2D Computer Graphics

Diego Nehab
Summer 2020

IMPA

Path representation

SVG path commands

Command		Parameters	Description
Abs	Rel		
`M`	`m`	$(x, y)+$	move
`L`	`l`	$(x, y)+$	line
`H`	`h`	$x+$	horizontal line
`V`	`v`	$y+$	vertical line
`C`	`c`	$(x_1, y_1, x_2, y_2, x, y)+$	cubic
`S`	`s`	$(x_2, y_2, x, y)+$	smooth cubic
`Q`	`q`	$(x_1, y_1, x, y)+$	quadratic
`T`	`t`	$(x, y)+$	smooth quadratic
`A`	`a`	$(r_x, r_y, \theta_x, \ell, o, x, y)+$	elliptical arc
`Z`	`z`		close path

Our representation

Input from SVG commands

Our representation

Input from SVG commands

- Relative control points converted to absolute

Our representation

Input from SVG commands

- Relative control points converted to absolute
- H, V, S, T converted to generic segments

Our representation

Input from SVG commands

- Relative control points converted to absolute
- H, V, S, T converted to generic segments
- A converted to rational quadratics (R command)

Our representation

Input from SVG commands

- Relative control points converted to absolute
- H, V, S, T converted to generic segments
- A converted to rational quadratics (R command)

Convert other primitives to paths

```
path_data = shape:as_path_data()
```

Content visible using *iterators*

Our representation

Input from SVG commands

- Relative control points converted to absolute
- H, V, S, T converted to generic segments
- A converted to rational quadratics (R command)

Convert other primitives to paths

```
path_data = shape:as_path_data()
```

Content visible using *iterators*

```
path_data:iterate{
  begin_contour = function(self, x0, y0) end
  end_open_contour = function(self, x0, y0) end
  end_closed_contour = function(self, x0, y0) end
  linear_segment = function(self, x0, y0, x1, y1) end
  quadratic_segment = function(self, x0, y0, x1, y1, x2, y2) end
  rational_quadratic_segment = function(self, x0, y0, x1, y1, w1, x2, y2) end
  cubic_segment = function(self, x0, y0, x1, y1, x2, y2, x3, y3) end
}
```

Example of filter

Transform a path and forward results on

```
function filter.make_input_path_f_xform(xf, forward)
  local px, py -- previous cursor
  local xformer = {}
  function xformer:begin_contour(x0, y0)
      px, py = xf:apply(x0, y0)
      forward:begin_contour(px, py)
  end
  function xformer:end_closed_contour(x0, y0)
      forward:end_closed_contour(px, py)
  end
  function xformer:linear_segment(x0, y0, x1, y1)
     x1, y1 = xf:apply(x1, y1)
     forward:linear_segment(px, py, x1, y1)
     px, py = x1, y1
  end
  function xformer:rational_quadratic_segment(x0, y0, x1, y1, w1, x2, y2)
      x1, y1, w1 = xf:apply(x1, y1, w1)
      x2, y2 = xf:apply(x2, y2)
      forward:rational_quadratic_segment(px, py, x1, y1, w1, x2, y2)
      px, py = x2, y2
  end
  ...
  return xformer
end
```

Example of filter chaining

Provided `filter.make_input_path_f_xform` transforms path

Example of filter chaining

Provided `filter.make_input_path_f_xform` transforms path

Implement `monotonize` to break into monotonic segments

Example of filter chaining

Provided `filter.make_input_path_f_xform` transforms path

Implement `monotonize` to break into monotonic segments

Implement `accelerate` to convert and store in your representation

Example of filter chaining

Provided `filter.make_input_path_f_xform` transforms path

Implement `monotonize` to break into monotonic segments

Implement `accelerate` to convert and store in your representation

Chain transformation, monotonization, and acceleration

```
path_xf = shape:get_xf():transform(cur_xf)
shape:as_path_data():iterate(
  filter.make_input_path_f_xform(path_xf,
    monotonize(
      accelerate(accel)))
```

Floating-point and root-finding

All real numbers represented in a computer?

Floating-point numbers

All real numbers represented in a computer?

Impossible, of course. Finite memory!

Floating-point numbers

All real numbers represented in a computer?

Impossible, of course. Finite memory!

Can only represent a finite set of values

Floating-point numbers

All real numbers represented in a computer?

Impossible, of course. Finite memory!

Can only represent a finite set of values

Widely accepted IEEE floating-point standard

Floating-point numbers

All real numbers represented in a computer?

Impossible, of course. Finite memory!

Can only represent a finite set of values

Widely accepted IEEE floating-point standard

Formats, rounding, arithmetic operations

Floating-point numbers

All real numbers represented in a computer?

Impossible, of course. Finite memory!

Can only represent a finite set of values

Widely accepted IEEE floating-point standard

Formats, rounding, arithmetic operations

Represented by familiar scientific notation

$$N_A = 6.022140857 \times 10^{23} \qquad q_e = 1.60217662 \times 10^{-19}$$

Floating-point numbers

All real numbers represented in a computer?

Impossible, of course. Finite memory!

Can only represent a finite set of values

Widely accepted IEEE floating-point standard

Formats, rounding, arithmetic operations

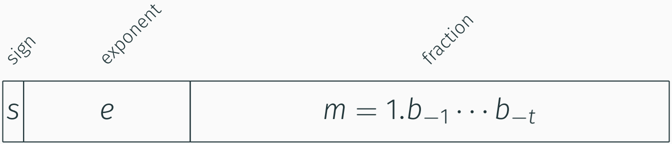
Represented by familiar scientific notation

$$N_A = 6.022140857 \times 10^{23} \qquad q_e = 1.60217662 \times 10^{-19}$$

Except, in binary...

Binary floating-point

$$(-1)^s \times (1.b_{-1}b_{-2}b_{-3}\cdots b_{-t})_2 \times 2^{e-z} \qquad z = 2^{w-1} - 1$$



One *sign* bit

w exponent bits

t fraction bits

$$(-1)^s \times (1.b_{-1}b_{-2}b_{-3}\cdots b_{-t})_2 \times 2^{e-z} \qquad z = 2^{w-1} - 1$$

Normalized representation for mantissa m

- Ensures unique representation for mantissa

$$1_2 \leq m < 10_2$$

- First bit is implicitly set to 1

Representation details

$$(-1)^s \times (1.b_{-1}b_{-2}b_{-3}\cdots b_{-t})_2 \times 2^{e-z} \qquad z = 2^{w-1} - 1$$

Normalized representation for mantissa m

- Ensures unique representation for mantissa

$$1_2 \leq m < 10_2$$

- First bit is implicitly set to 1

Excess encoding for exponent e

- Allows for positive and negative exponents
- Therefore large and small magnitudes
- Subtract $z = 2^{w-1} - 1$ from encoded exponent

Special values

Largest representable exponent is reserved

- $m = 0$ represents $\pm Inf$ (Infinity)
- $m \neq 0$ represents NaN (Not-a-Number)

Special values

Largest representable exponent is reserved

- $m = 0$ represents $\pm Inf$ (Infinity)
- $m \neq 0$ represents NaN (Not-a-Number)

NaN propagates and compares as false

- $NaN \odot x \rightarrow NaN$
- $NaN = NaN \rightarrow$ `false`

Special values

Largest representable exponent is reserved

- $m = 0$ represents $\pm Inf$ (Infinity)
- $m \neq 0$ represents NaN (Not-a-Number)

NaN propagates and compares as false

- $NaN \odot x \rightarrow NaN$
- $NaN = NaN \rightarrow$ `false`

Other special operations

$$x \div Inf = \pm 0 \qquad 0 \div 0 = NaN$$

$$Inf \times Inf = Inf \qquad Inf - Inf = NaN$$

$$x \div 0 = \pm Inf \qquad Inf \div Inf = NaN$$

$$Inf + Inf = Inf \qquad Inf \times 0 = NaN$$

Denormalization

$$(-1)^s \times (1.b_{-1}b_{-2}b_{-3}\cdots b_{-t})_2 \times 2^{e-z} \qquad z = 2^{w-1} - 1$$

Same number of values for each interval $[2^i, 2^{i+1}]$

$$(-1)^s \times (1.b_{-1}b_{-2}b_{-3}\cdots b_{-t})_2 \times 2^{e-z} \qquad z = 2^{w-1} - 1$$

Same number of values for each interval $[2^i, 2^{i+1}]$

Denormalization

$$(-1)^s \times (1.b_{-1}b_{-2}b_{-3}\cdots b_{-t})_2 \times 2^{e-z} \qquad z = 2^{w-1} - 1$$

Same number of values for each interval $[2^i, 2^{i+1}]$

$$(-1)^s \times (1.b_{-1}b_{-2}b_{-3}\cdots b_{-t})_2 \times 2^{e-z} \qquad z = 2^{w-1} - 1$$

Same number of values for each interval $[2^i, 2^{i+1}]$

Denormalization

$$(-1)^s \times (1.b_{-1}b_{-2}b_{-3}\cdots b_{-t})_2 \times 2^{e-z} \qquad z = 2^{w-1} - 1$$

Same number of values for each interval $[2^i, 2^{i+1}]$

What happens when exponent is the smallest?

Denormalization

$$(-1)^s \times (1.b_{-1}b_{-2}b_{-3}\cdots b_{-t})_2 \times 2^{e-z} \qquad z = 2^{w-1} - 1$$

Same number of values for each interval $[2^i, 2^{i+1}]$

What happens when exponent is the smallest?

Normalization causes abrupt underflow

- From $1.\overbrace{00\cdots1}^{t}\times 2^{-z}$ to 0

Denormalization

$$(-1)^s \times (1.b_{-1}b_{-2}b_{-3}\cdots b_{-t})_2 \times 2^{e-z} \qquad z = 2^{w-1} - 1$$

Same number of values for each interval $[2^i, 2^{i+1}]$

What happens when exponent is the smallest?

Normalization causes abrupt underflow

- From $1.\overbrace{00\cdots1}^{t} \times 2^{-z}$ to 0

Instead, *denormalized* numbers were introduced

- When exponent is smallest, there is no implicit leading 1

Denormalization

$$(-1)^s \times (1.b_{-1}b_{-2}b_{-3}\cdots b_{-t})_2 \times 2^{e-z} \qquad z = 2^{w-1} - 1$$

Same number of values for each interval $[2^i, 2^{i+1}]$

What happens when exponent is the smallest?

Normalization causes abrupt underflow

- From $1.\overbrace{00\cdots1}^{t}\times 2^{-z}$ to 0

Instead, *denormalized* numbers were introduced

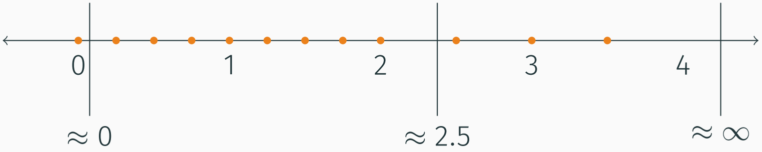
- When exponent is smallest, there is no implicit leading 1

Summary of representation

	sign	exponent	fraction
normalized	±	not $0\cdots0$ or $1\cdots1$	any
de-normalized	±	$0\cdots0$	not $0\cdots0$
zero	±	$0\cdots0$	$0\cdots0$
Inf	±	$1\cdots1$	$0\cdots0$
NaN	±	$1\cdots1$	not $0\cdots0$

	Single precision	Double precision
Total bits	32	64
Exponent bits	8	11
Fraction bits	23	52
Exponent range	-126...127	-1022...1023
Smallest magnitude	$\approx 10^{-45}$	$\approx 10^{-324}$
Decimal range	$\approx [-10^{38}, 10^{38}]$	$\approx [-10^{308}, 10^{308}]$
Decimal precision	7	16

Rounding, overflow, underflow



Let's try to represent 0.1 in floating-point

- Fraction is 0.00011001100110011001100...
- No exact representation possible

Errors can grow and dominate results

Problem often happens in practice

Source of arithmetic errors

Addition may not be exact even when exponents are equal

- $1.1010 + 1.0101 = 1.01111 \times 2^1 \rightarrow 1.1000 \times 2^1$

Source of arithmetic errors

Addition may not be exact even when exponents are equal

- $1.1010 + 1.0101 = 1.01111 \times 2^1 \to 1.1000 \times 2^1$

Trouble when exponents differ

- Must *pre-shift* to match exponents
- $1.0000 + 1.0000 \times 2^{-5} = 1.00001 \to 1.0000$
- Frequent source of rounding errors

Source of arithmetic errors

Addition may not be exact even when exponents are equal

- $1.1010 + 1.0101 = 1.01111 \times 2^1 \to 1.1000 \times 2^1$

Trouble when exponents differ

- Must *pre-shift* to match exponents
- $1.0000 + 1.0000 \times 2^{-5} = 1.00001 \to 1.0000$
- Frequent source of rounding errors

Multiplication is even worse

- Even with matching exponents, needs double number of bits!

Other weirdness

Associative property does not hold!

- $(a + b) + c \neq a + (b + c)$
- Beware of compiler optimizations

Other weirdness

Associative property does not hold!

- $(a + b) + c \neq a + (b + c)$
- Beware of compiler optimizations

Equality operator is basically useless

- Returns true only when *exactly* equal
- Must use special function

The only guarantee is the following

$$fl(x \odot y) = (x \odot y)(1 + \delta_1), \qquad |\delta_1| \leq u = 2^{-t}$$

$$fl(x \odot y) = \frac{x \odot y}{1 + \delta_2}, \qquad |\delta_2| \leq u$$

$$\odot = +, -, \times, \div, \sqrt{}$$

The only guarantee is the following

$$fl(x \odot y) = (x \odot y)(1 + \delta_1), \qquad |\delta_1| \leq u = 2^{-t}$$

$$fl(x \odot y) = \frac{x \odot y}{1 + \delta_2}, \qquad |\delta_2| \leq u$$

$$\odot = +, -, \times, \div, \sqrt{}$$

J.-M. Muller, N. Brisebarre, F. de Dinechin, C.-P. Jeannerod, V. Lefèvre, G. Melquiond, N. Revol, D. Stehlé, and S. Torres. *Handbook of floating-point arithmetic.* Birkhäuser, 2010

N. J. Higham. *Accuracy and stability of numerical algorithms.* SIAM, 2nd edition, 2002

Examples of possible problems

How can you even compare two numbers for equality?

Examples of possible problems

How can you even compare two numbers for equality?

Problem with the Pythagoras formula $\sqrt{x^2 + y^2}$

Examples of possible problems

How can you even compare two numbers for equality?

Problem with the Pythagoras formula $\sqrt{x^2 + y^2}$

- What if x or y too small or too large?

Examples of possible problems

How can you even compare two numbers for equality?

Problem with the Pythagoras formula $\sqrt{x^2 + y^2}$

- What if x or y too small or too large?
- Solution? (See the `hypot` function.)

Examples of possible problems

How can you even compare two numbers for equality?

Problem with the Pythagoras formula $\sqrt{x^2 + y^2}$

- What if x or y too small or too large?
- Solution? (See the `hypot` function.)

Problems with the quadratic formula

Examples of possible problems

How can you even compare two numbers for equality?

Problem with the Pythagoras formula $\sqrt{x^2 + y^2}$

- What if x or y too small or too large?
- Solution? (See the `hypot` function.)

Problems with the quadratic formula

- What if $\Delta \approx 0$?

Examples of possible problems

How can you even compare two numbers for equality?

Problem with the Pythagoras formula $\sqrt{x^2 + y^2}$

- What if x or y too small or too large?
- Solution? (See the `hypot` function.)

Problems with the quadratic formula

- What if $\Delta \approx 0$?
- What if $a \approx 0$? Interpretation?

How can you even compare two numbers for equality?

Problem with the Pythagoras formula $\sqrt{x^2 + y^2}$

- What if x or y too small or too large?
- Solution? (See the `hypot` function.)

Problems with the quadratic formula

- What if $\Delta \approx 0$?
- What if $a \approx 0$? Interpretation?
- Solution?

Examples of possible problems

How can you even compare two numbers for equality?

Problem with the Pythagoras formula $\sqrt{x^2 + y^2}$

- What if x or y too small or too large?
- Solution? (See the `hypot` function.)

Problems with the quadratic formula

- What if $\Delta \approx 0$?
- What if $a \approx 0$? Interpretation?
- Solution?

J. F. Blinn. How to solve a quadratic equation. *IEEE Computer Graphics and Applications*, 25(6):76–79, 2005

Solve $f(x) = 0$ for x

Iterative root-finding methods

Solve $f(x) = 0$ for x

Bisection

- Given an interval $[a, b]$ with $f(a)f(b) \leq 0$ and f continuous
- Guaranteed linear convergence

Solve $f(x) = 0$ for x

Bisection

- Given an interval $[a, b]$ with $f(a)f(b) \leq 0$ and f continuous
- Guaranteed linear convergence

Newton-Raphson

- Given f differentiable and given t_0 "near" root
- Quadratic convergence, but no guarantees

Iterative root-finding methods

Solve $f(x) = 0$ for x

Bisection

- Given an interval $[a, b]$ with $f(a)f(b) \leq 0$ and f continuous
- Guaranteed linear convergence

Newton-Raphson

- Given f differentiable and given t_0 "near" root
- Quadratic convergence, but no guarantees

Safe Newton-Raphson

- Combines advantages of both methods

Polynomial roots

Very simple method for finding roots of polynomial $p(x) = 0, x \in [a, b]$

Polynomial roots

Very simple method for finding roots of polynomial $p(x) = 0, x \in [a, b]$

- Solve for roots of $p'(x) = 0$ recursively
- Use roots of $p'(x)$ to isolate roots of $p(x)$ into brackets
- Use safe Newton-Raphson to refine one of $p(x)$ in each bracket

Polynomial roots

Very simple method for finding roots of polynomial $p(x) = 0, x \in [a, b]$

- Solve for roots of $p'(x) = 0$ recursively
- Use roots of $p'(x)$ to isolate roots of $p(x)$ into brackets
- Use safe Newton-Raphson to refine one of $p(x)$ in each bracket

What about in Bernstein basis?

References

J. F. Blinn. How to solve a quadratic equation. *IEEE Computer Graphics and Applications*, 25(6):76–79, 2005.

N. J. Higham. *Accuracy and stability of numerical algorithms*. SIAM, 2nd edition, 2002.

J.-M. Muller, N. Brisebarre, F. de Dinechin, C.-P. Jeannerod, V. Lefèvre, G. Melquiond, N. Revol, D. Stehlé, and S. Torres. *Handbook of floating-point arithmetic*. Birkhäuser, 2010.