CSC 240 Computer Graphics Lecture 7.5: Animation

Nick Howe

Smith College

Animations

The human eye has a bias towards perceiving motion

- **Persistence of vision**
  - Briefly presented stimulus remains perceptible for up to 1/16 second
  - If similar stimulus is presented within this time, will seem continuous
  - Animation possible by displaying sequence of slightly varying images

Single Object Animation

We can use transforms to do animation.

- Initialization: Define object appearance, initial state

  State = position, angle, scale, etc.

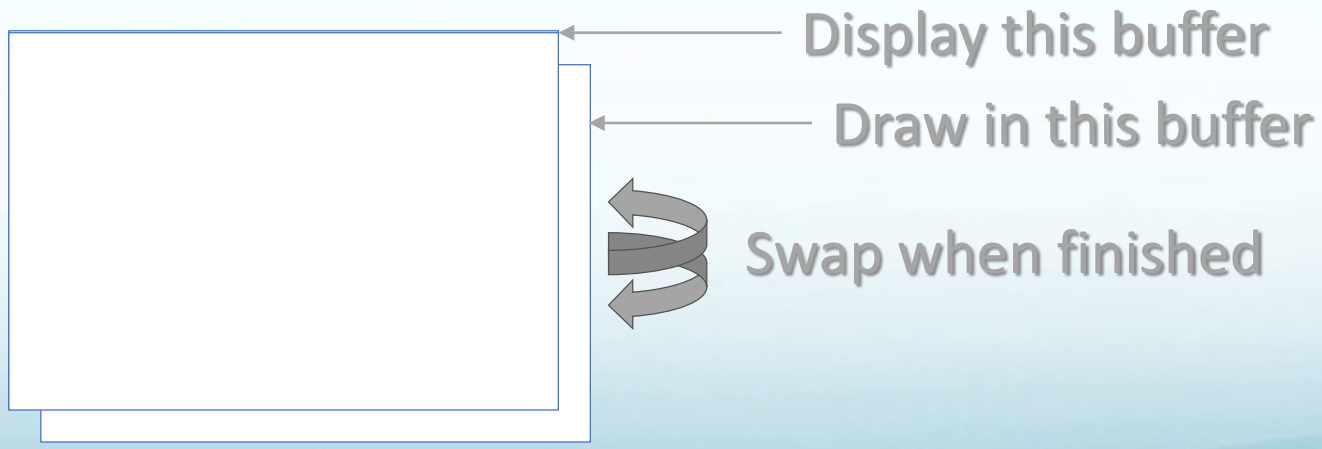
- Infinite loop:

  1. Erase screen / draw background
  2. Draw object(s) using transforms based upon current state
  3. Update state for next iteration

https://www.deviantart.com/flutterluv/art/A-Cow-Jumped-Over-The-Moon-622575145

Buffering

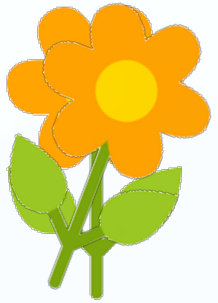
- For animation, need to draw and redraw graphics
- Complex renderings take time to produce
- Don’t want user to see drawing process
- Solution: **double buffering**

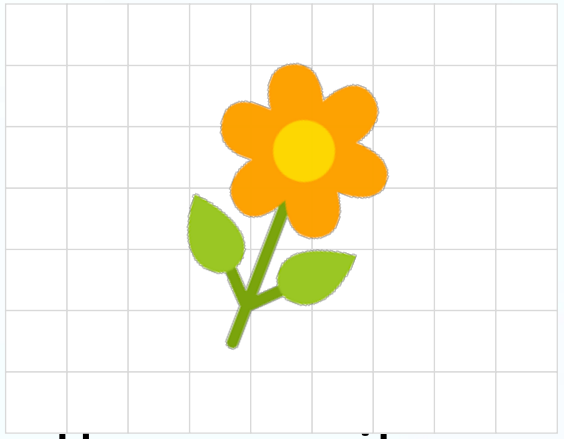


Modeling with Transforms

Typical work flow:

1. Design your object
2. Apply transform (scale, rotate, translate) & draw in scene





Note that in code, the applications should be applied in the opposite order (translate first, rotate second, scale third)

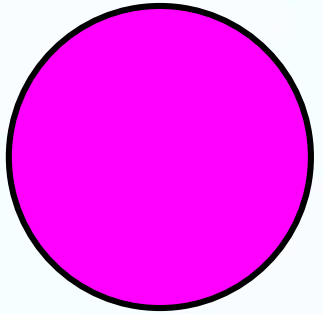
$T = I$	$P' = I \cdot P$	Initial state
$T = I \cdot T$	$P' = I \cdot T \cdot P$	Apply translate
$T = I \cdot T \cdot R$	$P' = I \cdot T \cdot R \cdot P$	Apply rotate
$T = I \cdot T \cdot R \cdot S$	$P' = I \cdot T \cdot R \cdot S \cdot P$	Apply scale

**If you have multiple objects to draw: use `save()` before you start and `restore()` when finished to undo all transforms*

Javascript Objects

In larger programs, it is often useful to group related data and actions.

- Example: object that will be displayed
  - Details about appearance
  - Details about location & pose
  - Functions for drawing, updating, etc.



- Javascript objects provide for a named set of values
  - Those holding data are called **properties**
  - Those holding actions are called **methods**
- Access via dot notation:

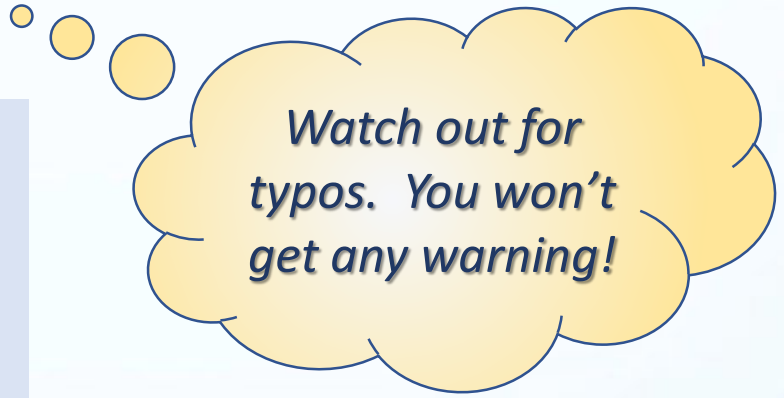
```
console.log(myCircle.area());
```

```
let myCircle = {
  radius: 32,
  color: "magenta",
  area: function() {
    return Math.PI*this.radius**2
  }
}
```

Using Javascript Objects

You can add and remove properties and methods at any time.

```
myCircle.edgeColor = "black";
delete myCircle.color;
myCircle.fillColor = "magenta";
myCircle.x = 100;
myCircle.y = 100;
```



Attempting to access a property that doesn't exist returns **undefined**.

Methods

Methods can access other properties and methods via **`this`**:

```
myCircle.draw = function(graphics) {
  graphics.beginPath();
  graphics.arc(this.x,this.y,this.radius,0,2*Math.PI);
  graphics.strokeStyle = this.edgeColor;
  graphics.stroke();
  graphics.fillStyle = this.fillColor;
  graphics.fill();
}
```

Constructors

A constructor simplifies the creation of many of similar objects

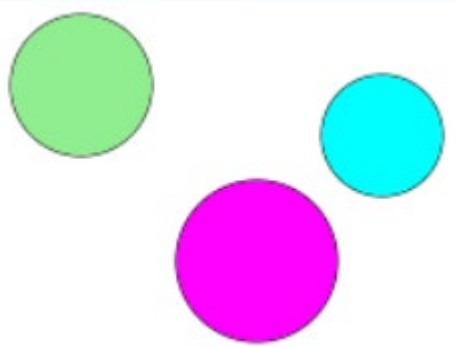
Without a constructor:

```
let myCircle = {
  radius: 32,
  color: "magenta",
  area: function() {
    return Math.PI*this.radius**2
  }
}
```

With a constructor:

```
// creates a new Circle object
function Circle(x,y,radius) {
  this.x = x;
  this.y = y;
  this.radius = radius;
  this.color = "cyan";
  this.area = function() {
    return Math.PI*radius**2;
  }
}

myCircle2 = new Circle(150,50,24);
myCircle3 = new Circle(30,30,28);
```

A Complete Script



```
// defines a new Circle object
function Circle(x,y,radius,color) {
  this.x = x;
  this.y = y;
  this.radius = radius;
  this.color = color;
  this.area = function() {
    return Math.PI*radius**2;
  }
  this.draw = function(graphics) {
    graphics.beginPath();
    graphics.arc(this.x,this.y,this.radius,0,2*Math.PI);
    graphics.strokeStyle = "black";
    graphics.stroke();
    graphics.fillStyle = this.color;
    graphics.fill();
  }
}
```

```
// set up array of Circle objects
var circles = [];
circles.push(new Circle(100,100,32,"magenta"));
circles.push(new Circle(150,50,24,"cyan"));
circles.push(new Circle(30,30,28,"lightgreen"));

// use the definitions above to draw
function draw(graphics) {
  for (c in circles) {
      circles[c].draw(graphics);
  }
}

function init() {
  canvas = document.getElementById("theCanvas");
  graphics = canvas.getContext("2d");
  draw(graphics);
}
```

Understanding Check

1. How would you change the height of **`myRect`** to 16?

   `myRect.height = 16`

2. How would you use the constructor to create a new **`Rect`** object 8 tall and 14 wide, call **`r`**?

   `Let r = new Rect(8,14)`

3. Suppose we add the following method. What value would **`r2.cost()`** return?

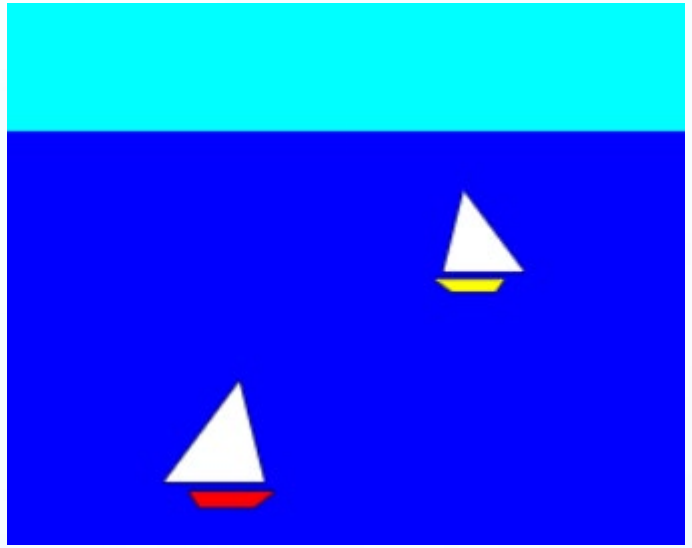
   `350`

```
let myRect = {
    width: 12,
    height: 18
}
function Rect(w,h) {
  this.width = w;
  this.height = h;
}
let r2 = {
  w: 5,
  h: 7
}
r2.cost = function() {
  return this.h*this.w*10
}
```

Use in Animation

We can use objects for 2D animation

- Object fields store state:
  - Appearance variables (color, etc.)
  - Position, rotation, scale
  - Update increments (velocity)
- Object methods:
  - Draw basic object without transformations
  - Draw object with transformations using the current state
  - Update the state after drawing



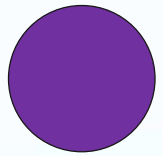
```
var canvas;    // DOM object corresponding to the canvas
var graphics;  // 2D graphics context for drawing on the canvas

var boxState = {x:50, y:50};

function animate() {
    // set up transform
    graphics.save();
    graphics.translate(boxState.x, boxState.y);

    // draw
    graphics.clearRect(0, 0, 450, 300);  // clear screen
    graphics.fillRect(boxState.x, boxState.y, 20, 20);

    // update
    boxState.x += 3;
    boxState.y += 1;
    graphics.restore();
}

function init() {
    canvas = document.getElementById("theCanvas");
    graphics = canvas.getContext("2d");
    setInterval(animate,40);
}
```

Animation Over Time

Sometimes we want to animate one motion, followed by another, etc.

- Use a counter to keep track

```
 if (t < 100) {
     // move right
     state.x += 10;
 } else if (t < 200) {
     // move down
     state.y += 10;
 } else  {
     // move up & left
     state.x -= 10;
     state.y += 10;
}
 t++;
 t = t % 300;  // reset after 300
```

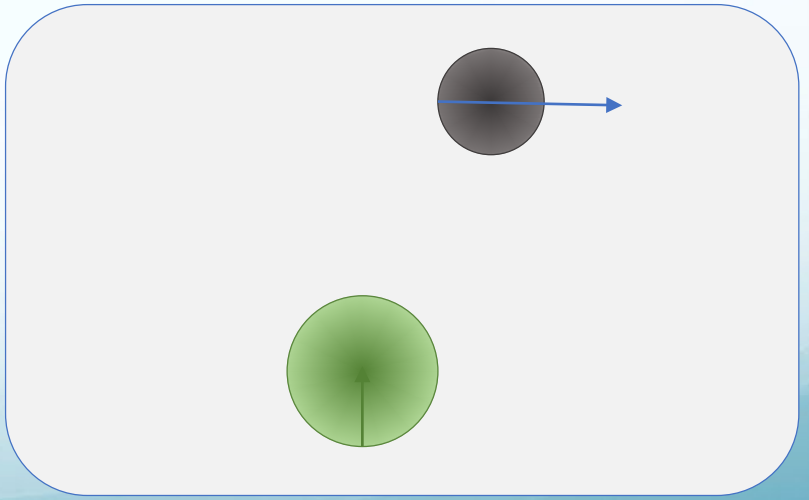


Animation with Multiple Objects

Multiple independent motions can be combined

- Each moving object has its own state defining a transformation
  Use **`save()`** and **`restore()`** to swap between transformtions
- Animation loop:
  1. Erase screen / draw background
  2. Loop over objects
     A. Install transform for the current object
     B. Draw object using current transform
     C. Update state of this object
     D. Undo the transform

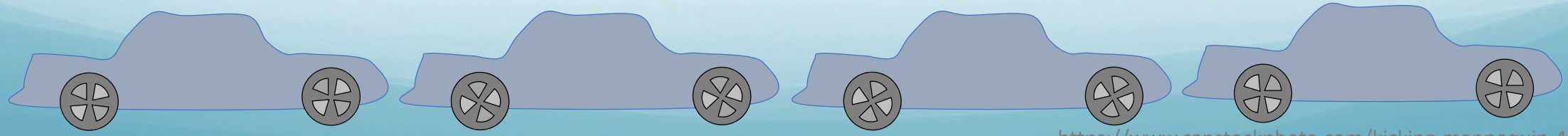
T_{BLUE}



Hierarchical Modeling

- Complex objects can be built up of subparts
- Overall object has one modeling transform
  - Subparts apply their own transform on top of the parent's
    - If they have subparts, they can apply yet another, etc.
  - After drawing each subpart, revert to the parent transform
- When animating, a change to the transform of the main object modifies all the subparts as well

Car transformation is T_c. Wheel transformation is $T_c T_w R_w$
Wheels rotate, but always have same position relative to car.
R_w and T_c update with time; T_w stays the same.



https://www.canstockphoto.com/kicking-mannequin-4280471.html

Questions

PAUSE NOW & ANSWER

A hierarchical model is used for a steam locomotive, with the boiler as the root and the hierarchy shown

1. What is the full transformation applied to driver wheel #1?

$$T_B T_{W_1} R_W$$

2. What is the full transformation applied to smoke puff #2?

$$T_B T_{S_1} S_s T_{S_2} S_s$$

3. Why can the smoke puffs use the same scale transformation but need different translations?

They grow by the same amount, but the movements are different.