Basic animations

← Previous Next →

Since we're using JavaScript to control `<canvas>` elements, it's also very easy to make (interactive) animations. In this chapter we will take a look at how to do some basic animations.

Probably the biggest limitation is, that once a shape gets drawn, it stays that way. If we need to move it we have to redraw it and everything that was drawn before it. It takes a lot of time to redraw complex frames and the performance depends highly on the speed of the computer it's running on.

Basic animation steps

These are the steps you need to take to draw a frame:

1. **Clear the canvas** Unless the shapes you'll be drawing fill the complete canvas (for instance a backdrop image), you need to clear any shapes that have been drawn previously. The easiest way to do this is using the `clearRect()` method.
2. **Save the canvas state** If you're changing any setting (such as styles, transformations, etc.) which affect the canvas state and you want to make sure the original state is used each time a frame is drawn, you need to save that original state.
3. **Draw animated shapes** The step where you do the actual frame rendering.
4. **Restore the canvas state** If you've saved the state, restore it before drawing a new frame.

Controlling an animation

Shapes are drawn to the canvas by using the canvas methods directly or by calling custom functions. In normal circumstances, we only see these results appear on the canvas when the script finishes executing. For instance, it isn't possible to do an animation from within a `for` loop.

That means we need a way to execute our drawing functions over a period of time. There are two ways to control an animation like this.

Scheduled updates

First there's the `setInterval()`, `setTimeout()`, and `requestAnimationFrame()` functions, which can be used to call a specific function over a set period of time.

`setInterval()`

Starts repeatedly executing the function specified by `function` every `delay` milliseconds.

`setTimeout()`

Executes the function specified by `function` in `delay` milliseconds.

`requestAnimationFrame()`

Tells the browser that you wish to perform an animation and requests that the browser call a specified function to update an animation before the next repaint.

If you don't want any user interaction you can use the `setInterval()` function, which repeatedly executes the supplied code. If we wanted to make a game, we could use keyboard or mouse events to control the animation and use `setTimeout()`. By setting listeners using `addEventListener()`, we catch any user interaction and execute our animation functions.

Note: In the examples below, we'll use the `Window.requestAnimationFrame()` method to control the animation. The `requestAnimationFrame` method provides a smoother and more efficient way for animating by calling the animation frame when the system is ready to paint the frame. The number of callbacks is usually 60 times per second and may be reduced to a lower rate when running in background tabs. For more information about the animation loop, especially for games, see the article Anatomy of a video game in our Game development zone.

An animated solar system

This example animates a small model of our solar system.

HTML

HTML

```html
<canvas id="canvas" width="300" height="300"></canvas>
```

JavaScript

JS

```js
const sun = new Image();
const moon = new Image();
const earth = new Image();
const ctx = document.getElementById("canvas").getContext("2d");

function init() {
  sun.src = "canvas_sun.png";
  moon.src = "canvas_moon.png";
  earth.src = "canvas_earth.png";
  window.requestAnimationFrame(draw);
}
```

```
function draw() {
  ctx.globalCompositeOperation = "destination-over";
  ctx.clearRect(0, 0, 300, 300); // clear canvas

  ctx.fillStyle = "rgb(0 0 0 / 40%)";
  ctx.strokeStyle = "rgb(0 153 255 / 40%)";
  ctx.save();
  ctx.translate(150, 150);

  // Earth
  const time = new Date();
  ctx.rotate(
    ((2 * Math.PI) / 60) * time.getSeconds() +
      ((2 * Math.PI) / 60000) * time.getMilliseconds(),
  );
  ctx.translate(105, 0);
  ctx.fillRect(0, -12, 40, 24); // Shadow
  ctx.drawImage(earth, -12, -12);

  // Moon
  ctx.save();
  ctx.rotate(
    ((2 * Math.PI) / 6) * time.getSeconds() +
      ((2 * Math.PI) / 6000) * time.getMilliseconds(),
  );
  ctx.translate(0, 28.5);
  ctx.drawImage(moon, -3.5, -3.5);
  ctx.restore();

  ctx.restore();

  ctx.beginPath();
  ctx.arc(150, 150, 105, 0, Math.PI * 2, false); // Earth orbit
  ctx.stroke();

  ctx.drawImage(sun, 0, 0, 300, 300);

  window.requestAnimationFrame(draw);
}

init();
```

Result

An animated clock

This example draws an animated clock, showing your current time.

HTML

HTML

```html
<canvas id="canvas" width="150" height="150">The current time</canvas>
```

JavaScript

JS

```js
function clock() {
  const now = new Date();
  const canvas = document.getElementById("canvas");
  const ctx = canvas.getContext("2d");
  ctx.save();
  ctx.clearRect(0, 0, 150, 150);
  ctx.translate(75, 75);
  ctx.scale(0.4, 0.4);
  ctx.rotate(-Math.PI / 2);
  ctx.strokeStyle = "black";
  ctx.fillStyle = "white";
  ctx.lineWidth = 8;
  ctx.lineCap = "round";

  // Hour marks
  ctx.save();
  for (let i = 0; i < 12; i++) {
    ctx.beginPath();
    ctx.rotate(Math.PI / 6);
    ctx.moveTo(100, 0);
    ctx.lineTo(120, 0);
    ctx.stroke();
  }
  ctx.restore();

  // Minute marks
  ctx.save();
  ctx.lineWidth = 5;
  for (let i = 0; i < 60; i++) {
    if (i % 5 !== 0) {
      ctx.beginPath();
      ctx.moveTo(117, 0);
      ctx.lineTo(120, 0);
      ctx.stroke();
    }
    ctx.rotate(Math.PI / 30);
  }
  ctx.restore();

  const sec = now.getSeconds();
  // To display a clock with a sweeping second hand, use:
  // const sec = now.getSeconds() + now.getMilliseconds() / 1000;
  const min = now.getMinutes();
  const hr = now.getHours() % 12;

  ctx.fillStyle = "black";

  // Write image description
  canvas.innerText = `The time is: ${hr}:${min}`;

  // Write Hours
  ctx.save();
  ctx.rotate(
    (Math.PI / 6) * hr + (Math.PI / 360) * min + (Math.PI / 21600) * sec,
  );
  ctx.lineWidth = 14;
  ctx.beginPath();
  ctx.moveTo(-20, 0);
  ctx.lineTo(80, 0);
  ctx.stroke();
```

```
    ctx.restore();

    // Write Minutes
    ctx.save();
    ctx.rotate((Math.PI / 30) * min + (Math.PI / 1800) * sec);
    ctx.lineWidth = 10;
    ctx.beginPath();
    ctx.moveTo(-28, 0);
    ctx.lineTo(112, 0);
    ctx.stroke();
    ctx.restore();

    // Write seconds
    ctx.save();
    ctx.rotate((sec * Math.PI) / 30);
    ctx.strokeStyle = "#D40000";
    ctx.fillStyle = "#D40000";
    ctx.lineWidth = 6;
    ctx.beginPath();
    ctx.moveTo(-30, 0);
    ctx.lineTo(83, 0);
    ctx.stroke();
    ctx.beginPath();
    ctx.arc(0, 0, 10, 0, Math.PI * 2, true);
    ctx.fill();
    ctx.beginPath();
    ctx.arc(95, 0, 10, 0, Math.PI * 2, true);
    ctx.stroke();
    ctx.fillStyle = "transparent";
    ctx.arc(0, 0, 3, 0, Math.PI * 2, true);
    ctx.fill();
    ctx.restore();

    ctx.beginPath();
    ctx.lineWidth = 14;
    ctx.strokeStyle = "#325FA2";
    ctx.arc(0, 0, 142, 0, Math.PI * 2, true);
    ctx.stroke();

    ctx.restore();

    window.requestAnimationFrame(clock);
  }

  window.requestAnimationFrame(clock);
```

Result

Note: Although the clock updates only once every second, the animated image is updated at 60 frames per second (or at the display refresh rate of your web browser). To display the clock with a sweeping second hand, replace the definition of `const sec` above with the version that has been commented out.

A looping panorama

In this example, a panorama is scrolled left-to-right. We're using an image of Yosemite National Park we took from Wikipedia, but you could use any image that's larger than the canvas.

HTML

The HTML includes the `<canvas>` in which the image is scrolled. Note that the width and height specified here must match the values of the `canvasXSize` and `canvasYSize` variables in the JavaScript code.

HTML

```html
<canvas id="canvas" width="800" height="200"
  >Yosemite National Park, meadow at the base of El Capitan</canvas
>
```

JavaScript

JS

```js
const img = new Image();

// User Variables - customize these to change the image being scrolled, its
// direction, and the speed.
img.src = "capitan_meadows_yosemite_national_park.jpg";
const canvasXSize = 800;
const canvasYSize = 200;
const speed = 30; // lower is faster
const scale = 1.05;
const y = -4.5; // vertical offset

// Main program
const dx = 0.75;
let imgW;
let imgH;
let x = 0;
let clearX;
let clearY;
let ctx;

img.onload = () => {
  imgW = img.width * scale;
  imgH = img.height * scale;

  if (imgW > canvasXSize) {
    // Image larger than canvas
    x = canvasXSize - imgW;
  }

  // Check if image dimension is larger than canvas
  clearX = Math.max(imgW, canvasXSize);
  clearY = Math.max(imgH, canvasYSize);

  // Get canvas context
  ctx = document.getElementById("canvas").getContext("2d");

  // Set refresh rate
  return setInterval(draw, speed);
};

function draw() {
  ctx.clearRect(0, 0, clearX, clearY); // clear the canvas

  // If image is <= canvas size
  if (imgW <= canvasXSize) {
    // Reset, start from beginning
    if (x > canvasXSize) {
      x = -imgW + x;
    }

    // Draw additional image1
    if (x > 0) {
      ctx.drawImage(img, -imgW + x, y, imgW, imgH);
    }

    // Draw additional image2
```

```
      if (x - imgW > 0) {
        ctx.drawImage(img, -imgW * 2 + x, y, imgW, imgH);
      }
    } else {
      // Image is > canvas size
      // Reset, start from beginning
      if (x > canvasXSize) {
        x = canvasXSize - imgW;
      }

      // Draw additional image
      if (x > canvasXSize - imgW) {
        ctx.drawImage(img, x - imgW + 1, y, imgW, imgH);
      }
    }

    // Draw image
    ctx.drawImage(img, x, y, imgW, imgH);

    // Amount to move
    x += dx;
  }
```

Result

Mouse following animation

HTML

HTML

```html
<canvas id="cw"
  >Animation creating multi-colored disappearing stream of light that follow the
  cursor as it moves over the image
</canvas>
```

CSS

CSS

```css
#cw {
  position: fixed;
  z-index: -1;
}

body {
  margin: 0;
  padding: 0;
  background-color: rgb(0 0 0 / 5%);
}
```

JavaScript

JS

```js
const canvas = document.getElementById("cw");
const context = canvas.getContext("2d");
context.globalAlpha = 0.5;

const cursor = {
  x: innerWidth / 2,
  y: innerHeight / 2,
};

let particlesArray = [];

generateParticles(101);
setSize();
anim();

addEventListener("mousemove", (e) => {
  cursor.x = e.clientX;
  cursor.y = e.clientY;
});
```

```
addEventListener(
  "touchmove",
  (e) => {
    e.preventDefault();
    cursor.x = e.touches[0].clientX;
    cursor.y = e.touches[0].clientY;
  },
  { passive: false },
);

addEventListener("resize", () => setSize());

function generateParticles(amount) {
  for (let i = 0; i < amount; i++) {
    particlesArray[i] = new Particle(
      innerWidth / 2,
      innerHeight / 2,
      4,
      generateColor(),
      0.02,
    );
  }
}

function generateColor() {
  let hexSet = "0123456789ABCDEF";
  let finalHexString = "#";
  for (let i = 0; i < 6; i++) {
    finalHexString += hexSet[Math.ceil(Math.random() * 15)];
  }
  return finalHexString;
}

function setSize() {
  canvas.height = innerHeight;
  canvas.width = innerWidth;
}

function Particle(x, y, particleTrailWidth, strokeColor, rotateSpeed) {
  this.x = x;
  this.y = y;
  this.particleTrailWidth = particleTrailWidth;
  this.strokeColor = strokeColor;
  this.theta = Math.random() * Math.PI * 2;
  this.rotateSpeed = rotateSpeed;
  this.t = Math.random() * 150;

  this.rotate = () => {
    const ls = {
      x: this.x,
      y: this.y,
    };
    this.theta += this.rotateSpeed;
    this.x = cursor.x + Math.cos(this.theta) * this.t;
    this.y = cursor.y + Math.sin(this.theta) * this.t;
    context.beginPath();
    context.lineWidth = this.particleTrailWidth;
    context.strokeStyle = this.strokeColor;
    context.moveTo(ls.x, ls.y);
    context.lineTo(this.x, this.y);
    context.stroke();
  };
}

function anim() {
  requestAnimationFrame(anim);

  context.fillStyle = "rgb(0 0 0 / 5%)";
  context.fillRect(0, 0, canvas.width, canvas.height);

  particlesArray.forEach((particle) => particle.rotate());
}
```

Result