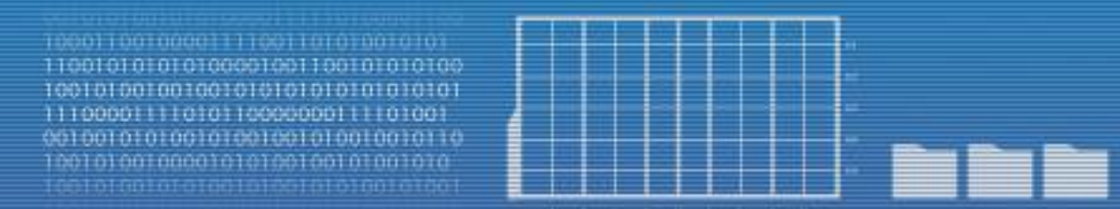
DEPARTMENT OF INFORMATION SYSTEMS AND COMPUTER SCIENCE

Vectors and Circles

A Review

Lecture Time!

- Vectors: Math
- Circles: More Math
- SFML: Now With More Math



DISCS

WARNING

- This should be a review of linear algebra
- Since this subject is primarily for DGDD majors, I am going to fast-forward through a lot of this

Vectors

- An n-tuple of numbers from the domain of real numbers
  - Number of dimensions = n
  - Therefore, an example of a 2-D vector would be (4, –8.9)
- Interpreted as displacement from the origin to a specific point
  - Position
  - Can be used for other things (like a rectangle's size)

Vector Arithmetic

- Vector addition:

  $a + b = (a_x + b_x, a_y + b_y)$

  - What is the resulting vector?
- Useful for:
  - Translation (motion, update of position)
  - Subtraction ($a - b = a + (-b)$)

Vector Arithmetic

- Vector subtraction:

  $a - b = (a_x - b_x, a_y - b_y)$

  - What is the resulting vector if b is used as an origin?

- Useful for:

  - Distance (between two points, used in conjunction with getting a vector's magnitude)
  - Getting an "opposing" vector (faces the opposite direction, same magnitude)

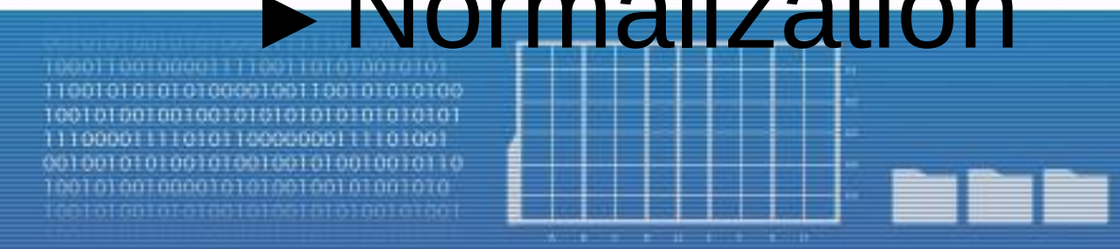
DISCS

Vector Arithmetic

- Multiplying a vector with a scalar:

  $sa = (sa_x, sa_y)$

  - What is the resulting vector?
  - Division is just multiplying by a reciprocal of the divisor
- Useful for:
  - Scaling (shrinking or enlarging relative to a local origin)
  - Normalization



Vector Arithmetic

- Getting the magnitude of a vector:

  $|a| = \text{sqrt}(a_x^2 + a_y^2)$

  - What is magnitude, assuming a line segment formed by point a and the origin?

- Useful for:
  - Distance (between two points, used in conjunction with subtraction)
  - Speed (given a velocity vector)

Vector Arithmetic

- Normalizing a vector:

  $\hat{a} = (a_x / |a|, a_y / |a|)$

  - What is the resulting vector?
- Useful for:
  - Emphasis on direction without scale (using non-unit vectors may add extra “corrective” steps in some operations)

Dot Product

- Dot product of two vectors:

$$a \bullet b = a_x b_x + a_y b_y$$

  - Also known as scalar product

- Useful for:
  - Lots of stuff

Dot Product

- $a \bullet b = |a||b| \cos \theta$
  - θ being the smallest angle between a and b
- $a \bullet a = |a|^2$

Pseudo Cross Product

- Perp-dot product of two vectors: $a^P \bullet b$
  - a^P is counterclockwise vector perpendicular to a
  - a = (x, y), a^P = (-y, x)
- Useful for:
  - Determining if b is counterclockwise (positive), clockwise (negative), or along a (zero)

Lines

- A line can be defined as the set of points expressible as the linear combination of two distinct points A and B:
  - L(t) = (1 - t)A + tB = A +t(B - A)
    - In other words, just need a point and a direction
- A ray is a line but $t \geq 0$
- A segment is a line but $0 \leq t \leq 1$

Projection onto a Vector

- Useful for things like Separating Axis Theorem
- Assume u is a unit length vector and any other vector v
- The projection of v onto u is another vector along u
  - Can be expressed as some value L multiplied by u

Projection onto a Vector

- L = v ● u
- The projection of v onto u therefore is
  proj(v, u) = (v ● u)u
  - Proof will be or was already taught in Linear Algebra
- Projecting v onto a non-unit vector d:
  proj(v, d) = ((v ● d) / (d ● d))d

Circles

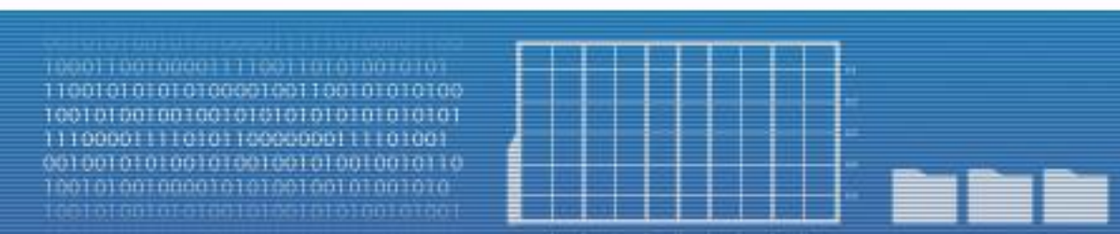
- Usually rendered as a polygon with many sides
- Represented by a vector indicating its center and a value representing its radius
- Most memory-efficient compared to other 2-D shape representations

Circles

- Overlap test involves checking if the distance between two circle centers is less than or equal to the sum of their radii
  - But square root operations are expensive
  - Is there a way to perform this check without the square root operation?

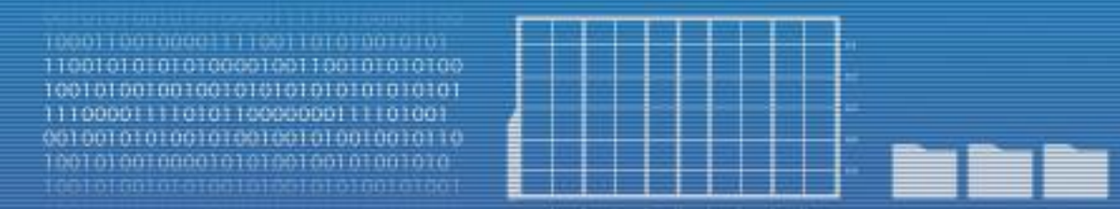
SFML Window

```
int main()
{
    // create window
    sf::RenderWindow window(
        sf::VideoMode( 480, 320 ),
        "Title Goes Here" );
// other initializations here
// ...
```



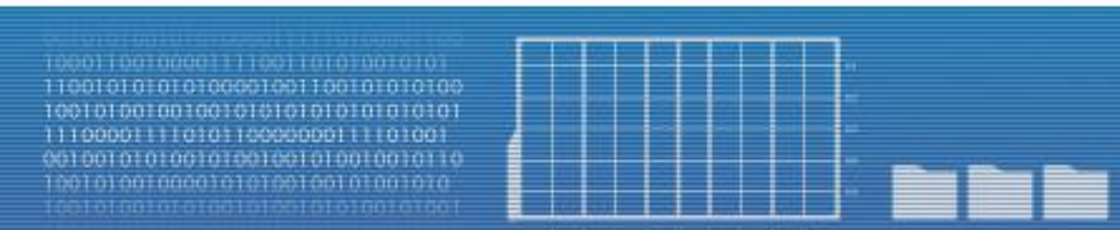
SFML Window

```
// ...
while( window.isOpen() )
{
	// check all the window's events that were triggered
	// since the last iteration of the loop
	sf::Event event;
	while( window.pollEvent( event ) )
	{
		// "close requested" event
		if( event.type == sf::Event::Closed )
		{
			window.close();
		}
	}
	// ...
```



SFML Graphics

```
// in initialization part, after creating window
sf::CircleShape circ;
sf::RectangleShape rect;
circ.setPosition( 40, 40 );
circ.setRadius( 100.0f );
circ.setFillColor( sf::Color( 0, 255, 0 ) );
rect.setPosition( 340, 180 );
rect.setSize( sf::Vector2f( 100.0f, 60.0f ) );
rect.setFillColor( sf::Color( 0, 0, 255 ) );
```



SFML Graphics

```
// in window.isOpen() loop
// always clear buffer at start of current frame
window.clear( sf::Color::Black );

// draw shapes
window.draw( circ );
window.draw( rect );

// always call next line at end of current frame
window.display();
```

Exercises

- Make programs that draw the following:
  - RectangleShape that travels left and right
  - CircleShape that also moves in a circle
  - RectangleShape that slowly changes size and color
    - Should eventually loop back to its original size and color
- Ensure that your programs run at (roughly) 60 frames per second

Homework

- Create an array of 60 CircleShapes and 40 RectangleShapes
  - CircleShape radius is 30.0f
  - RectangleShape dimensions are 50.0f x 50.0f
- Assign different colors to each shape
  - Cycle through red, green, blue, yellow, cyan, and white
  - Do consider a convenience function for this

Homework

- Starting positions of these shapes should be random
  - (0, 0) to (window_w – 1, window_h – 1)
- CircleShapes should drift downwards at a rate of 20 pixels per second
- RectangleShapes should drift to the right at a rate of 20 pixels per second
  - Not 20 pixels per tick (movement should be very smooth)