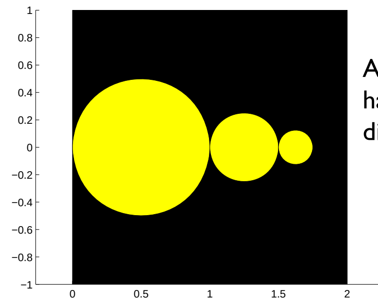
- Previous Lecture:
  - Examples on vectors and simulation
- Today's Lecture:
  - Finite vs. Infinite; Discrete vs. Continuous
  - Vectors and vectorized code
  - Color computation with *linear interpolation*
  - `plot` and `fill`
- Announcements:
  - Project 3 due Friday 10/3 at 11pm
  - Prelim 1 on Oct 16th at 7:30pm. Email Randy Hess (rbh27) now if you have an exam conflict (specify conflicting course and instructor contact info)

Screen Granularity



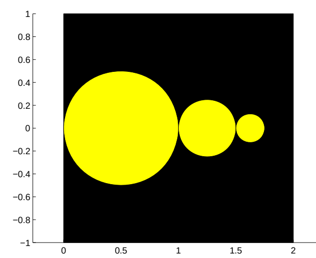
After how many halvings will the disks disappear?

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 =

$$1 + \frac{1}{2} + \frac{1}{4} + \ldots + 1/2^n = 2 - 1/2^n$$

Example: "Xeno" disks

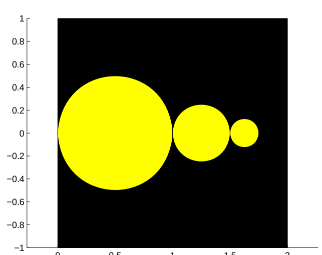


Draw a sequence of 20 disks where the (k+1)th disk has a diameter that is half that of the kth disk.

The disks are tangent to each other and have centers on the x-axis.

First disk has diameter 1 and center (1/2, 0).

Example: "Xeno" disks



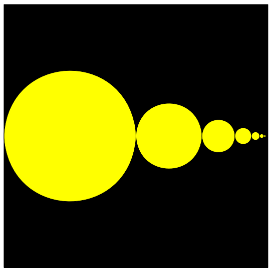
What do you need to keep track of?

- Diameter (d)
- Position
  Left tangent point (x)

Disk	x	d
1	0	1
2	0+1	1/2
3	0+1+1/2	1/4


```
% 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?

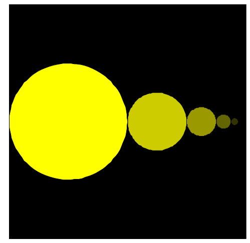


The "screen" is an array of dots called pixels.

Disks smaller than the dots don't show up.

The 20th disk has radius<.000001

Fading Xeno disks

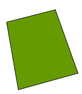


- First disk is yellow
- Last disk is black (invisible)
- *Interpolate* the color in between

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

- Any color is a mix of red, green, and blue
- Example:

```
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


```
% Draw n fading Xeno disks
d= 1;
x= 0;  % Left tangent point
yellow= [1 1 0];
black=  [0 0 0];
for k= 1:n
   % Compute color of kth disk


   % Draw kth disk
   DrawDisk(x+d/2, 0, d/2, _______)
   x= x+d;
   d= d/2;
end
```

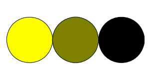

Example: 3 disks fading from yellow to black

```
r= 1;  % radius of  disk
yellow= [1 1 0];
black = [0 0 0];

% Left disk yellow, at x=1
DrawDisk(1,0,r,yellow)
% Right disk black, at x=5
DrawDisk(5,0,r,black)

% Middle disk with average color, at x=3
colr= 0.5*yellow + 0.5*black;
DrawDisk(3,0,r,colr)
```

.5 * [1 1 0] → [.5 .5 0]

.5 * [0 0 0] → [0 0 0]

Vectorized multiplication



Vectorized code allows an operation on multiple values at the same time

```
yellow= [1 1 0];
black = [0 0 0];
```

Vectorized addition

[.5 .5 0]
\+ [0 0 0]
= [.5 .5 0]

```
% Average color via vectorized op
colr= 0.5*yellow + 0.5*black;
```

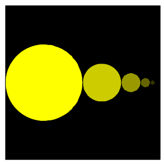
Operation performed on vectors

```
% Average color via scalar op
for k = 1:length(black)
   colr(k)= 0.5*yellow(k) + 0.5*black(k);
end
```

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) …

```
f= ???
colr= f*black + (1-f)*yellow;
```



Linear interpolation

x	g(x)
:	:
9	110
10	118
10.25	?
10.50	?
10.75	?
11	126
12	134
:	:

g(10.5) = $\frac{1}{2}$ g(11) + $\frac{1}{2}$ g(10)

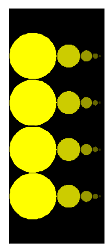
g(10) = 0/4·g(11) + 4/4·g(10)
g(10.25) = 1/4·g(11) + 3/4·g(10)
g(10.50) = 2/4·g(11) + 2/4·g(10)
g(10.75) = 3/4·g(11) + 1/4·g(10)
g(11) = 4/4·g(11) + 0/4·g(10)

f · g(11) + (1-f) · g(10)


```
% Draw n fading Xeno disks
d= 1;
x= 0;  % Left tangent point
yellow= [1 1 0];
black=  [0 0 0];
for k= 1:n
   % Compute color of kth disk
   f= ???
   colr= f*black + (1-f)*yellow;
   % Draw kth disk
   DrawDisk(x+d/2, 0, d/2, colr)
   x= x+d;
   d= d/2;
end
```

A k/n
B k/(n-1)
C (k-1)/n
D (k-1)/(n-1)
E (k-1)/(n+1)

Rows of Xeno disks



for y = __ : __ : __

Code to draw one row of Xeno disks at some y-coordinate

end

Be careful with "initializations"

Does this script print anything?

```
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 x 10^{-3}

Floating point addition

+	2	4	1	-	3
+	1	0	0	-	3

Result:

+	3	4	1	-	3

41

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

```
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.

46

Patriot missile failure

www.namsa.nato.int/gallery/systems

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

0.1

Lecture 12 47

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 x 0.1

"exact" value

.00011001100110011001100110011…

.00011001100110011001100110011 value in Patriot system

Error of .000000095 every clock tick

Lecture 12 49

Resulting error

… after 100 hours

.000000095 x (100x60x60)

0.34 second

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

Lecture 12 50

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

Lecture 12 51

Vectorized code
—a Matlab-specific feature

See Sec 4.1 for list of vectorized arithmetic operations

- 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

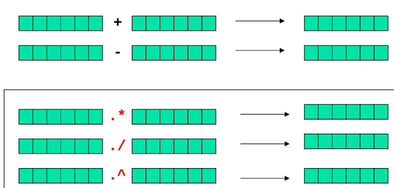
Vectorized multiplication

	a	2	1	.5	8
x	b	1	2	0	1
=	c	2	2	0	8

Matlab code: `c= a .* b`

Vectorized element-by-element arithmetic operations on arrays

See full list of ops in §4.1



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

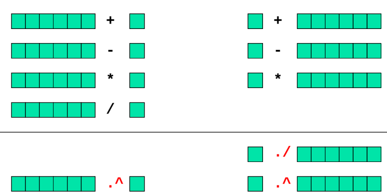
Reciprocate

	x	1			
/	y	2	1	.5	8
=	z	.5	1	2	.125

Matlab code: `z= x ./ y`

Vectorized element-by-element arithmetic operations between an array and a scalar

See full list of ops in §4.1



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

The dot in .* , .* , ./ not necessary but OK

Element-by-element arithmetic operations on arrays… Also called "vectorized code"

```
x = linspace(-2,3,200);
y = sin(5*x).*exp(-x/2)./(1 + x.^2);
```

x and y are vectors

Contrast with scalar operations that we've used previously…

```
a = 2.1;
b = sin(5*a);
```

a and b are scalars

The operators are (mostly) the same; the operands may be scalars or vectors.

When an operand is a vector, you have "vectorized code."