

# ArcadeJS Tutorial

## Table of Content

Overview.....	1
Creating a game page.....	2
Render loop.....	3
Frames, velocities and timing.....	4
Coordinate systems and transformation pipeline.....	5
Drawing.....	6
Augmented canvas.....	6
Game play.....	7
Set up the object list.....	7
Collision detection.....	7
Scheduled events (timeout trigger).....	7
Activities.....	7
The object list.....	7
Sound.....	7
User input.....	8
Event handling.....	8
Keyboard input.....	8
Mouse input, Drag'n'drop.....	8
Touch events.....	8
Controls.....	8
Mobile devices and touch events.....	8
Debugging.....	8
Further information.....	8

## Overview

Arcade.js is a 2D game engine that drives a render loop for multiple moving objects.

Also sound, keyboard, mouse and touch events are supported.

The linear algebra math is provided by LinaJS (which comes as part of the same project), so you might want to read [LinaIntro](#) first.

API details:

<https://rawgit.com/mar10/arcade-js/master/doc/jsdocs/arcade.js/jsdoc/index.html>

Arcade.js is open source, released under the [MIT license](#).

The project home is located at <https://github.com/mar10/arcade-js/>.

## Creating a game page

To implement a game, we need

1. One HTML page that includes the required JavaScript libraries and one canvas element.  
The Arcade-JS game object is instantiated here.
2. The game code

```
<!DOCTYPE html>
<html>
<head>
  <script src="../../depends/jquery.js"></script>
  <script src="../../lina.js/lina.js"></script>
  <script src="../../arcade.js/arcade.js"></script>
  <script src="jsAsteroids.js"></script>

  <script type="text/javascript">
  $(function(){
    // Initialize the game
    var canvas = document.getElementById("gameCanvas");
    var game = new AsteroidsGame(canvas);
  });
  </script>
</head>
<body>
<canvas id="gameCanvas">This game requires HTML 5 support.</canvas>
</body>
</html>
```

## Render loop

A game is essentially an infinite sequence of scene snapshots (or 'frames').

```
While game.isRunning:
    // Set up the new scene positions
    for obj in game.object_list:
        calculate_new_object_position(obj)
        // Let object modify this
        obj.step()

    // Draw all objects
    clear_canvas()
    for obj in game.object_list:
        set_canvas_context(obj.pos, obj.orientation)
        // Let object draw itself, using modelling coordinates
        obj.draw()
```

More detailed:

```

While game.isRunning:
    // Trigger timeout event
    if game.timeout_reached:
        game.onTimeout()

    // --- Step all objects
    game.preStep()

    for obj in game.object_list:
        // Trigger timeout event
        if obj.timeout_reached:
            obj.onTimeout()

        // Calculate new position
        obj.pos += obj.velocity
        obj.orientation += obj.rotationalSpeed
        if obj.auto_wrap:
            <calculate wrapped position>

        // Let object modify this
        obj.step()

    game.postStep()

    // --- Draw all objects
    clear_canvas()
    game.preDraw(ctx)

    for obj in game.object_list:
        save_canvas_context()
        set_canvas_context(obj.pos, obj.orientation)

        // Let object draw itself, using it's own modelling coordinates
        obj.draw(ctx)

        draw_object_debug_infos()
        restore_canvas_context()

    game.postDraw(ctx)

    draw_game_debug_infos()

```

## ***Frames, velocities and timing***

Object velocities (object.velocity and object.rotationalSpeed) are defined in World Coordinate units per second.

### **Time correction**

If the requested frame rate drops due to expensive processing, the objects appear to move slower.

Optional time correction makes sure, that the perceived on-screen-speed of objects remains constant, by adjusting the  $\Delta t$  of every frame step.

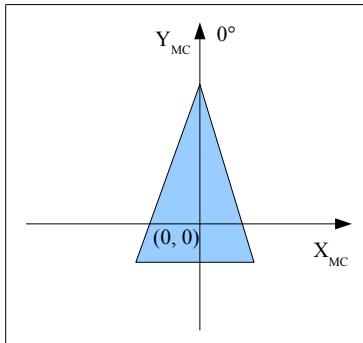
# Coordinate systems and transformation pipeline

## Modelling Coordinates ('MC')

All game objects are designed in Modelling Coordinates.

The object's neutral orientation is assumed to be upward (along the positive y-axis).

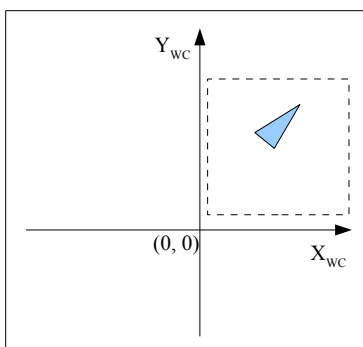
The rotation pivot should be at (0, 0).



## World Coordinates ('WC')

The game play takes place in World Coordinates with infinite dimension.

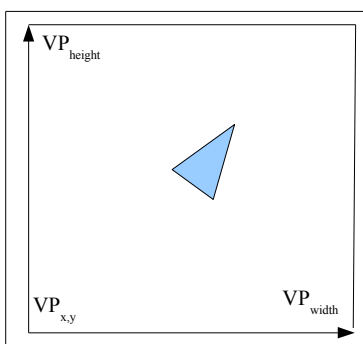
The dashed rectangle marks the part of the world is visible to the user ('Viewport').



## Viewport

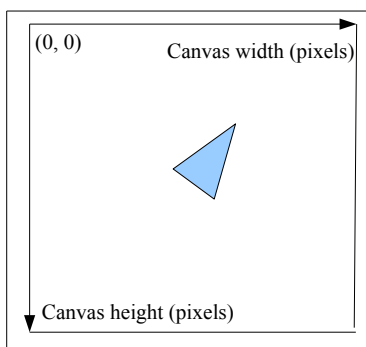
The Viewport defines the visible part of the 'world'.

It's dimensions are specified in World Coordinate units.



## Canvas Coordionates ('CC')

Finally the objects are rendered to the canvas using pixel coordinates.  
Note that the positive y-axis of the canvas points downward.



## Viewport definition

The viewport is defined using World Coordinates

```
game.setViewport(0, 0, 640, 480, "extend");
```

If the aspect ratio of the viewport and the canvas are different, some parts outside of the original viewport may be displayed in order to prevent stretching. The mapMode controls this:

- 'stretch'
- 'fit'
- 'extend'
- 'trim'
- 'none'

## Usable canvas area

Otionally the 'usable' part of the canvas can be restricted to a smaller rectangle:

```
game.setCanvasArea(10, 10, 620, 440, true);
```

In this case, the viewport is projected into this area, leaving an unused margin.

All rendering that uses World Coordinates takes place in this area.

But it is still possible to draw outside by using Canvas Coordinates.

## Drawing

### Augmented canvas

Details:

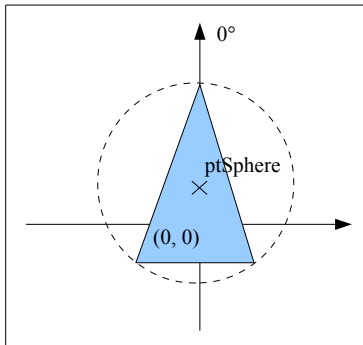
<https://rawgit.com/mar10/arcade-js/master/doc/jsdocs/arcade.js/jsdoc/symbols/ArcadeCanvas.html>

## Game play

### *Set up the object list*

### *Collision detection*

The center of the bounding sphere is not necessarily identical with the rotation pivot.



### *Scheduled events (timeout trigger)*

- `game.later()`
- `object.later()`

### *Activities*

- `onActivity()`, `setActivity()`, `isActivity()`
- `==>` API Doc

### *The object list*

### *Sound*

See

<https://rawgit.com/mar10/arcade-js/master/doc/jsdocs/arcade.js/jsdoc/symbols/AudioJS.html>

## User input

### *Event handling*

### Keyboard input

### Mouse input, Drag'n'drop

### Touch events

### *Controls*

## Mobile devices and touch events

## Debugging

Game.debug

Game.opts.debug.showVelocities = true

stoprequest 0 true

logToCanvas

## Further information

The LinaJS API is documented at

<https://rawgit.com/mar10/arcade-js/master/doc/jsdocs/lina.js/jsdoc/index.html>.

A tutorial can be found here:

[https://rawgit.com/mar10/arcade-js/master/doc/lina-js\\_tutorial.pdf](https://rawgit.com/mar10/arcade-js/master/doc/lina-js_tutorial.pdf)