

Boids!

Hawk Weisman and Willem Yarbrough

Department of Computer Science
Allegheny College

April 27, 2015

What are Boids?

- ▶ An artificial life simulation [1, 3]
- ▶ 'Bird-oid' flocking behaviour [1, 3]
- ▶ First described by Craig Reynolds in 1987 [3]

Why Boids?

- ▶ Some major appearances:
 - ▶ *Half-Life* (1998)
 - ▶ *Batman Returns* (1992)
- ▶ Other applications:
 - ▶ Swarm optimization
 - ▶ Unmanned vehicle guidance

Our Implementation

- ▶ **Haskell** programming language:
 - ▶ A strongly-typed, lazy, purely functional programming language
 - ▶ **Why?**
 - ▶ Good for rapid prototyping [2]
 - ▶ Prior experience
 - ▶ Explore non-OO ways of representing agents
- ▶ **Our simulation:** Boids in a toroidal 2D space

What is a Boid?

- ▶ Consists of
 - ▶ A position p_i
 - ▶ A velocity vector \vec{v}_i
 - ▶ A sight radius r

What is a Boid?

- ▶ Consists of
 - ▶ A position p_i
 - ▶ A velocity vector \vec{v}_i
 - ▶ A sight radius r
- ▶ In Haskell:

```
type Vector = V2 Float
```

```
type Point  = V2 Float
```

```
type Radius = Float
```

```
data Boid = Boid { position :: !Point  
                  , velocity :: !Vector  
                  , radius   :: !Radius  
                  }
```

```
deriving (Show)
```

Boid Behaviour

- First, we define some types:

```
type Update      = Boid -> Boid
```

```
type Perception = [Boid]
```

```
type Behaviour  = Perception -> Update
```

Boid Behaviour

- First, we define some types:

```
type Update      = Boid -> Boid
type Perception  = [Boid]
type Behaviour   = Perception -> Update
```

- Functions for finding a boid's neighborhood:

```
inCircle :: Point -> Radius -> Point -> Bool
inCircle p_0 r p_i = ((x_i - x)^n + (y_i - y)^n) <= r^n
  where x_i = p_i ^. _x
        y_i = p_i ^. _y
        x   = p_0 ^. _x
        y   = p_0 ^. _y
        n   = 2 :: Integer
```

```
neighborhood :: World -> Boid -> Perception
neighborhood world self =
  filter (inCircle cent rad . position) world
  where cent = position self
        rad  = radius self
```


Separation steering vector

- ▶ Tendency to avoid collisions with other boids

$$\vec{s}_i = - \sum_{\forall b_j \in V_i} (p_i - p_j)$$

Separation steering vector

- Tendency to avoid collisions with other boids

$$\vec{s}_i = - \sum_{\forall b_j \in V_i} (p_i - p_j)$$

- In Haskell:

```
separation :: Boid -> Perception -> Vector
separation self neighbors =
  let p = position self
  in negated $
    sumV $ map (^-^ p) $ positions neighbors
```

Cohesion steering vector

- ▶ Tendency to steer towards the centre of visible boids
- ▶ Calculated in two steps.

Cohesion steering vector

- ▶ Tendency to steer towards the centre of visible boids
- ▶ Calculated in two steps.

1. Find the centre:

$$c_i = \sum_{\forall b_j \in V_i} \frac{p_j}{m} \quad (1)$$

Cohesion steering vector

- ▶ Tendency to steer towards the centre of visible boids
- ▶ Calculated in two steps.

1. Find the centre:

$$c_i = \sum_{\forall b_j \in V_i} \frac{p_j}{m} \quad (1)$$

2. Calculate vector:

$$\vec{k}_i = c_i - p_i$$

Cohesion steering vector

- ▶ Tendency to steer towards the centre of visible boids
- ▶ Calculated in two steps.

1. Find the centre:

$$c_i = \sum_{\forall b_j \in V_i} \frac{p_j}{m} \quad (1)$$

2. Calculate vector:

$$\vec{k}_i = c_i - p_i$$

- ▶ In Haskell:

```
centre :: Perception -> Vector
centre boids =
    let m = fromIntegral $ length boids :: Float
    in sumV (positions boids) ^/ m
cohesion :: Boid -> Perception -> Vector
cohesion self neighbors =
    let p = position self
    in centre neighbors ^-^ p
```

Alignment steering vector

- ▶ Tendency to match velocity with visible boids

$$\vec{m}_i = \sum_{\forall b_j \in V_i} \frac{\vec{v}_j}{m}$$

Alignment steering vector

- Tendency to match velocity with visible boids

$$\vec{m}_i = \sum_{\forall b_j \in V_i} \frac{\vec{v}_j}{m}$$

- In Haskell:

```
alignment :: Boid -> Perception -> Vector
           -- :: Boid -> [Boid]      -> V2 Float
alignment _ [] = V2 0 0
alignment _ neighbors =
    let m = fromIntegral $ length neighbors :: Float
    in (sumV $ map velocity neighbors) ^/ m
```


Simulating a boid

1. Velocity update

$$\vec{v}_i' = \vec{v}_i + S.\vec{s}_i + K.\vec{k}_i + M.\vec{m}_i$$

Where S , K , and $M \in [0, 1]$

Simulating a boid

1. Velocity update

$$\vec{v}_i' = \vec{v}_i + S.\vec{s}_i + K.\vec{k}_i + M.\vec{m}_i$$

Where S , K , and $M \in [0, 1]$

2. Position update

$$p_i' = p_i + \Delta t \vec{v}_i$$

Simulating a boid

1. In Haskell:

```
steer :: Weights -> Behaviour
steer (s, c, m) neighbors self =
    let s_i  = s *^ separation self neighbors
        c_i  = c *^ cohesion self neighbors
        m_i  = m *^ alignment self neighbors
        v'   = velocity self ^+^ s_i ^+^ c_i ^+^ m_i
        p    = position self
        p'   = p ^+^ v'
    in self { position = p', velocity = v'}
```

A brief demonstration

References



Christopher Hartman and Bedrich Benes.

Autonomous boids.

Computer Animation and Virtual Worlds, 17(3-4):199–206, 2006.



Paul Hudak and Mark P Jones.

Haskell vs. Ada vs. C++ vs. awk vs.... an experiment in software prototyping productivity.

Contract, 14(92-C):0153, 1994.



Craig W Reynolds.

Flocks, herds and schools: A distributed behavioral model.

ACM Siggraph Computer Graphics, 21(4):25–34, 1987.