Chapter 6: Moving things around

K. Jünemann
Department Informations- und Elektrotechnik
HAW Hamburg

Content

Chapter 6: Moving things around

- Moving objects using overall time *t*
- Moving objects using time increment *h*
- **`requestAnimationFrame`** and **`setInterval`**
- Browser events

Animation and frame rates

- Animation / Film: quick sequence of images called *frames*
- Frame rate: number of frames (images) per second
  - measured in *fps* (frames per second) or *Hz* (Hertz)
  - Normal films: frame rate = 24 fps
  - Computer screen: frame rate = 60 fps
    - corresponds to life time of 16,7 ms for one frame
  - Browser screen: frame rate $\leq$ 60 fps
    - depends on performance!
    - the browser tries to render a scene once every 16,7 ms!

Moving objects using overall time t

Example 1: An object moving along a straight line:

- Given:
  - Speed: $\vec{v}$
  - Initial position $\vec{x}_0$
- Position at time t:

$$\vec{x}(t) = \vec{x}_0 + \vec{v} \cdot t$$

- Time t is overall time since beginning of motion.

Moving objects using overall time *t*

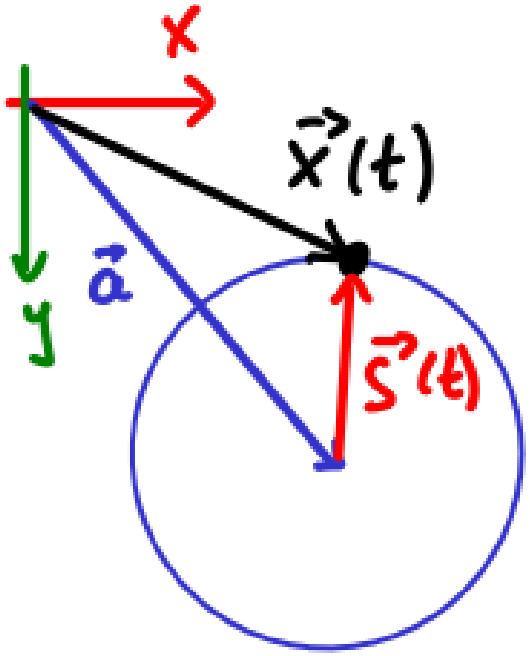
`requestAnimationFrame`: a special browser mechanism to run the *render loop*:

```
function render(tms) {
  requestAnimationFrame(render);
  // code running the application ...
}
render();
```

- `render` function repeatedly called at times *determined by browser*.
- Name of `render` function not important.
- `tms`: time (in ms) since animation started (`undefined` for first call of `render`).

Moving objects using overall time t

Example 2: Circular motion with constant angular velocity



- Given:
  - Angular velocity: ω
  - Center of circle: $\vec{a}$
  - Radius of circle: r
- Position at time t:

$$\vec{x}(t) = \vec{a} + \vec{s}(t)$$

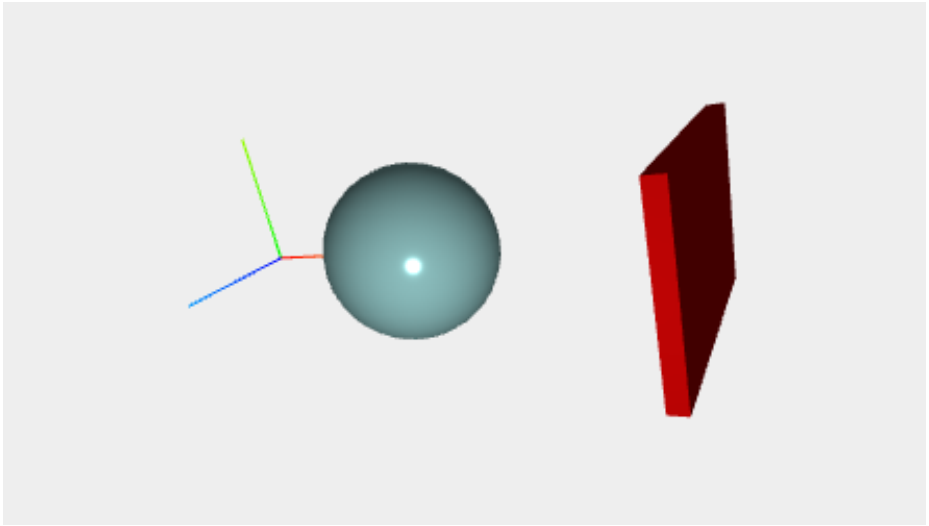
with

$$\vec{s}(t) = r \begin{pmatrix} \cos(\omega t) \\ \sin(\omega t) \end{pmatrix}$$

$\vec{s}(t)$: circular motion around origin

Moving objects using overall time t

Example 3: A sphere moves along the x-axis and is reflected off a wall.



- Approach using overall time t only works if speed is constant!
- Does not work in case of reflection, because speed changes at reflection.

Moving objects using time increment h

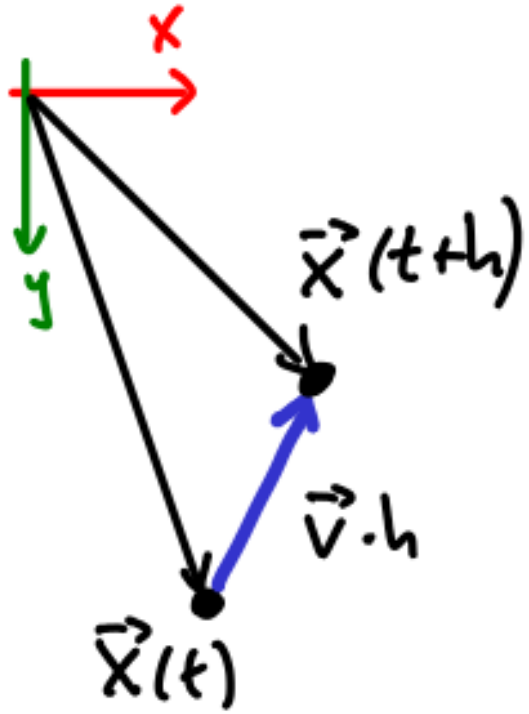
- Given:
  - Position at time t: $\vec{x}(t)$
  - Speed at time t: $\vec{v}(t)$
- h: time increment between two frames.
- Distance travelled between t and $t + h$

$$\vec{v}(t) \cdot h$$

- Position at time $t + h$:

$$\vec{x}(t + h) = \vec{x}(t) + \vec{v}(t) \cdot h$$

- This solves ode $\dot{\vec{x}}(t) = \vec{v}(t)$ with the Euler method.



Moving objects using time increment *h*

Where to get the time variables *t* and *h* from?

- Option 1: Use time variable passed to `render` function.
- Option 2: Use `THREE.Clock`:

```
const clock = new THREE.Clock();
function render() {
   requestAnimationFrame(render);
   const h = clock.getDelta();
   const t = clock.getElapsedTime();
}
render();
```

  - `t`: seconds passed since clock started.
  - `h`: seconds passed since last call to `getDelta` or `getElapsedTime`.

Note: Don't call `getDelta` right after `getElapsedTime`!

Moving objects with time increment h

General speed vector $\vec{v}$: use `THREE.Vector3`

```
let speed = new THREE.Vector3(...);

function render() {
  requestAnimationFrame(render);

  // Version with overall time
  obj.position.copy(speed.clone().multiplyScalar(t));

  // Version with time increment
  obj.position.add(speed.clone().multiplyScalar(h));
}
```

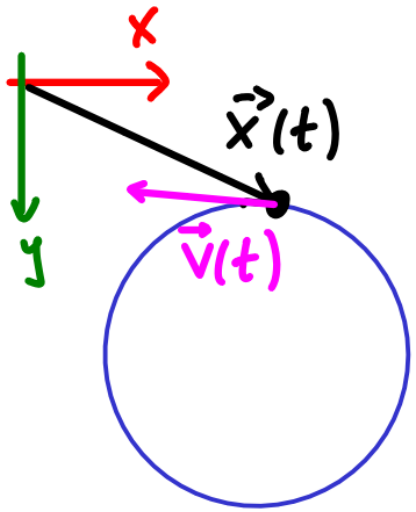
Moving objects with time increment h

Example: Circular motion again

- Calculation of speed

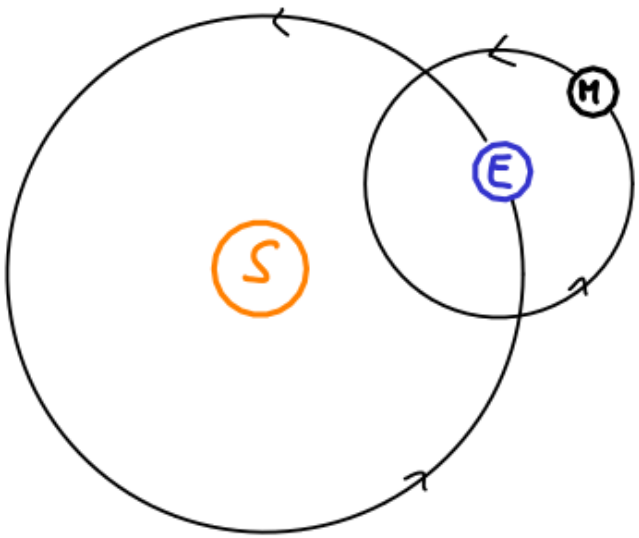
$$\vec{v}(t) = \dot{\vec{x}}(t) == \frac{\mathrm{d}}{\mathrm{dt}} r \begin{pmatrix} \cos(\omega t) \\ \sin(\omega t) \end{pmatrix} = r\omega \begin{pmatrix} -\sin(\omega t) \\ \cos(\omega t) \end{pmatrix}$$

- Initial position: $\vec{x}_0 = \vec{a} + \vec{s}(0) = \vec{a} + r \begin{pmatrix} 1 \\ 0 \end{pmatrix}$



The speed vector is tangent to circle.

Exercise 1: Earth and moon moving around sun



- Create a yellow sphere of radius 0.3 at the origin of the scene.
- Create a blue sphere of radius 0.2 at a distance of 3 from the origin rotating in the x-y-plane around the origin at a speed of 1 rotation in 15 seconds.
- Create a grey sphere of radius 0.1 at a distance of 1 from the blue sphere rotating in the x-y-plane around the blue sphere at a speed of 1 rotation in 5 seconds.

Hint: Set up a useful hierarchy of coordinate systems.

`requestAnimationFrame` and `setInterval`

The *render loop* is different from an ordinary loop!

Some features of requestAnimationFrame:

- If the browser is busy `requestAnimationFrame` may run slower than 60 fps.
- If webpage becomes invisible, animation stops!

That's why requestAnimationFrame has been invented.

`requestAnimationFrame` and `setInterval`

- *setInterval*: an alternative to requestAnimationFrame.
- allows to call functions repeatedly!

```
setInterval(callback, h);
```

- `callback`: function to be called periodically after time `h`.
- `h`: time in ms between two successive calls to `callback`.

Disadvantages of `setInterval`:

- How to choose time interval `h`?
- If callback takes longer than `h` to complete behaviour can be inconsistent.
- setInterval is periodically called even if no effect is created. (e.g. a page in a hidden tab)

Browser events

- A browser can react to certain events, e.g.
  - mouse action or keyboard input
  - window resizing
  - page loading
- Events trigger a *callback* function.
  - Browsers contain default callbacks.
  - Default can be overridden with Javascript
- Events are associated to elements of the DOM tree.
- Event types are specifed by a standard.

Browser events

Events are identified by name, e.g.

- `keydown`
- `keyup`
- `resize`

See `https://developer.mozilla.org/en-US/docs/Web/Events` for a complete list.

Register callback to entire document or any other DOM element:

```
document.addEventListener("keydown", myCallback);
```

- First argument: event name
- Second argument: event callback

Browser events

Event callback is function with signature

```
function myCallback(event) {
  // ...
}
```

- `event` contains event infos, e.g.
  - `key`: String which key has been pressed
    (used to be `keyCode` which is deprecated, but still in the videos)
  - `ctrlKey`: has Ctrl-Key been pressed
- The following code inside the handler turns off default handler:

```
event.preventDefault();
```

- There is also an old-style event handling API:

```
document.onkeydown = myCallback;
```

Example and exercise 2

Example: Create a sphere that

- moves to the right if you press the right arrow key
- moves to the left if you press the left arrow key

Exercise 2: Expand the code so that ...

1. ... the up and down arrow keys make the sphere move up and down.
2. ... the ball stops moving when the arrow keys are released (Hint: add a callback for the *keyup* event).