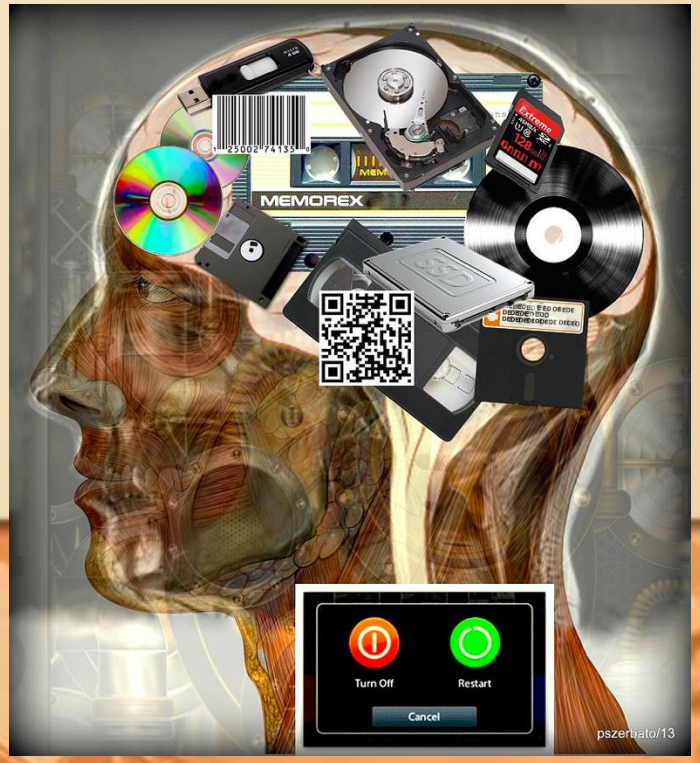
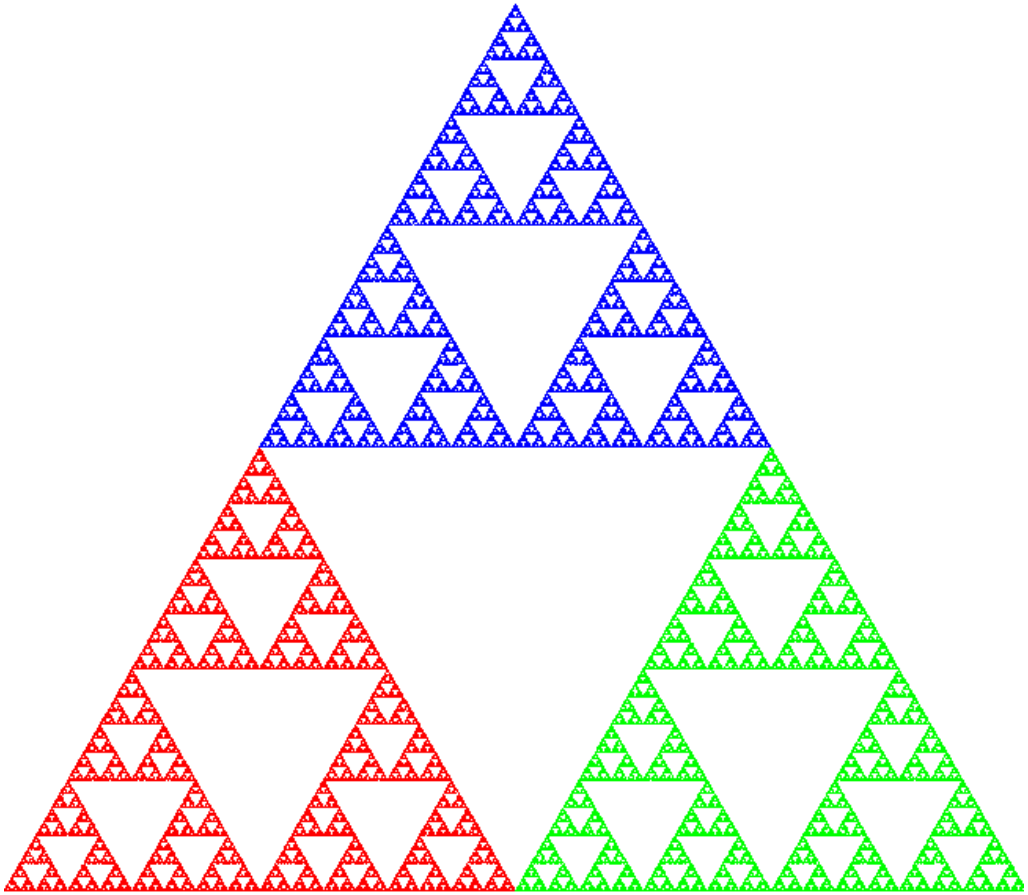
Lecture 7-1

Multimedia, Part II



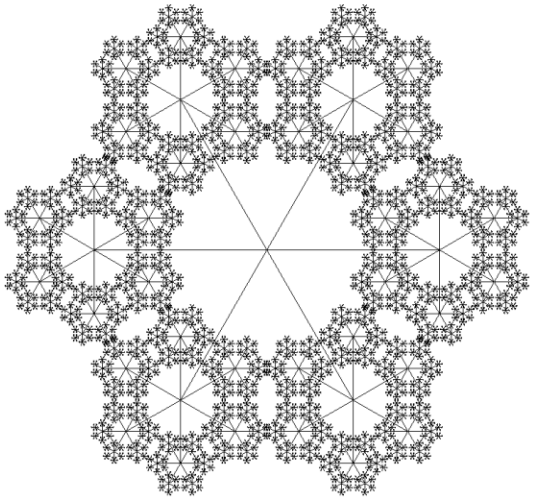
Lecture plan

- Graphics
- Animation
- Sound

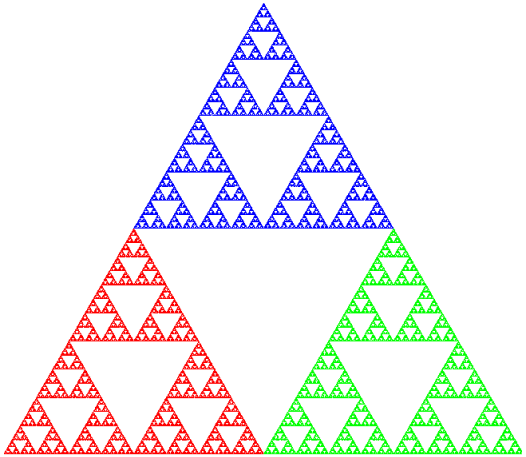


Fractals

Self-replicating patterns:



Fractal: The deeper you dive into the image, the more you see the same image.



Graphics: StdDraw API (partial)

```
static void  circle(double x, double y, double radius)
static void  clear()
static void  enableDoubleBuffering()
static void  filledCircle(x, y, radius);
static Color getBackgroundColor()
static Color getPenColor()
static void  line(double x0, double y0, double x1, double y1)
static void  pause(int t)
static void  picture(double x, double y, String filename)
static void  point(double x, double y)
static void  rectangle(double x, double y, double halfWidth, double halfHeight)
static void  save(String filename)
static void  setCanvasSize(int canvasWidth, int canvasHeight)
static void  setPenColor(int red, int green, int blue)
static void  setPenColor(Color color)
static void  setPenRadius(double radius)
static void  setTitle(String title)
static void  setXscale(double min, double max)
static void  setYscale(double min, double max)
static void  show()
static void  square(double x, double y, double halfLength)
```

Complete API (click)

Graphics: StdDraw API (partial)

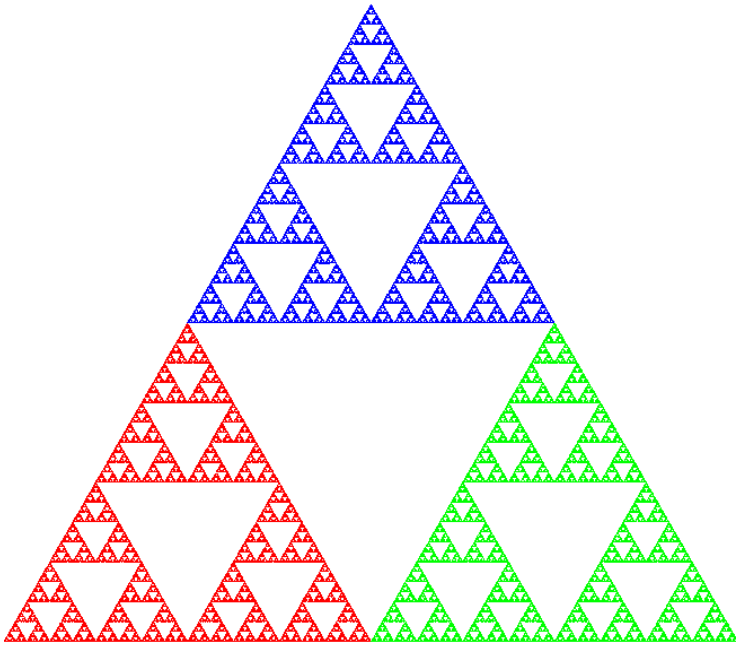
```
static void  circle(double x, double y, double radius)
static void  clear()
static void  enableDoubleBuffering()
static void  filledCircle(x, y, radius);
static Color getBackgroundColor()
static Color getPenColor()
static void  line(double x0, double y0, double x1, double y1)
static void  pause(int t)
static void  picture(double x, double y, String filename)
static void  point(double x, double y)
static void  rectangle(double x, double y, double halfWidth, double halfHeight)
static void  save(String filename)
static void  setCanvasSize(int canvasWidth, int canvasHeight)
static void  setPenColor(int red, int green, int blue)
static void  setPenColor(Color color)
static void  setPenRadius(double radius)
static void  setTitle(String title)
static void  setXscale(double min, double max)
static void  setYscale(double min, double max)
static void  show()
static void  square(double x, double y, double halfLength)
```

Complete API (click)

For the next example we'll need only the `point` drawing function

Fractals

Sierpinski triangle:



Sierpinski triangle:

Algorithm

Construct an equilateral triangle with vertices R, G, B;

currentPoint = R

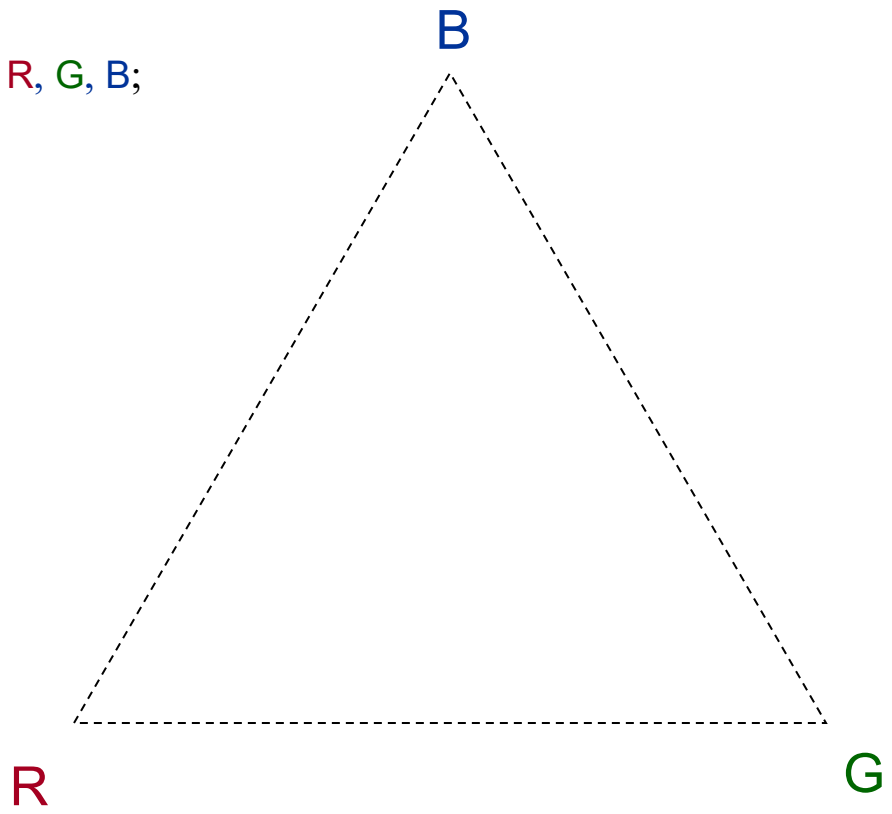
repeat N times:

target = random point (R, G, or B)

compute the halfway point between the currentPoint and the target

draw this point using the chosen color

currentPoint = target



Sierpinski triangle:

Algorithm

Construct an equilateral triangle with vertices R, G, B;

currentPoint = R

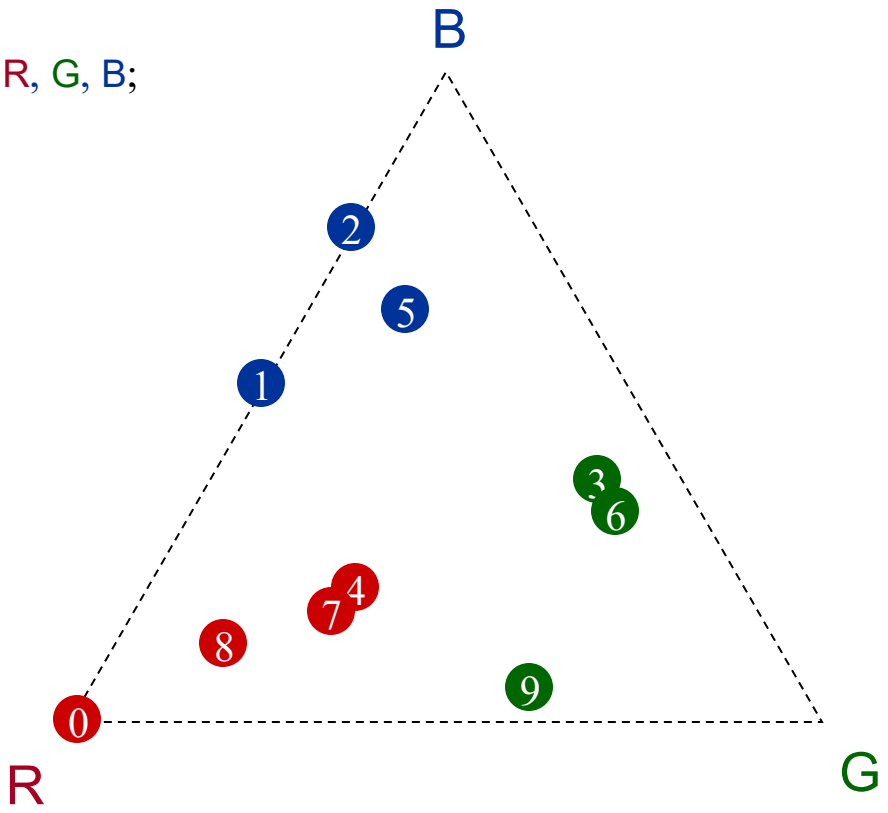
repeat N times:

target = random point (R, G, or B)

compute the halfway point between the currentPoint and the target

draw this point using the chosen color

currentPoint = target



Fractals

Sierpinski triangle:

Algorithm

Construct an equilateral triangle with vertices R, G, B;

currentPoint = R

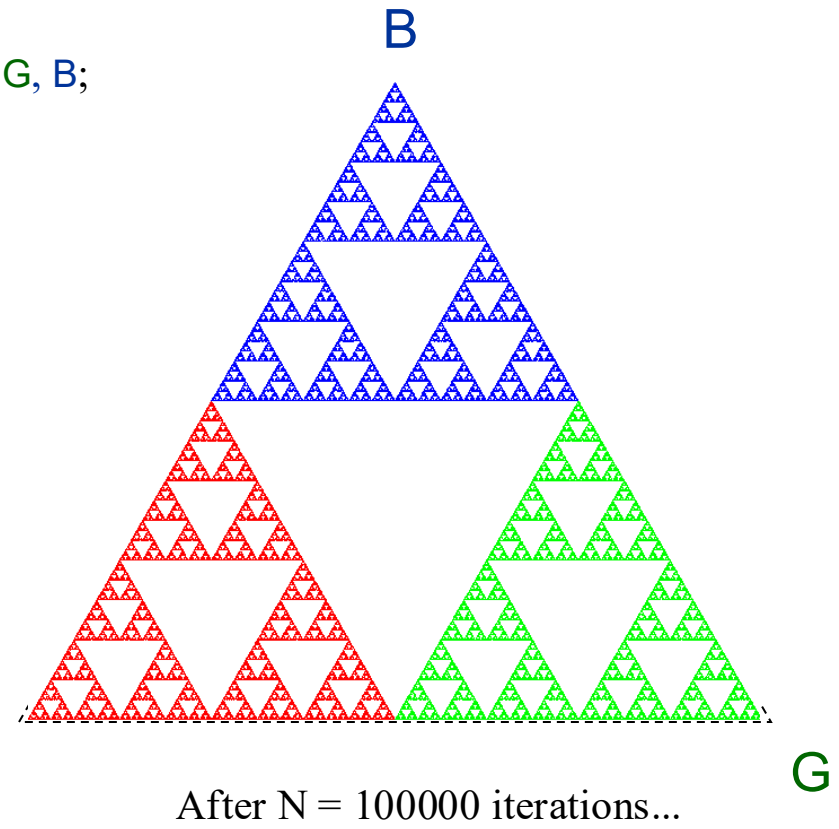
repeat N times:

target = random point (R, G, or B)

compute the halfway point between the currentPoint and the target

draw this point using the chosen color

currentPoint = target



After N = 100000 iterations...

Sierpinski triangle:

Algorithm

Construct an equilateral triangle with vertices `0`, `1`, `2`;

currentPoint = `0`

repeat N times:

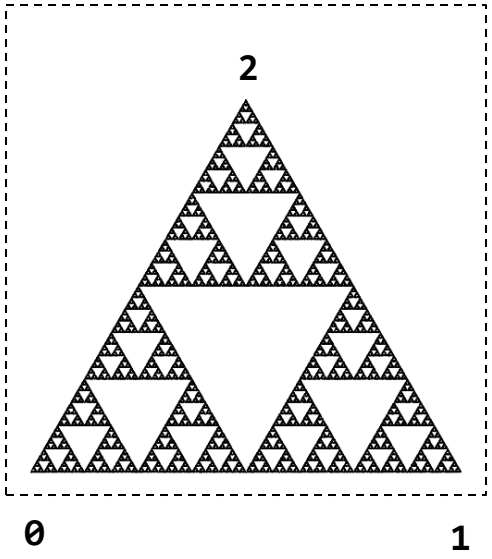
target = random point (`0`, `1`, or `2`)

compute the halfway point between the currentPoint and the target

draw this point using the chosen color

currentPoint = target

```
% java Sierpinski 100000
```



Black and white version

Sierpinski triangle:

Algorithm

Construct an equilateral triangle with vertices `0`, `1`, `2`;

currentPoint = `0`

repeat N times:

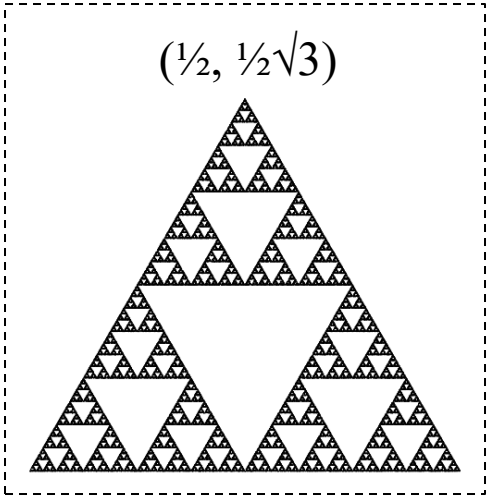
target = random point (`0`, `1`, or `2`)

compute the halfway point between the currentPoint and the target

draw this point using the chosen color

currentPoint = target

```
% java Sierpinski 100000
```



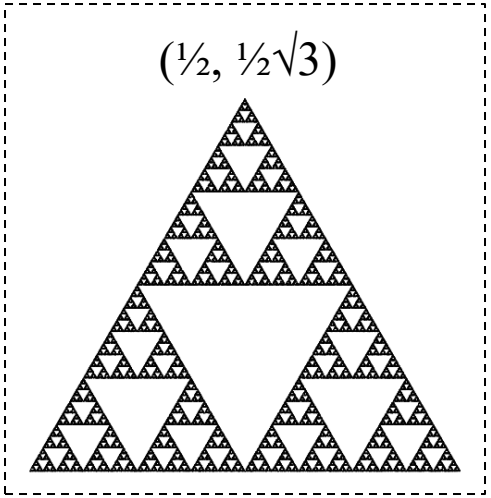

```
xVertex = { 0.0, 1.0, 0.5   };
yVertex = { 0.0, 0.0, 0.866 };
          0   1   2
```

Fractals

Sierpinski triangle:

```
public class Sierpinski {
   public static void main(String[] args) {
      // Gets the number of iterations
      int N = Integer.parseInt(args[0]);

      // Sets the coordinates of the triangle's vertices
      double[] xVertex = { 0.0, 1.0, 0.5   };
      double[] yVertex = { 0.0, 0.0, 0.866 };

      double x = 0.0, y = 0.0;

      for (int i = 0; i < N; i++) {
         // Chooses a random index from (0, 1, 2),
         // and draws the midpoint
         int r = (int) (Math.random() * 3);
         x = (x + xVertex[r]) / 2.0;
         y = (y + yVertex[r]) / 2.0;
         StdDraw.point(x, y);
      }
   }
}
```

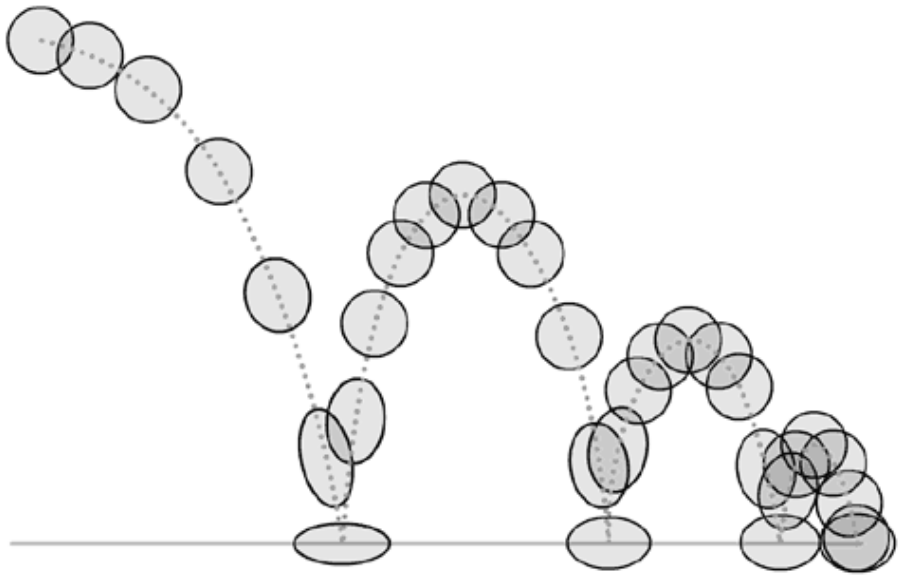
```
% java Sierpinski 100000
```




```
xVertex = { 0.0, 1.0, 0.5   };
yVertex = { 0.0, 0.0, 0.866 };
          0   1   2
```

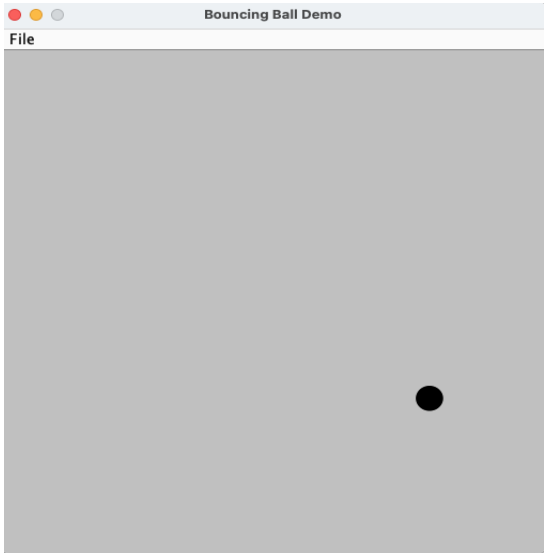
Lecture plan

- Graphics
- Animation
- Sound



Bouncing ball

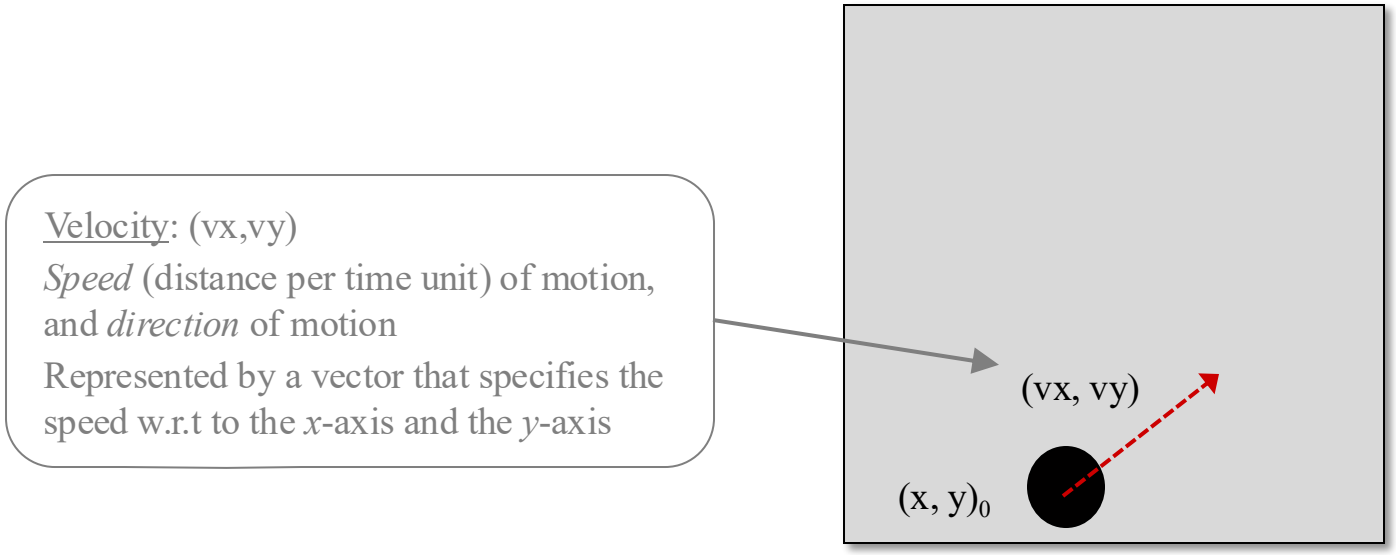
```
% java BouncingBall
```



Bouncing ball

Bouncing ball physics

- At any given time, the ball has a *position* (*x*, *y*) and a *velocity* (*vx*, *vy*)

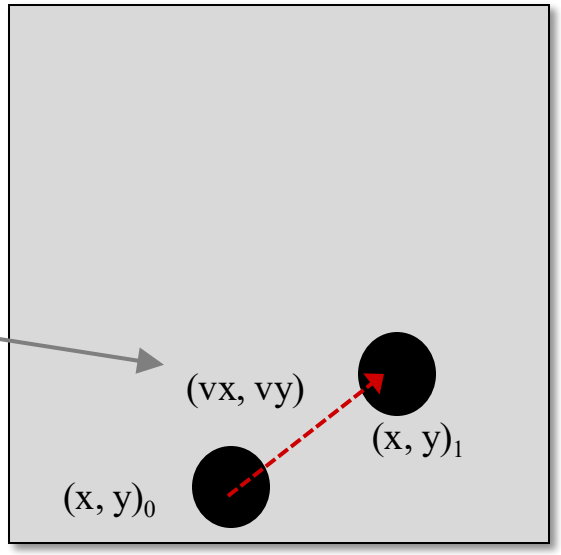


Bouncing ball

Bouncing ball physics

- At any given time, the ball has a *position* (*x*, *y*) and a *velocity* (*vx*, *vy*)
- Assumptions: Constant velocity, no friction, no loss of energy
- To compute the ball's position in the next time unit, we add the ball's velocity to its current position

Velocity: (vx,vy)
Speed (distance per time unit) of motion, and *direction* of motion
Represented by a vector that specifies the speed w.r.t to the *x*-axis and the *y*-axis

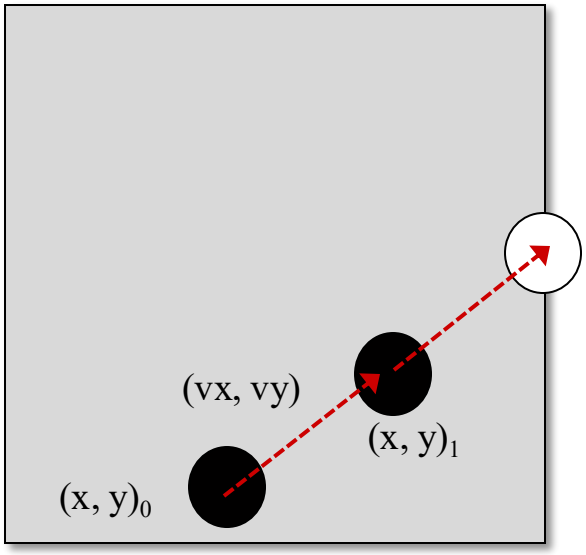


$$x_{t+1} = x_t + vx_t$$
$$y_{t+1} = y_t + vy_t$$

Bouncing ball

Bouncing ball physics

- At any given time, the ball has a *position* (x, y) and a *velocity* (vx, vy)
- Assumptions: Constant velocity, no friction, no loss of energy
- To compute the ball's position in the next time unit, we add the ball's velocity to its current position

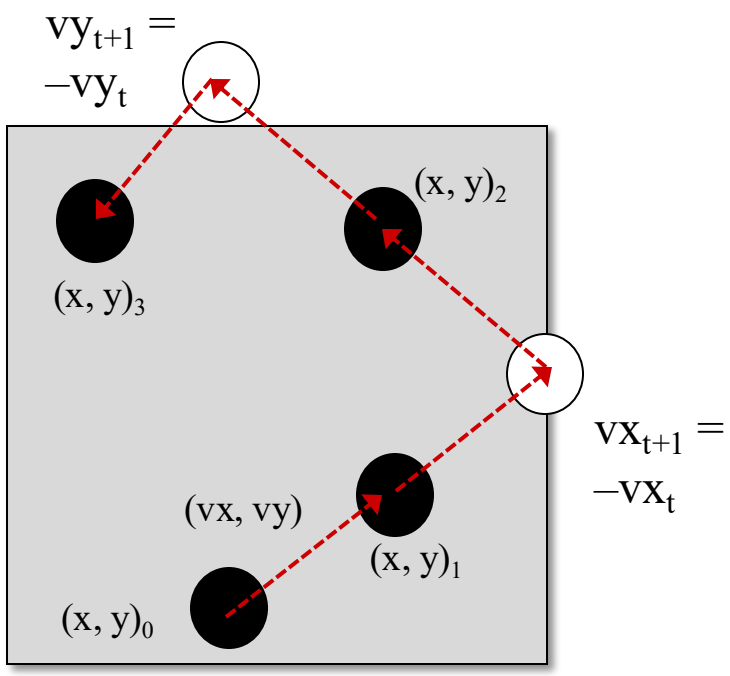


$$x_{t+1} = x_t + vx_t$$
$$y_{t+1} = y_t + vy_t$$

Bouncing ball

Bouncing ball physics

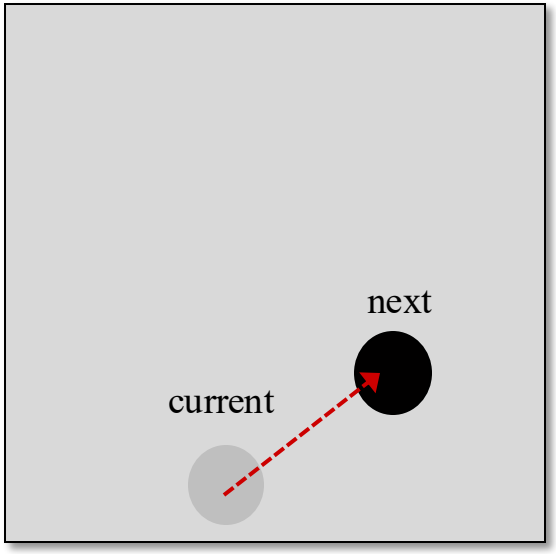
- At any given time, the ball has a *position* (x, y) and a *velocity* (vx, vy)
- Assumptions: Constant velocity, no friction, no loss of energy
- To compute the ball's position in the next time unit, we add the ball's velocity to its current position
- If the ball hits a wall, we update its velocity according to the laws of *elastic collision*:
  - energy (velocity) is conserved
  - angle of collision = angle of reflection



$$x_{t+1} = x_t + vx_t$$
$$y_{t+1} = y_t + vy_t$$

Bouncing ball

Animation technique: Stop Motion



Repeat:

- Get into offscreen mode
- Erase the ball
- Compute the ball's next position
- Draw the ball in its next position
- Show the new image

Graphics: StdDraw API (partial)

```
static void  circle(double x, double y, double radius)
static void  clear()
static void  enableDoubleBuffering()
static void  filledCircle(x, y, radius);
static Color getBackgroundColor()
static Color getPenColor()
static void  line(double x0, double y0, double x1, double y1)
static void  pause(int t)
static void  picture(double x, double y, String filename)
static void  point(double x, double y)
static void  rectangle(double x, double y, double halfWidth, double halfHeight)
static void  save(String filename)
static void  setCanvasSize(int canvasWidth, int canvasHeight)
static void  setPenColor(int red, int green, int blue)
static void  setPenColor(Color color)
static void  setPenRadius(double radius)
static void  setTitle(String title)
static void  setXscale(double min, double max)
static void  setYscale(double min, double max)
static void  show()
static void  square(double x, double y, double halfLength)
```

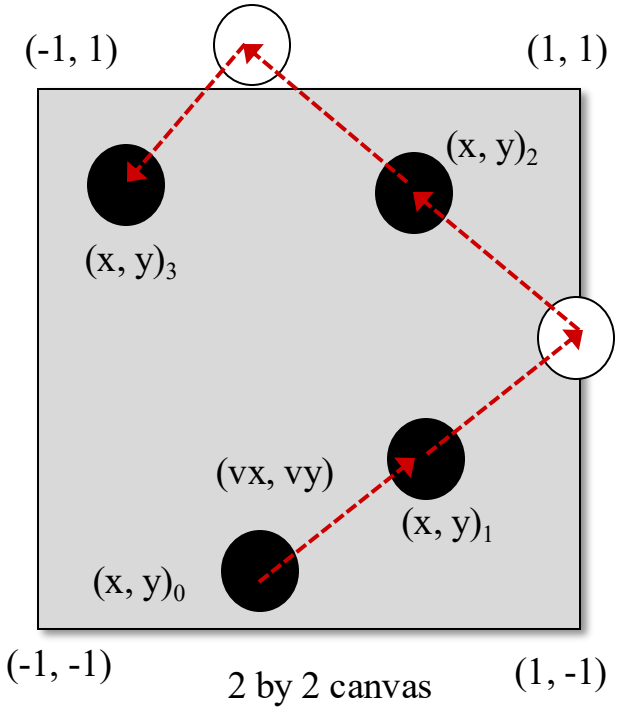
Complete API (click)

We'll need the underlined functions

Bouncing balls

Implementation

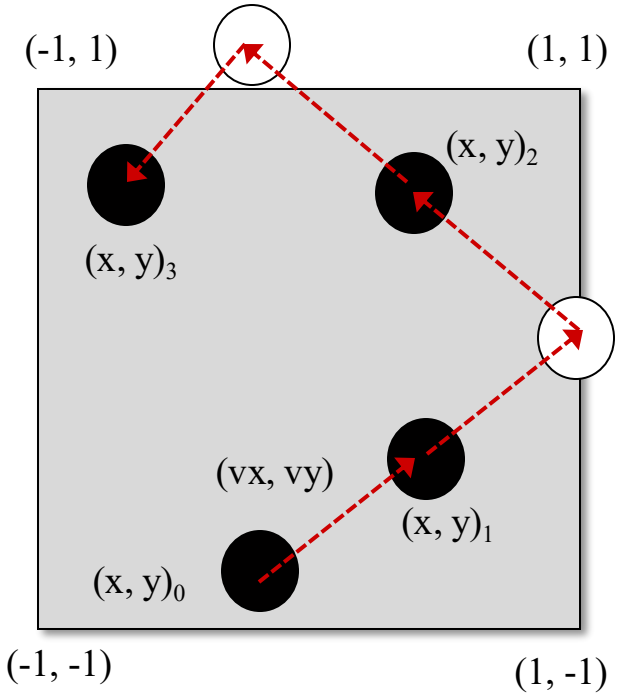
```
public class BouncingBall {
   public static void main(String[] args) {
      // initializes the canvas to be a -1 by -1 square,
      // and enables offscreen drawing.
      StdDraw.setXscale(-1.0, 1.0);
      StdDraw.setYscale(-1.0, 1.0);
      StdDraw.enableDoubleBuffering();

      // sets the ball's radius
      double radius = 0.05;

      // sets the ball's velocity (speed)
      double vx = 0.015, vy = 0.023;

      // initializes the ball's initial position (arbitrary point)
      double x = 0.480, y = 0.860;

      // main animation loop
      while (true)  {
            //// next slide...
```



Bouncing balls

Implementation

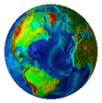
```
public class BouncingBall {
   public static void main(String[] args) {
      // initializes the canvas and the ball's values... (previous slide)
      // main animation loop
      while (true)  {
         // if the ball hits a wall, update the velocity
         // according to law of elastic collision
         if (Math.abs(x + vx) > 1.0 - radius) vx = -vx;
         if (Math.abs(y + vy) > 1.0 - radius) vy = -vy;
         // updates ball's position
         x = x + vx;
         y = y + vy;
         // clears the canvas, and re-draws the ball
         StdDraw.clear(StdDraw.LIGHT_GRAY);
         StdDraw.setPenColor(StdDraw.BLACK);
         StdDraw.filledCircle(x, y, radius);
         // copies offscreen buffer to onscreen buffer, and pauses briefly
         StdDraw.show();
         StdDraw.pause(20);
      }
   }
}
```



Bouncing balls

Special effects:

Instead of drawing a boring sphere, draw some image:



earth.gif

When hitting a wall, make a sound effect:

laser.wav

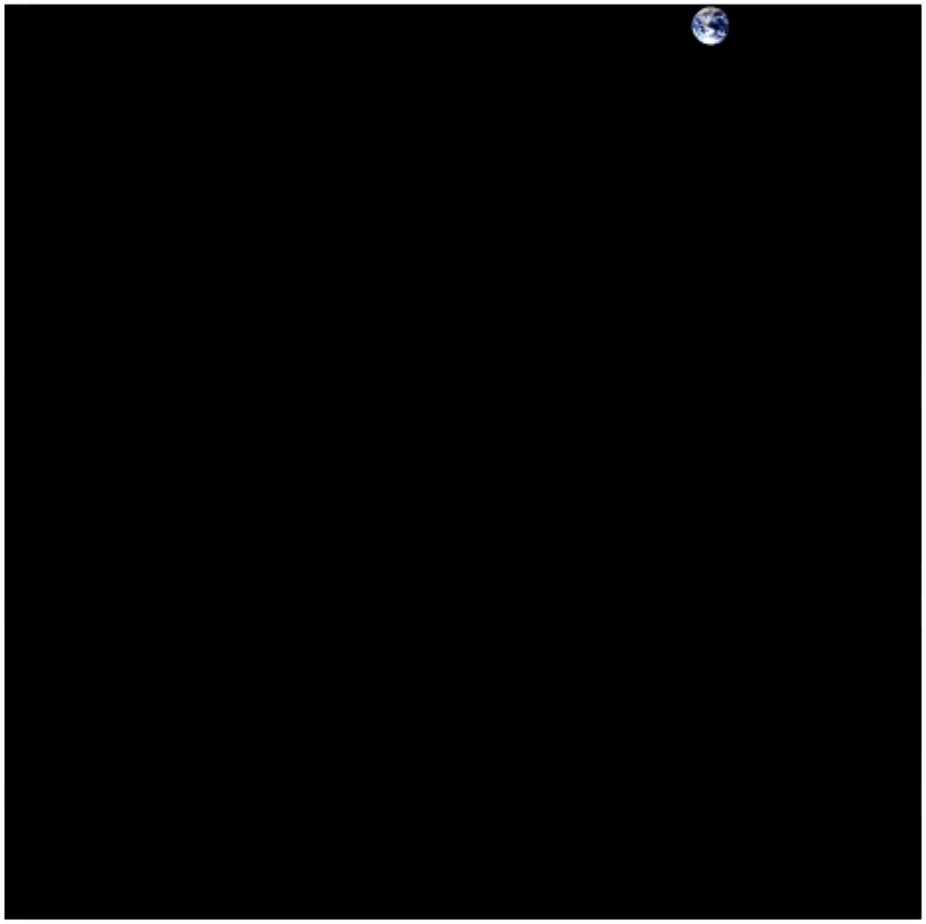
pop.wav

Implementation:

```
StdDraw.picture(x, y, "earth.gif");     // Draws the image at location (x, y)

StdAudio.play("laser.wav");             // Makes sounds
```

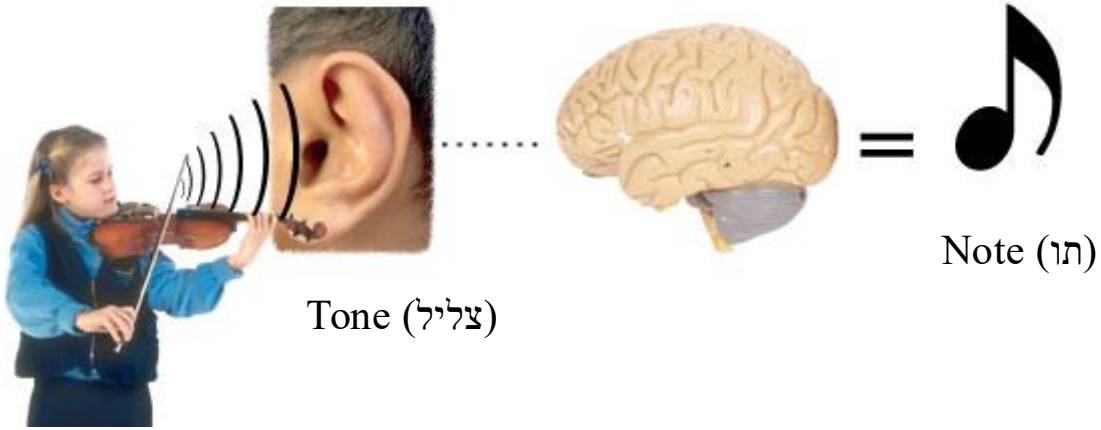
Bouncing balls

Special effects



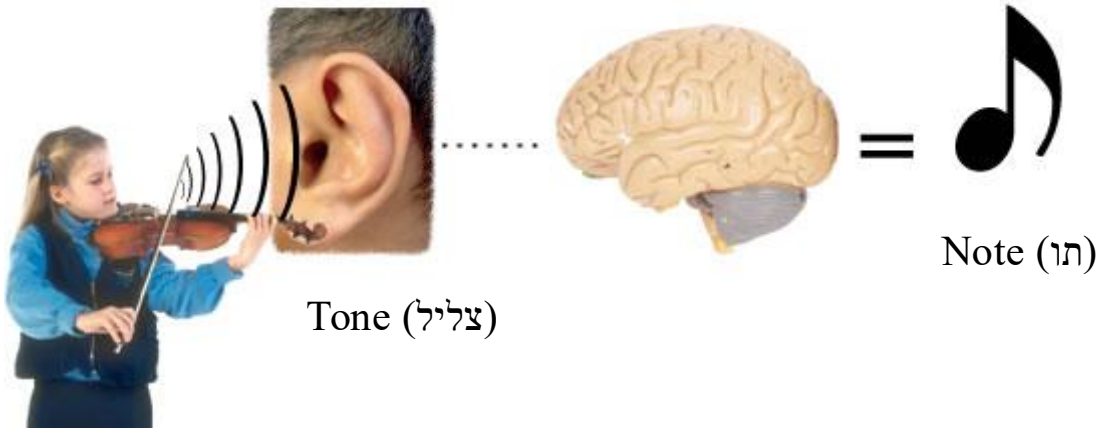
Lecture plan

- Graphics
- Animation
- Sound



Sound

A perception in the brain caused by vibration of molecules hitting the eardrums



Tone (צליל): The sound's *pitch* (wave frequency)

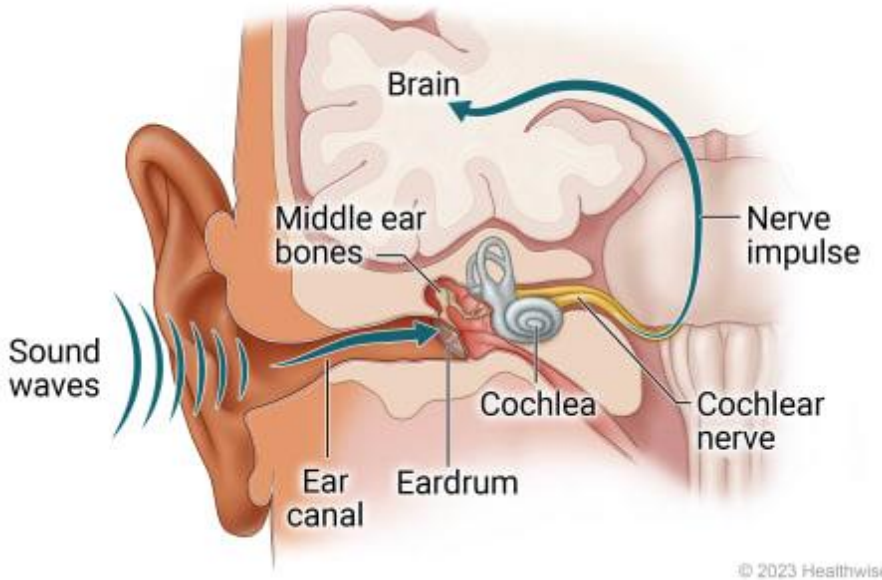
Note (תו): Selected pitch values are called *notes*

Tune (לחן): Combination of notes that sounds "right" to our brain.

Sound crash course

lower frequency,
lower pitch

higher frequency,
higher pitch



Pitch = frequency

(measured in Herz,
number of cycles per second)

<u>Audible sensitivity:</u>

humans: 20 - 20,000 Hz

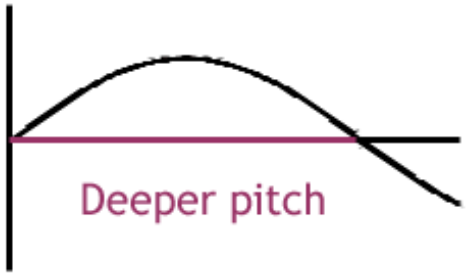
dogs: 67 – 45,000 Hz

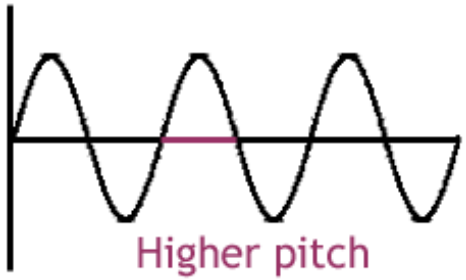
mice: 1 – 70,000 Hz

bats: 1 – 200,000 Hz

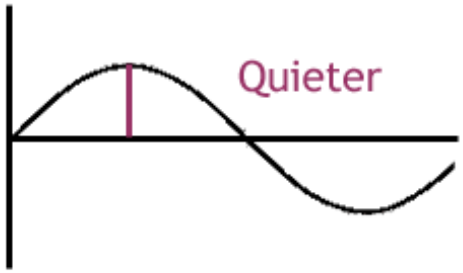
Sound

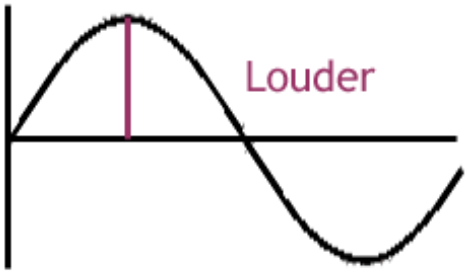
Can be modeled using a $sin(x)$ function with two properties:





Pitch

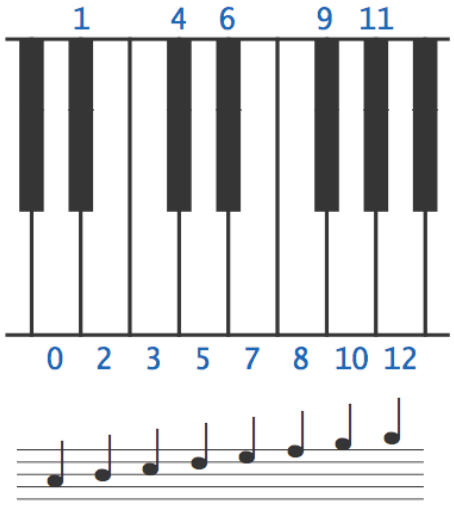




volume

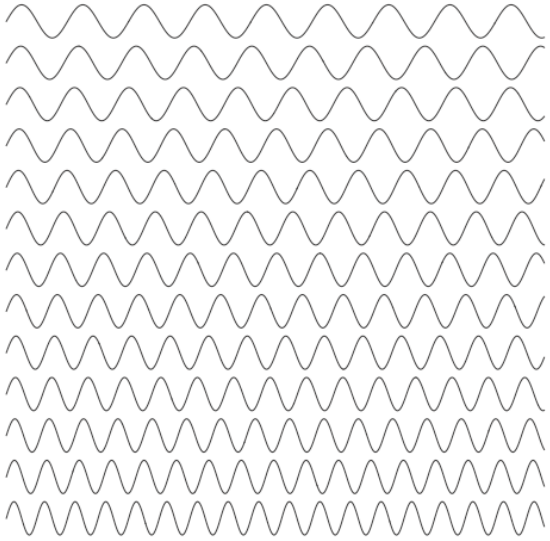
Chromatic scale

Selected 12 notes from 440 Hz to 880 Hz:



note	*i*	*frequency*
A	0	440.00
A♯ *or* B♭	1	466.16
B	2	493.88
C	3	523.25
C♯ *or* D♭	4	554.37
D	5	587.33
D♯ *or* E♭	6	622.25
E	7	659.26
F	8	698.46
F♯ *or* G♭	9	739.99
G	10	783.99
G♯ *or* A♭	11	830.61
A	12	880.00

(pitch)



$$\text{pitch}(i) = 440 * 2^{\,i/12}$$

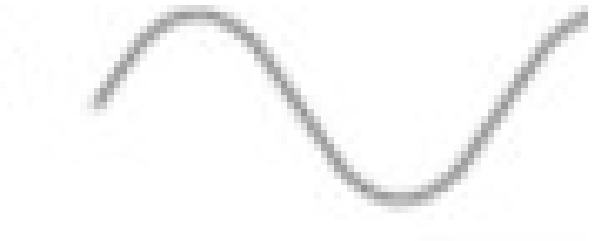
Example

The note “concert-C” (middle Do) is represented by
a *sine wave*, scaled to oscillate at 523.25 Hz (cycles per second)

```
% java Sound 523.25
(plays Do)

% java Sound 587.33
(plays Re)

% java Sound 659.26
(plays Mi)
```

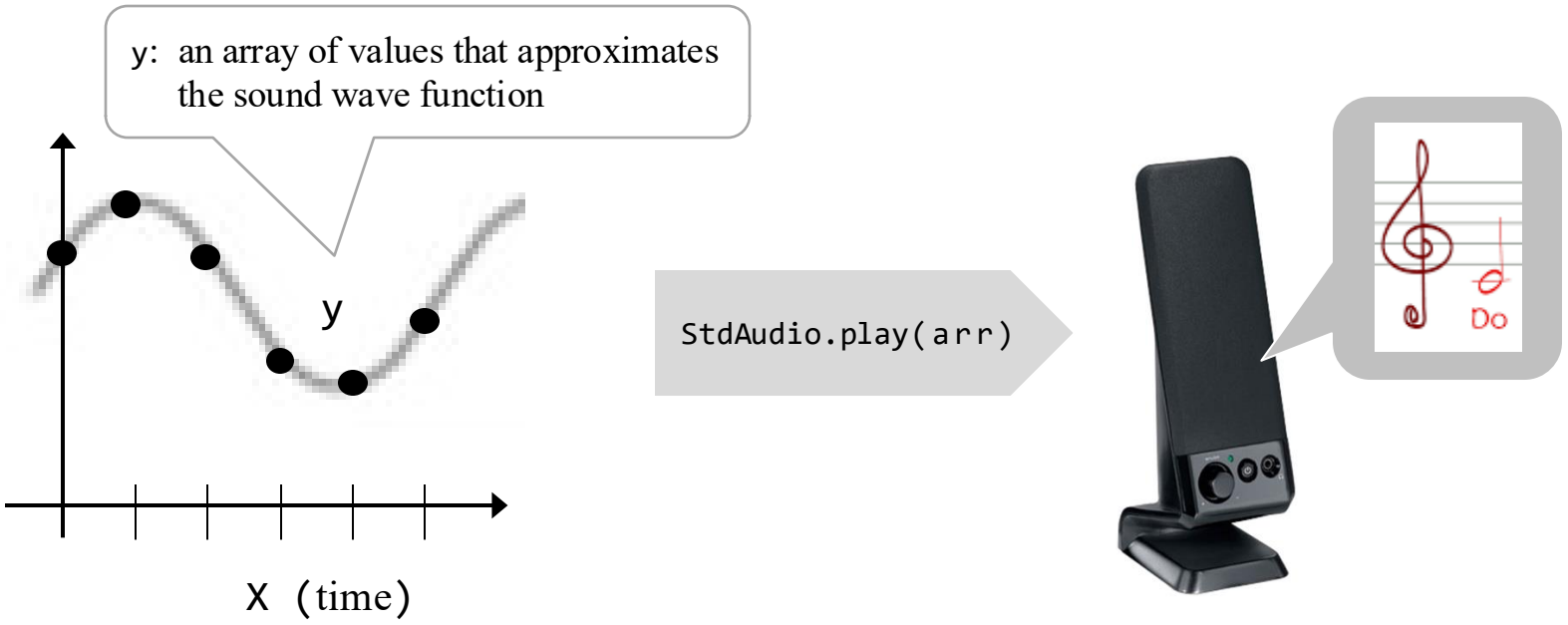
Digitized sound

- Using software to render sound by generating a sampled (digitized) sound wave



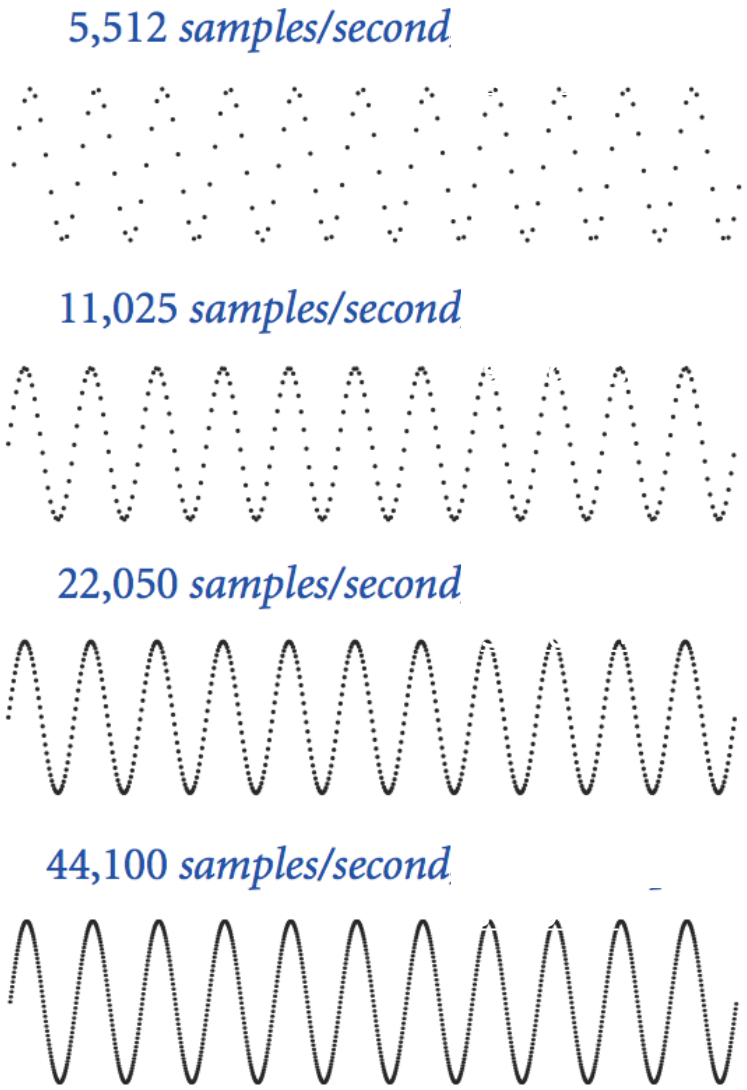
Digitized sound

- Using software to render sound by generating a sampled (digitized) sound wave
- The digitized sample is then played by the computer's speakers



- `StdAudio.play(arr)` sends the sequence of the array values to the audio driver, which controls the sound card (hardware), that controls the speakers
- `StdAudio` = Sound library (Java)

Sampling



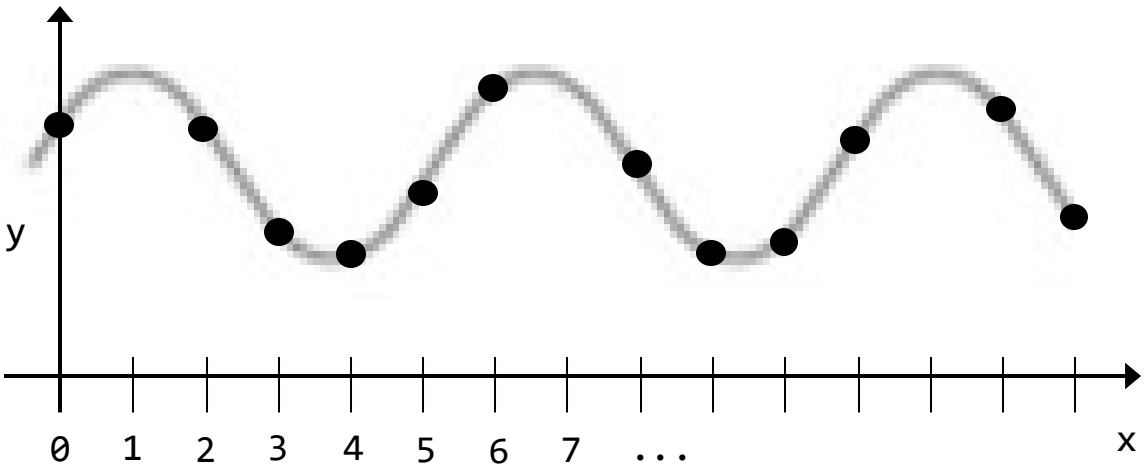
Sampling rate (= "resolution")

- Number of samples per second (sps)
- CD quality: Commonly used sampling rate, sps = 44100

Playing tones (תדרים)

Example: Play a C note (*hz* = 523.25) for 2 seconds, sps (samples per second) = 44100

Implementation: Create an array (`y`) that samples the sine wave function, according to the given parameters, and then "play the array", using the function call `StdAudio.play(y)`



$$y(x) = \sin\left(x \cdot hz \cdot \frac{2\pi}{sps}\right)$$

$x = 0, 1, 2, 3, \ldots$

Number of x points = $sps \cdot seconcs$

Playing tones (תדרים)

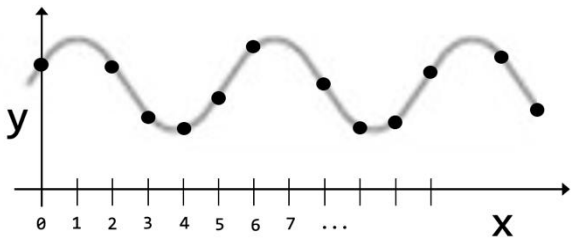
Example: Play a C note (*hz* = 523.25) for 2 seconds, sps (samples per second) = 44100

Implementation: Create an array (`y`) that samples the sine wave function, according to the given parameters, and then "play the array", using the function call `StdAudio.play(y)`

```
    // Returns an array representing a sampled sine wave modulated according
    // to the given pitch (frequency, in Hz) and  seconds, using a sampling
    // rate of 44100 values per second.
    public static double[] tone(double pitch, double seconds) {
        int sps = 44100;
        int N = (int) (seconds * sps);
        double[] y = new double[N];
        for (int x = 0; x < N; x++) {
            y[x] = Math.sin(x * pitch * (2 * Math.PI) / sps);
        }
        return y;
    }
}
```

$$y(x) = \sin\left(x \cdot hz \cdot \frac{2\pi}{sps}\right)$$

$x = 0, 1, 2, 3, \ldots$



Playing tones (תדרים)

Example: Play a C note (*hz* = 523.25) for 2 seconds, sps (samples per second) = 44100

Implementation: Create an array (`y`) that samples the sine wave function, according to the given parameters, and then "play the array", using the function call `StdAudio.play(y)`

```
// A library of functions for generating musical notes.
public class Sound {
    public static void main(String[] args) {
        double pitch = Double.parseDouble(args[0]);
        // Creates a tone array representing the given pitch (frequency)
        double[] tone = tone(pitch, 3);   // plays it 3 seconds
        StdAudio.play(tone);
    }

    // Returns an array representing a sampled sine wave modulated according
    // to the given pitch (frequency, in Hz) and  seconds, using a sampling
    // rate of 44100 values per second.
    public static double[] tone(double pitch, double seconds) {
        int sps = 44100;
        int N = (int) (seconds * sps);
        double[] y = new double[N];
        for (int x = 0; x < N; x++) {
            y[x] = Math.sin(x * pitch * (2 * Math.PI) / sps);
        }
        return y;
    }
}
```

$$y(x) = \sin\left(x \cdot hz \cdot \frac{2\pi}{sps}\right)$$

$x = 0, 1, 2, 3, \ldots$

Plays some arbitrary tones:

```
% java Sound 8526

% java Sound 576

% java Sound 14365
```

Playing notes (תווים)

```
// A library of functions for generating musical notes.
public class Sound {
    public static void main(String[] args) {
        // Plays Do, Re, Mi, each for 0.5 second.
        StdAudio.play(note(3, .5));
        StdAudio.play(note(5, .5));
        StdAudio.play(note(7, .5));
    }

    // Returns an array representing a sampled sine wave modulated according
    // to the given pitch (frequency, in Hz) and  seconds, using a sampling
    // rate of 44100 values per second.
    public static double[] tone(double pitch, double seconds) {
        int sps = 44100;
        int N = (int) (seconds * sps);
        double[] y = new double[N];
        for (int x = 0; x < N; x++) {
            y[x] = Math.sin(x * pitch * (2 * Math.PI) / sps);
        }
        return y;
    }

    // Returns an array representing the given musical note, and seconds.
    // (0 is A, 1 is A#, 2 is B, 3 is C, 4 is C#, ...).
    public static double[] note(int i, double seconds) {
        double pitch = 440.0 * Math.pow(2, i / 12.0);
        return tone(pitch, seconds); // Computes the array (previous slide)
    }
    ...
}
```

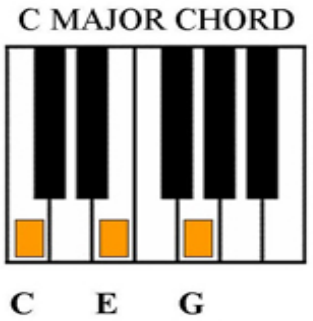
$$\text{pitch}(i) = 440 * 2^{i/12}$$

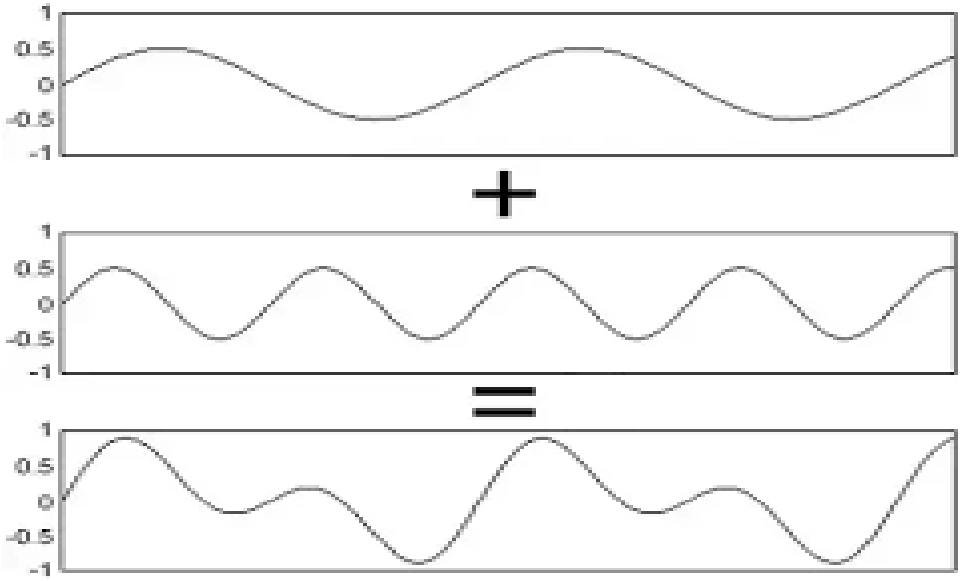
note	*i*	*frequency*
A	0	440.00
A♯ *or* B♭	1	466.16
B	2	493.88
C	3	523.25
C♯ *or* D♭	4	554.37
D	5	587.33
D♯ *or* E♭	6	622.25
E	7	659.26
F	8	698.46
F♯ *or* G♭	9	739.99
G	10	783.99
G♯ *or* A♭	11	830.61
A	12	880.00

Plays Do Re Mi:

```
% java Sound 523.25

% java Sound 587.23

% java Sound 659.26
```

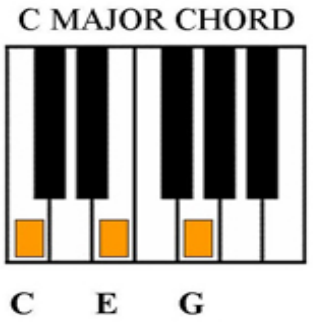
Playing chords (אקורדים)



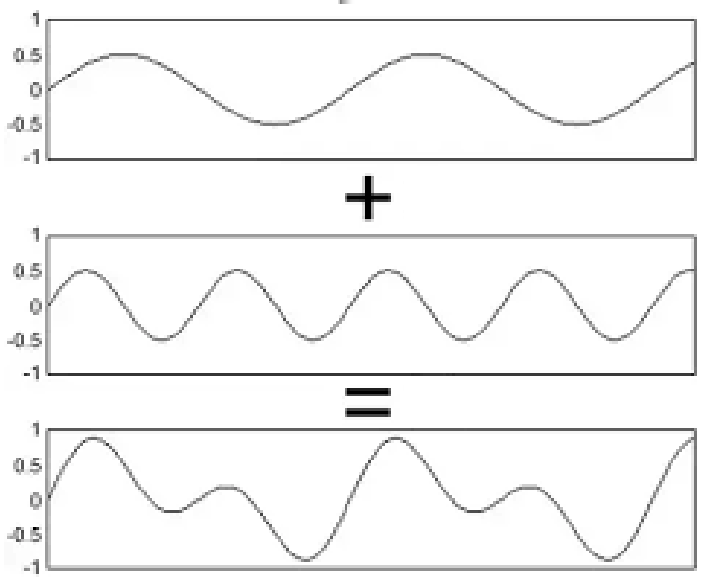



```
// Returns the weighted sum of two given arrays, using two given weights.
 private static double[] sum(double[] a, double[] b, double awt, double bwt) {
     double[] c = new double[a.length];
     for (int i = 0; i < a.length; i++)
         c[i] = awt * a[i] + bwt * b[i];
     return c;
 }
```

Playing chords (אקורדים)




```
// A library of functions for generating musical notes.
public class Sound {
    public static void main(String[] args) {
        // Plays the Do-major chord (Do, Mi and Sol), for 5 seconds
       StdAudio.play(chord(3, 7, 10, 5));
    }
    // Tone and Note functions come here
    // Returns the weighted sum of two given arrays, using two given weights.
    private static double[] sum(double[] a, double[] b, double awt, double bwt) {
        double[] c = new double[a.length];
        for (int i = 0; i < a.length; i++)
            c[i] = awt * a[i] + bwt * b[i];
        return c;
    }

    // Returns a 3-note chord by combining the three given notes.
    public static double[] chord(int i1, int i2, int i3,
                                 double seconds) {
        double[] a1 = note(i1, seconds);
        double[] a2 = note(i2, seconds);
        double[] a3 = note(i3, seconds);
        // builds and returns a mix of the three notes
        double[] s = sum(a1, a2, .5, .5);
        s = sum(s, a3, .5, .5);
        return s;
    }
}
```



Playing tunes (לחנים)

```
// A library of functions for generating musical notes.
public class Sound {
    public static void main(String[] args) {
        In in = new In(fileName);
        while (!in.isEmpty()) {
           int i = in.readInt();
           double seconds = in.readDouble();
           StdAudio.play(note(i, seconds));
        }
    }
    // All other Sound functions come here
}
```

Elise (Beethoven):

```
% more elise.txt
7 .3
6 .3
7 .3
6 .3
7 .3
2 .3
5 .3
3 .3
0 1.0

% java Sound elise.txt
```

(we'll hear the first 9 notes of Elise)