Previous Lecture:
Examples on vectors and simulation

Today’s Lecture:
Finite vs. Infinite; Discrete vs. Continuous
Vectors and vectorized code
Color computation with linear interpolation
plot and fill

Announcements:
Project 3 due Thursday 3/5 at 11pm
Prelim 1 on March 10th at 7:30pm. Review questions and old exams have been posted
Optional review session on Sunday 3/8, 1:30-3pm, Kimball B11

---

Xeno’s Paradox

- A wall is two feet away
- Take steps that repeatedly halve the remaining distance
- You never reach the wall because the distance traveled after n steps =
  \[ 1 + \frac{1}{2} + \frac{1}{4} + \ldots + \frac{1}{2^n} = 2 - \frac{1}{2^n} \]

---

Example: “Xeno” disks

What do you need to keep track of?
- Diameter (d)
- Position
- Left tangent point (x)

<table>
<thead>
<tr>
<th>Disk</th>
<th>x</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0+1</td>
<td>1/2</td>
</tr>
<tr>
<td>3</td>
<td>0+1+1/2</td>
<td>1/4</td>
</tr>
</tbody>
</table>

% Xeno Disks

```
DrawRect(0,-1,2,2,'k')
% Draw 20 Xeno disks
d= 1;
x= 0;  % Left tangent point
for k= 1:20
    % Draw kth disk
    % Update x, d for next disk
end
```
Here’s the output… Shouldn’t there be 20 disks?

The “screen” is an array of dots called pixels.

Disks smaller than the dots don’t show up.

The 20th disk has radius<.00001

---

Fading Xeno disks

- First disk is yellow
- Last disk is black (invisible)
- Interpolate the color in between

---

Color is a 3-vector, sometimes called the RGB values

- Any color is a mix of red, green, and blue
- Example: \( \text{colr} = [0.4 \ 0.6 \ 0] \)
- Each component is a real value in \([0,1]\)
- \([0 \ 0 \ 0]\) is black
- \([1 \ 1 \ 1]\) is white

---

Example: 3 disks fading from yellow to black

\[
\begin{align*}
\text{r} &= 1; & \text{radius of disk} \\
\text{yellow} &= [1 \ 1 \ 0]; & \text{blue} = [0 \ 0 \ 0]; \\
\text{black} &= [0 \ 0 \ 0]; \\
\text{Left disk yellow, at x=1} & \quad \text{DrawDisk}(1,0,r,\text{yellow}) \\
\text{Right disk black, at x=5} & \quad \text{DrawDisk}(5,0,r,\text{black}) \\
\text{Middle disk with average color, at x=3} & \quad \text{colr} = 0.5*\text{yellow} + 0.5*\text{black}; \\
& \quad \text{DrawDisk}(3,0,r,\text{colr})
\end{align*}
\]

---

Example: 3 disks fading from yellow to black

\[
\begin{align*}
\text{r} &= 1; & \text{radius of disk} \\
\text{yellow} &= [1 \ 1 \ 0]; & \text{black} = [0 \ 0 \ 0]; \\
\text{Left disk yellow, at x=1} & \quad \text{DrawDisk}(1,0,r,\text{yellow}) \\
\text{Right disk black, at x=5} & \quad \text{DrawDisk}(5,0,r,\text{black}) \\
\text{Middle disk with average color, at x=3} & \quad \text{colr} = 0.5*\text{yellow} + 0.5*\text{black}; \\
& \quad \text{DrawDisk}(3,0,r,\text{colr})
\end{align*}
\]
Vectorized code allows an operation on multiple values at the same time.

\[
\begin{align*}
\text{yellow} &= [1 \ 1 \ 0]; \\
\text{black} &= [0 \ 0 \ 0]; \\
\text{colr} &= 0.5 \times \text{yellow} + 0.5 \times \text{black};
\end{align*}
\]

Average color via vectorized op
Operation performed on vectors

Average color via scalar op
Operation performed on scalars

Use linear interpolation to obtain the colors. Each disk has a color that is a linear combination of yellow and black. Let \( f \) be a fraction in \((0,1)\) ...

\[
\begin{align*}
f &= \text{???} \\
\text{colr} &= f \times \text{black} + (1-f) \times \text{yellow};
\end{align*}
\]

Rows of Xeno disks

\[
\text{for } y = \_ : \_ : \_ \\
\text{Code to draw one row of Xeno disks at some } y\text{-coordinate}
\]

Be careful with "initializations"
Does this script print anything?

```matlab
k = 0;
while 1 + 1/2^k > 1
    k = k+1;
end
disp(k)
```

Computer Arithmetic—floating point arithmetic

Suppose you have a calculator with a window like this:

```
+ 2 4 1 - 3
```

representing $2.41 \times 10^{-3}$

Floating point addition

```
+ 2 4 1 - 3
+ 1 0 0 - 6
```

Result:

```
+ 2 4 1 - 3
```

Floating point addition

```
+ 2 4 1 - 3
+ 1 0 0 - 6
```

Result:

```
+ 2 4 1 - 3
```

The loop DOES terminate given the limitations of floating point arithmetic!

```matlab
k = 0;
while 1 + 1/2^k > 1
    k = k+1;
end
disp(k)
```

$1 + 1/2^{53}$ is calculated to be just 1, so “53” is printed.

Patriot missile failure

In 1991, a Patriot Missile failed, resulting in 28 deaths and about 100 injured. The cause?

```
0.1
```
Inexact representation of time/number

- System clock represented time in tenths of a second: a clock tick every $1/10$ of a second
- Time = number of clock ticks $\times 0.1$

```
0.00011001100110011001100110011
```

Value in Patriot system

```
0.0001100110011001100110011
```

Error of $0.00000095$ every clock tick

Resulting error

... after 100 hours

\[ 0.00000095 \times (100 \times 60 \times 60) \]

0.34 second

At a velocity of 1700 m/s, missed target by more than 500 meters!

Computer arithmetic is inexact

- There is error in computer arithmetic—floating point arithmetic—due to limitation in “hardware.” Computer memory is finite.
- What is $1 + 10^{-16}$?
  - $1.0000000000000001$ in real arithmetic
  - $1$ in floating point arithmetic (IEEE)
- Read Sec 4.3

Vectorized code

- A Matlab-specific feature

- Code that performs element-by-element arithmetic RELATIONAL/logical operations on array operands in one step
- Scalar operation: $x + y$
  where $x, y$ are scalar variables
- Vectorized code: $x + y$
  where $x$ and/or $y$ are vectors. If $x$ and $y$ are both vectors, they must be of the same shape and length

Vectorized addition

```
2 1.5 8
```

```
1 2 0 1
```

```
3 3.5 9
```

Matlab code: $z = x + y$

Vectorized subtraction

```
2 1.5 8
```

```
1 2 0 1
```

```
1 -1.5 7
```

Matlab code: $z = x - y$
Vectorized multiplication

\[
\begin{array}{c}
a & 2 & 1.5 & 8 \\
x & b & 1 & 2 & 0 & 1 \\
\hline \\
\end{array}
\]

\[c = a \times b = 2 & 2 & 0 & 8\]

Matlab code: \(c = a .* b\)

---

Vectorized element-by-element arithmetic operations on arrays

\[
\begin{array}{c}
+ & - & \times & \div \\
\hline \\
+ & - & \times & \div \\
\hline \\
\hline \\
\end{array}
\]

A dot (.) is necessary in front of these math operators

---

Shift

\[
\begin{array}{c}
x & 3 \\
+ & y & 2 & 1.5 & 8 \\
\hline \\
\end{array}
\]

\[z = 5 & 4 & 3.5 & 11\]

Matlab code: \(z = x + y\)

---

Reciprocate

\[
\begin{array}{c}
x & 1 \\
/ & y & 2 & 1.5 & 8 \\
\hline \\
\end{array}
\]

\[z = 5 & 1 & 2.125\]

Matlab code: \(z = x ./ y\)

---

Element-by-element arithmetic operations on arrays...

Also called “vectorized code”

\[
x = \text{linspace}(-2,3,200); \quad x \text{ and } y \text{ are vectors} \\
y = \sin(5^*x) .* \exp(-x/2) ./ (1 + x.^2); \\
\]

Contrast with scalar operations that we’ve used previously...

\[a = 2.1; \quad a \text{ and } b \text{ are scalars} \\
b = \sin(5*a); \quad \text{The operators are (mostly) the same: the operands may be scalars or vectors.} \]

When an operand is a vector, you have “vectorized code.”
Drawing a polygon (multiple line segments)

% Draw a rectangle with the lower-left corner at (a,b), width w, height h.
x = [a  a+w  a+w  a  a ]; % x data
y = [b  b  b+h  b+h  b ]; % y data
plot(x, y)

Fill in the missing vector values!

Coloring a polygon (fill)

% Draw a rectangle with the lower-left corner at (a,b), width w, height h, % and fill it with a color named by c.
x = [a  a+w  a+w  a  a ]; % x data
y = [b  b  b+h  b+h  b ]; % y data
fill(x, y, c)

Built-in function fill actually does the "wrap-around" automatically.

x = [0.1 -9.2 -7 4.4];
y = [9.4 7 -6.2 -3];
fill(x,y,'g')

Can be a vector (RGB values)