CS529
Fundamentals of Game Development

Lecture 9b

Antoine Abi Chakra
Karim Fikani

Questions

- 2D Transformations
- Homogeneous Coordinates and Matrix Representation
- Composition of 2D Transformations

Overview

- Object Kinematics
- Object Animation
  - Frame Based
  - Time Based
  - Acceleration
- Asteroids
  - Ship's Acceleration, Deceleration
  - Velocity Cap
  - Friction

Object Kinematics (1/2)

- A CS529 object has a position and a velocity
  - Objects do not respond to forces
  - Objects move with constant velocity – that is, zero acceleration
  - Simplest to simulate

Object Kinematics (2/2)

- Obvious structure definition in C might look like (neglecting appearance and other properties):

```
struct Object
{
        … // Object methods and variables
        float  p[2];     // Position
        float  v[2];     // Velocity
};
```

Overview

- Object Kinematics
- Object Animation
  - Frame Based
  - Time Based
  - Acceleration
- Asteroids
  - Ship's Acceleration, Deceleration
  - Velocity Cap
  - Friction

Object Animation – Frame Based

- Specify the initial position p and velocity v of each object
  - Velocity consists of a speed and direction vector (that is, a vector with unit magnitude)
  - Having the directional vector normalized allows us to change its direction without affecting its magnitude
- Every frame, update object's previous position:

$$\vec{p} += \vec{v}$$

Object Animation – Time Based

- Each frame:
  - Compute time interval between previous and current frame – dt
  - Compute object's displacement within time interval *dt*: $\vec{v} * dt$
  - Finally, compute object's new position as

$$\vec{p} += \vec{v} * dt$$

Object Animation – Based on Velocity

- Compute time interval between previous and current frame
  - *dt*
- Compute object's displacement within time interval *dt*:
  - *v* dt*
- Finally, compute object's new position as

$$newPos = v * dt + currPos$$

Object Animation – Based on Acceleration

- Computing object's new position as:

$$newPos = \frac{1}{2} a * dt^2 + v * dt + currPos$$

References

- Computer Graphics Principles and Practice by Foley, van Dam, Feiner and Hughes

Overview

- Object Kinematics
- Object Animation
  - Frame Based
  - Time Based
  - Acceleration
- Asteroids
  - Ship's Acceleration, Deceleration
  - Velocity Cap
  - Friction

Asteroids

- Bullets & Asteroids have constant velocities
  - Velocities are set at creation time
- The ship has a varying velocity
  - Depending on its acceleration, which in turn depends on user input
  - Its acceleration is non-zero when either the forward or backward key is pressed

Asteroids – Ship's Acceleration (1/5)

- The ship's new position can be calculated in 2 ways
  - Directly from the acceleration:

$$newPos = \frac{1}{2} a * dt^2 + currVel * dt + currPos$$

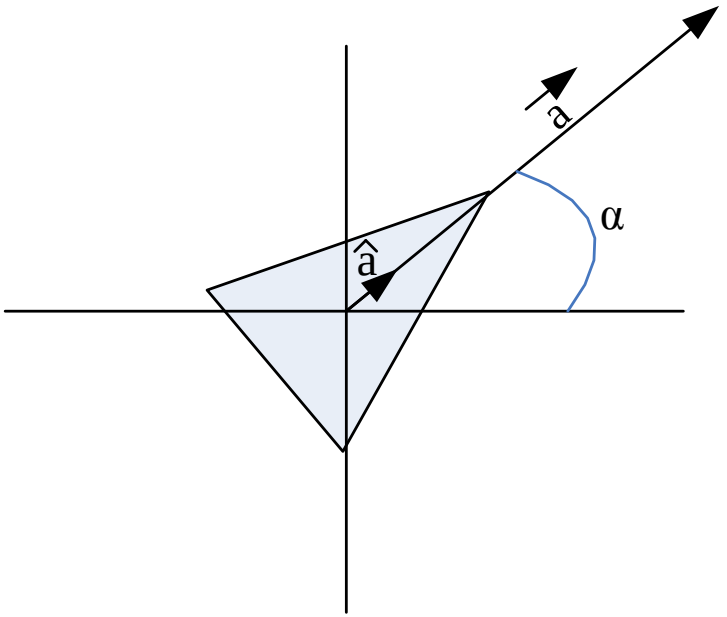
  - Calculate the new velocity, then use it to get the new position:

$$newVel = a * dt + currVel$$

$$newPos = newVel * dt + currPos$$

Asteroids – Ship's Acceleration (2/5)

- When the forward button is pressed, a forward acceleration should be applied to the ship
- Ship data that we have:
  - Ship's current position
  - Ship's current velocity
  - Ship's current orientation: α
- What we need to calculate:
  - Ship's acceleration: $\vec{a}$



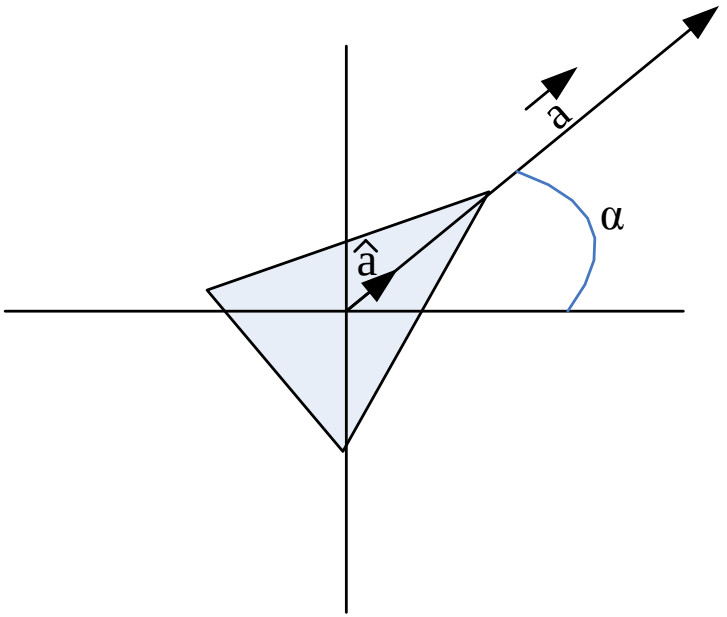
Asteroids – Ship's Acceleration (3/5)

- **The new acceleration vector $\vec{a}$ is independent from its current velocity**
- We can use the ship's current orientation α to compute the normalized acceleration vector: $\hat{a}$

$$\hat{a} = (\cos\alpha\,;\sin\alpha)$$

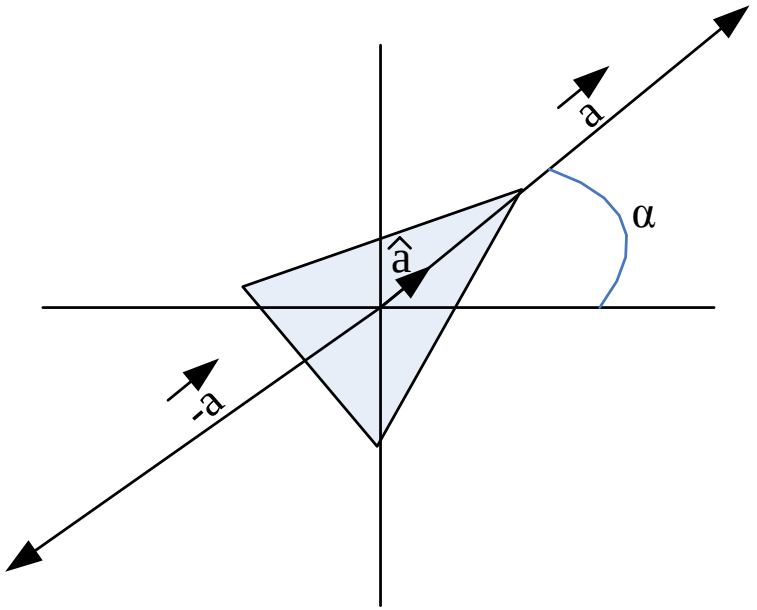
- Scaling $\hat{a}$ by a predefined value will give the full acceleration vector $\vec{a}$

$$\vec{a} = (\hat{a}.x * 100; \hat{a}.y * 100)$$



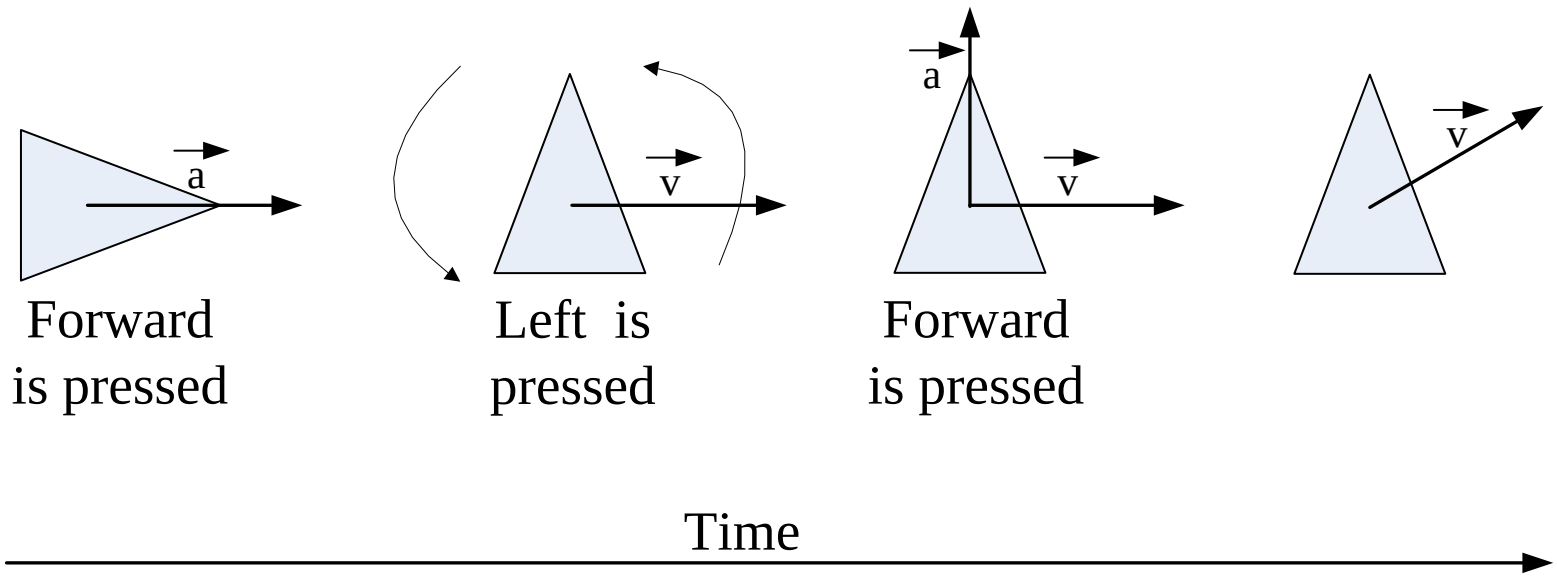
Asteroids – Ship's Acceleration (4/5)

- The deceleration vector $-\vec{a}$ is similarly calculated
  - It's just the opposite vector
- Compute $\vec{a}$ as described previously
- Negate both coordinates to get $-\vec{a}$



Asteroids – Ship's Acceleration (5/5)

- Assuming the ship is initially not moving



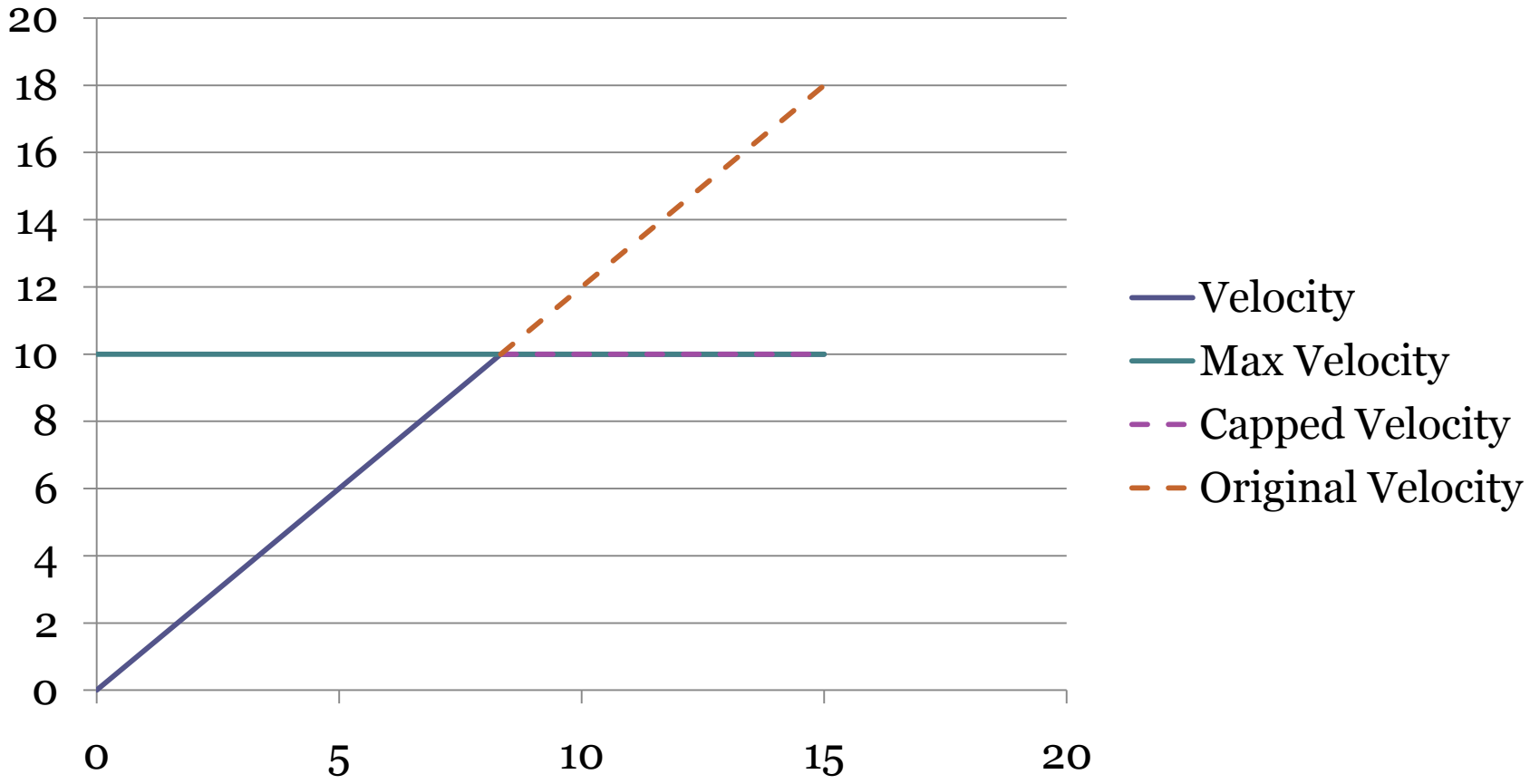
Asteroids – Ship's Velocity Cap (1/4)

- Both techniques will achieve the same results
- But the 2nd one give us more flexibility
  - Allows us to manipulate the velocity before updating the position
    - Set a velocity cap
    - Simulate friction
    - Etc..

Asteroids – Ship's Velocity Cap (2/4)

- A velocity cap can be set in different ways
- Simplest:
  - Set a maximum velocity magnitude
  - Every time a new velocity is computed, compare its magnitude to the maximum
  - Greater? Set it to the maximum
- Works, but feels unrealistic
  - Reaching the maximum velocity is instantaneous
  - Maximum velocity is not reached smoothly

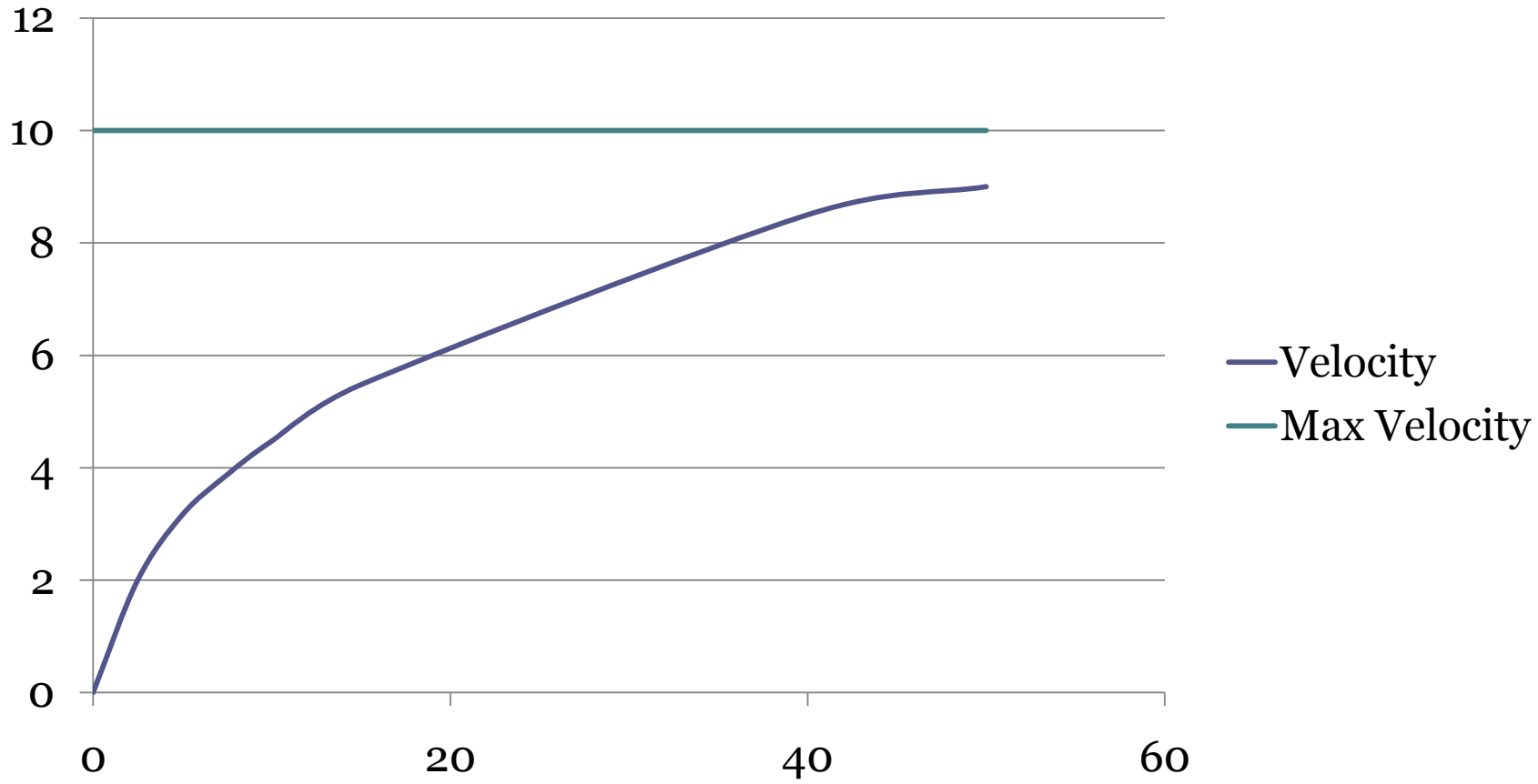
Asteroids – Ship's Velocity Cap (3/4)



Asteroids – Ship's Velocity Cap (4/4)

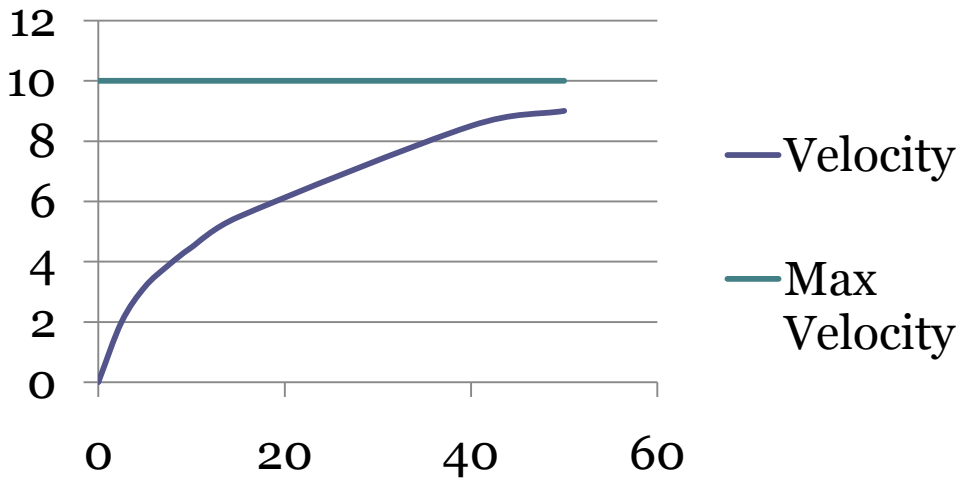
- In reality, maximum velocity is reached due to **friction**
- Friction is a force
  - Accelerations are derived from forces
  - Velocities are derived from accelerations
  - Conclusion: Velocities are affected by friction!
- Friction allows objects to smoothly reach their maximum velocities

Asteroids – Better Velocity Cap (1/6)



Asteroids – Better Velocity Cap (2/6)

- In CS529, we're not using forces
  - Accelerations and velocities are directly assigned
  - This means that we can't apply friction
  - But we still want to achieve the following result:



Asteroids – Better Velocity Cap (3/6)

- Friction will be emulated
- There are different techniques to achieve a smooth velocity capping
- Previous Implementation

$$newVel = a * dt + currVel$$

$$newPos = newVel * dt + currPos$$

Asteroids – Better Velocity Cap (4/6)

- Our velocity capping technique

$$newVel = a * dt + currVel$$

- New step: $newVel = newVel * 0.99$

$$newPos = newVel * dt + currPos$$

- Isn't that just reducing the velocity by 1%?

Asteroids – Better Velocity Cap (5/6)

dt = 1	Frame 1		Frame 2		Frame 3	
	Original	*0.99	Original	*0.99	Original	*0.99
Given: currPos	(0;0)	(0;0)	(2;3)	(1.98;2.97)	(4;6)	(3.94;5.91)
Given: currVel	(0;0)	(0;0)	(2;3)	(1.98;2.97)	(6;9)	(5.92;8.88)
Given: a	(2;3)	(2;3)	(2;3)	(2;3)	(2;3)	(2;3)
Computed: newVel	(2;3)	(1.98;2.97)	(4;6)	(3.94;5.91)	(8;12)	(7.84;11.76)
Computed: newPos	(2;3)	(1.98;2.97) 99% of the original value	(6;9)	(5.92;8.88) 98.6% of the original value	(12;18)	(11.78; 17.67) 98.1% of the original value

Asteroids – Better Velocity Cap (6/6)

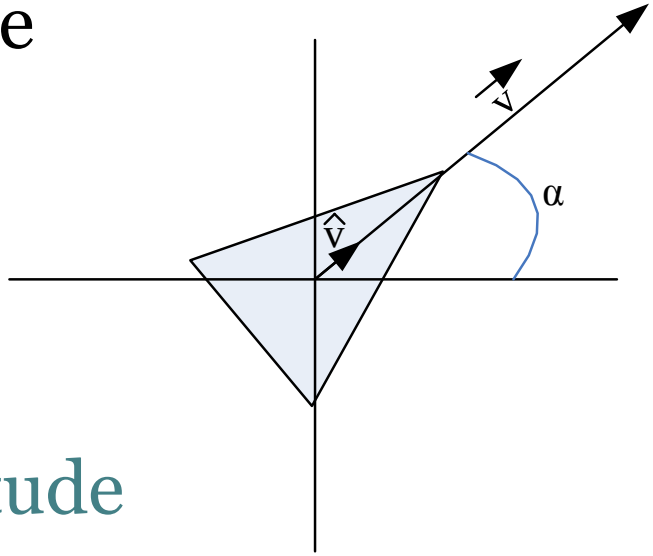
- Every frame, the velocity is reduced by a greater %
  - Feels realistic
  - Maximum velocity is reached smoothly

Creating Bullets (1/2)

- For simplicity, bullets will be created at the same location of the ship
  - Which means the ship's current position is needed
- Bullets are not accelerated
  - They have a constant velocity
  - That velocity has a predefined magnitude
    - Similar to the ship's predefined acceleration magnitude
  - Problem: Computing the direction of the bullet's velocity

Creating Bullets (2/2)

- Computing a newly created bullet's direction is similar to computing the ship's acceleration
  - They both depend on the ship's orientation α
  - Compute $\hat{v}$, which is equal to $\hat{a}$
  - Scale $\hat{v}$ by the predefined magnitude in order to get $\vec{v}$



$$\vec{v} = (\hat{v}.x * 200 + \hat{v}.y * 200)$$

Creating Asteroids

- Asteroids, like bullets, have a constant velocities
- The 2 differences:
  - Asteroids are created at random locations (preferably outside the viewport, or at a destroyed asteroid's last position), while bullets are created at the ship's current position
  - Asteroids' velocities have "random" directions, while bullet velocities' direction depend on the ship's orientation