y => vars.y,
     Op::Const(c) => c,
     Op::Sqrt(arg) => regs[arg].sqrt(),
     Op::Square(arg) => regs[arg].square(),
     Op::Sub(lhs, rhs) => regs[lhs] - regs[rhs],
     Op::Add(lhs, rhs) => regs[lhs] + regs[rhs],
     Op::Max(lhs, rhs) => max(regs[lhs], regs[rhs]),
```

Bytecode interpreter overhead

```
fn eval(ops, reg_count, vars) {
  regs = vec![0; reg_count]
  for (out, op) in ops {
   regs[out] = match op {
     Op::X => vars.x,
     Op::Y => vars.y,
     Op::Const(c) => c,
      Op::Sqrt(arg) => regs[arg].sqrt(),
      Op::Square(arg) => regs[arg].square(),
      Op::Sub(lhs, rhs) => regs[lhs] - regs[rhs],
      Op::Add(lhs, rhs) => regs[lhs] + regs[rhs],
      Op::Max(lhs, rhs) => max(regs[lhs], regs[rhs]),
  regs[0]
```

Unpredictable branch

Lots of reading and writing to RAM



Bytecode → Assembly The final frontier

- r0 = var-y
- r0 = square r0
- r1 = var-x
- r1 = square r1
- r0 = add r0 r2
- r0 = sqrt r0
- r1 = const 1
- r0 = sub r0 r1

ldr s0, [x0, 4]fmul s0, s0, s0 ldr s1, [x0, 0]fmul s1, s1, s1 fadd s0, s0, s1 fsqrt s0, s0 movz w9, $0 \times 3 f 80$, lsl 16 fmov s1, w9 fsub s0, s0, s1





But this rough magic I here abjure, and when I have required some heavenly music, which even now I' do, to work mine end upon their senses that this airy charm is tor, break 'my staff, bury certain fathoms the Īn did earth, and deeper than plummet sound L'II drown my book.

7867 math operations, 2354 of which are CSG

But this rough magic I here abjure, and when L have required some heavenly music, which even do, to work mine Ľ now end upon their senses that this airy charm is tor, break 'my staff, bury certain fathoms the In did than earth, and deeper plummet sound l'll drown my booK.





But this rough magic 1 here abjure, and when L have required some heavenly music, which even do, to work mine now Ľ end upon their senses that this airy charm is tor, break 'my staff, bury certain fathoms The In did than earth, and deeper sound plummet 'er l'll drown my booK.



But this rough magic 1 here abjure, and when L have required some <u>heavenly music, which even</u> do, to work mine now Ľ end upon their senses that this airy charm is tor, break 'my staff, bury certain fathoms the In did than earth, and deeper sound plummet 'er L'II drown my booK.



But this rough magic 1 here abjure, and when L have required some <u>heavenly music, which even</u> do, to work mine now Ľ end upon their senses that this airy charm is tor, break 'my staff, bury certain fathoms the In did earth, and deeper than plummet sound ever L'II drown my booK.



Graphics programming

Algorithms

Compilers

Data structures

Numerical programming

GPU

programming

What does independent research look like?



You can just do things

publish blog posts

write software

email authors

submit to journals



Learn more about a subject Get better at the craft Recognize promising ideas

Do things

Read blog posts Read papers Write software Do experiments Write blog posts Submit papers Publish demos Talk about your work on social media

Talk about them

Meet interesting people Learn about their ideas Synthesize from conversations

Kokopelli (2013) C, Python, script-based UI

A A A

46 tooth.shape - True 47 tooth &= reflect_y(tooth) 48 40 # If we have an odd number of teeth, then we can't take # advantage of bilateral tooth symmetry. 51 1F N % 21 tooth &= -X teeth = reduce(operator.add, [rotate(tooth, 1*360/N) for i in range(N)]) 58 ss else: teeth = reduce(operator.add, [rotate(tooth, i*368/N) for i in range(N/2)]) 59 teeth += circle(0, 0, RR) is teeth &= circle(0, 0, R0) - circle(0, 0, RR*0.9) it teeth.bounds = circle(0, 0, RO).bounds 62 teeth = extrusion(teeth, -0.1, 0.1) 63 54 teeth.color = 'red' 12 as # Create a set of six ribs inside the gear i7 ribs = rectangle(-0.002*N, 0.002*N, -RR*0.95, RR*0.95) Fill ribs = reduce(operator.add, [rotate(ribs, i*120) for i in n i0 ribs += circle(0, 0, 0.4) 78 ribs -= circle(0,0,0.25) 71 ribs -= rectangle(-0.06, 0.06, 0, 0.3) 72 ribs = extrusion(ribs, -0.08, 0.08) 73 ribs.color - 'green' 75 # Create a base for the gear 76 base = circle(0, 0, RR*0.95) - circle(0, 0, 0.35) 27 base -= rectangle(-0.06, 0.06, 0, 0.3) 71 base = extrusion(base, -0.04, 0.04) 79 base.color - 'blue' A1 cod.shapes - Leeth, ribs, base 82 183 **MI Notes** - 00 #7 We want to find the angle such that the involute curve II intersects a circle of radius R, where the involute is being ID unwound from a circle of radius RB (and RB < R).</p> In the involute has coordinates x, y = RB*(cos(t)+t*sin(t)), RB*(sin(t)-t*cos(t)) Re-render the output image



Antimony (2015) C/C++, Python, same kernel, graph-based UI



libfive + Studio (2018) C++, Scheme, new kernel with robust meshing

$\Theta \Theta \Theta$

```
;; All shapes are defined in mm units
(define radius-wire 3/2)
(define radius-iron 10/2)
(define base (circle #[0 0] (+ radius-wire #2)))
(define rest (let ((shift #8.06361)))
 (difference
    (circle #[shift 0] (+ radius-iron #4.02195))
    (circle #[shift 0] radius-iron)
    (lambda-shape (x y z) (- shift x)))))
(define cutout-height 8)
(define base-height 10)
(define loft-height (+ base-height 10))
(define rest-height (+ loft-height 10))
(reflect-xz (union
  (difference
    (extrude-z base 0 base-height)
    (cylinder-z #[0 0 -1] radius-wire cutout-height))
  (loft base rest base-height loft-height)
  (extrude-z rest loft-height rest-height)))
```





Porting to CUDA (2020) **GPU-powered rendering**

View		
▼ Tex	t editor	
1	(set-bounds!*[-50*-50*-20]*[50*50*20])	
2	(set-quality! 8)	
3	(set-resolution 15)	
4		
5	;(box*[-5*-0.5]*[5*5*0])	
6	(define arch	
7	''(let((h'2.5)'(r'1.5))	
8	<pre>```(union (rectangle [(- r) 0] [r h])</pre>	
9	<pre>(circle r [0 h])))</pre>	
10		
11	(define⁺subpillar	
12	**(union	
13	(box [5.7 -0.2 6] [6.8 0.9 8.3])	
14	<pre>(scale-z*(sphere*0.5*[6.25*0.35*8.1])</pre>	
15	2 8.3)	
16	(cylinder-z 0.1 0.8 [6.25 0.35 8.3])	
17	(sphere 0.1 L6.25 0.35 9.1)	
18		
19		
20	(define ralling-cut	
21	(define reiling multicut	
22	(derine railing-multicut	
2.3	(union	

#<<shape> 109b5c000>

Shapes Shape at 0x7fc9cfec0ee0 Render time: 0.091248 s SSAO time: 0.112193 s Texture load time: 0.007226 s Save shape.frep







Fidget (2024) Rust, WebAssembly, new kernel with JIT compiler

```
let r_outer = 1;
    let r_inner = 0.35;
 2
 3
    let r = sqrt(square(x) + square(y));
 4
    let cross = union(
 5
      sqrt(square(x - r_outer) + square(z)) - r_inner,
 6
      intersection(x - r_outer,
        intersection(z - r_inner, -r_inner - z)));
 8
    let puck = cross.remap_xyz(r, y, z);
 9
10
    let three = 10000.0;
11
    let PI = 3.14159;
12
    let offset = 2 * r_outer + r_inner;
13
    for i in 0..3 {
14
        let angle = i / 3.0 * 2 * PI;
15
        let shifted = puck.remap_xyz(
16
            x + offset * cos(angle),
17
18
            y + offset * sin(angle),
            z);
19
20
        three = union(three, shifted);
21
22
    // smooth blend
23
    let k = 0.3;
24
   let v = three - puck;
    let out = 0.5 * (puck + three - sqrt(square(v) + k*k));
26
27
   // clip to Z bounds
28
    0k(..)
```



Implicit Surfaces & Independent Research 2025 Fidget

- 2024 Fidget: Yet Another Implicit Kernel Efficiently updating implicit in-order forests A simple adversarial model for dual contouring
- Do Not Taunt Happy Fun Branch Predictor 2022 The Solid-State Register Allocator Ray tracing with M-reps Writing a SIGGRAPH paper (for fun) Implicit Surfaces on the GPU
- 2020 Massively Parallel Rendering of Complex Closed-Form Implicit Surfaces (+talk) Quadratic Error Function Explainer Consulting on libfive libfive + Studio
- Implicit Kernels for Solid Modeling 2018 Consistent Ordering of N-Dimensional Neighbors

2017	QEFs, Eigenvalues, and Normals
2016	Finding bounding boxes with interval math
	Fixing a soldering iron with 3D printing
	Zero-crossing logic for robust meshing
	Higher-order reactive graph programming
	Lineage of CBA CAD tools
	Abstraction and instances in graph program
	Ao: Homoiconic solid modeling
	Automatic tracking of bounding boxes
	Affine coordinates in Ao
	Ao
	Representation and JITting of math trees
2013	2D contouring
	Antimony
	Kokopelli



Implicit Surfaces & Independent Research Fidget

Fidget: Yet Another Implicit Kernel Efficiently updating implicit in-order forests A simple adversarial model for dual contouring Do Not Taunt Happy Fun Branch Predictor The Solid-State Register Allocator Ray tracing with M-reps Writing a SIGGRAPH paper (for fun) Implicit Surfaces on the GPU Massively Parallel Rendering of Complex Closed-Form Implicit Surfaces (+talk) **Quadratic Error Function Explainer** Consulting on libfive libfive + Studio Implicit Kernels for Solid Modeling **Consistent Ordering of N-Dimensional Neighbors** QEFs, Eigenvalues, and Normals Finding bounding boxes with interval math Fixing a soldering iron with 3D printing Zero-crossing logic for robust meshing Higher-order reactive graph programming Lineage of CBA CAD tools Abstraction and instances in graph programming Ao: Homoiconic solid modeling Automatic tracking of bounding boxes Affine coordinates in Ao Ao Representation and JITting of math trees 2D contouring Antimony Kokopelli

19 blog posts 6 presentations 5 software packages 1 research paper 1 paid consulting gig Not shown: Dozens of email conversations and one-on-one interactions Too many tweets



Motivation and energy

Staying motivated

Project completion



Staying motivated through the power of blog-post driven development

Project completion (%)



about the project

Blog-post driven development Why does this work?

- Stop doing open-ended work!
- Narrowly focus on the writeup
- Reminds me what's cool about the project
- Sharing work has social benefits
- You should have a website

Coming up with ideas



Implicit surface rendering

Fast evaluation and simplification

Robust meshing





Raph Levien's blog

2D Graphics on Modern GPU

May 8, 2019

Massively Parallel Rendering of Complex Closed-Form Implicit Surfaces

MATTHEW J. KEETER, Independent researcher



Fig. 1. An assortment of implicit surfaces rendered using our technique. Left: an extruded text string, rotated and rendered as a heightmap. Center: a bear head sculpted using smooth blending operations, with normals found by automatic differentiation. Right: a complex architectural model rendered with screen-space ambient occlusion and perspective. All models are rendered directly from their mathematical representations, without triangulation or raytracing.

We present a new method for directly rendering complex closed-form implicit surfaces on modern GPUs, taking advantage of their massive parallelism. Our model representation is unambiguously solid, can be sampled at arbitrary resolution, and supports both constructive solid geometry (CSG) and more unusual modeling operations (e.g. smooth blending of shapes). The rendering strategy scales to large-scale models with thousands of arithmetic operations in their underlying mathematical expressions. Our method only requires C^0 continuity, allowing for warping and blending operations which break Lipshitz continuity.

To render a model, its underlying expression is evaluated in a shallow hierarchy of spatial regions, using a high branching factor for efficient parallelization. Interval arithmetic is used to both skip empty regions and construct reduced versions of the expression. The latter is the optimization that makes our algorithm practical: in one benchmark, expression complexity decreases by two orders of magnitude between the original and reduced expressions. Similar algorithms exist in the literature, but tend to be deeply recursive with heterogeneous workloads in each branch, which makes them GPU-unfriendly; our evaluation and expression reduction both run efficiently as massively parallel algorithms, entirely on the GPU.

The resulting system renders complex implicit surfaces in high resolution and at interactive speeds. We examine how performance scales with computing power, presenting performance results on hardware ranging from older laptops to modern data-center GPUs, and showing significant improvements at each stage.

CCS Concepts: • Computing methodologies \rightarrow Rasterization; Volumetric models.

Additional Key Words and Phrases: implicit surface, signed distance field, freps, octrees, rasterization, gpu, cuda

© 2020 Copyright held by the owner/author(s). Publication rights licensed to ACM. 0730-0301/2020/7-ART141 \$15.00 https://doi.org/10.1145/3386569.3392429

ACM Reference Format:

Matthew J. Keeter. 2020. Massively Parallel Rendering of Complex Closed-Form Implicit Surfaces. *ACM Trans. Graph.* 39, 4, Article 141 (July 2020), 10 pages. https://doi.org/10.1145/3386569.3392429

1 INTRODUCTION

Implicit surfaces and functional representations are a powerful way to represent solid models [Bloomenthal and Wyvill 1997; Gomes et al. 2009]. Compared to boundary representations (e.g. triangle meshes or NURBS surfaces), they offer unambiguous inside-outside checking, easy constructive solid geometry (CSG) operations, and arbitrary resolution. In recent years, functional representations (freps) have been used as the kernel of both commercial [Courter 2019] and open-source [Keeter 2019] CAD packages. They are a fundamental building block in the demoscene community [Burger et al. 2002; Quilez 2008], used as a representation for generative art [Moen 2019], and even as the underlying technology for a recent PlayStation 4 game [Evans 2015].

Unlike boundary representations, implicit surfaces cannot easily be rendered in their native forms. This paper presents a new method for rendering the family of implicit surfaces represented by arbitrary closed-form arithmetic expressions, i.e., representing a sphere as

$$f(x, y, z) < 0$$
 where $f(x, y, z) = \sqrt{x^2 + y^2 + z^2} - 1$

This representation is particularly flexible and can be treated as an "assembly language for shapes" which is targeted by higherlevel tools. The space of higher-level tools spans the gamut from advanced solid modeling packages [Allen 2019] to user-friendly content generation tools [Keeter 2015].

Our rendering strategy runs in both 2D and 3D, making efficient use of modern GPU hardware and APIs. Unlike previous work, it scales to complex expressions, maintaining interactive framerates while rendering models built from hundreds or thousands of arithmetic operations. It requires no continuity higher than C^0 , which allows for extremely flexible modeling and unusual spatial transformations. Finally, it scales well with GPU power; as GPU performance

ACM Trans. Graph., Vol. 39, No. 4, Article 141. Publication date: July 2020.



 $Author's \ address: Matthew \ J. \ Keeter, \ Independent \ researcher, \ matt.j. \\ keeter@gmail.com.$

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

Where to find pet problems?

- Adopt them
- Read a lot (papers, blog posts, etc)
 - Find interesting people through news aggregators
 - Subscribe to them via RSS
- Look for personal-scale problems
- Revisiting old ideas on modern hardware

Thank you!

Matt Keeter mattkeeter.com