

Remotion Complete Documentation

Make videos programmatically using React

Comprehensive documentation scraped from <https://www.remotion.dev/docs/>

Table of Contents

- Getting Started
 - The Fundamentals
 - Designing Visuals
 - Embedding Videos
 - Audio
 - Parameterized Videos
 - Rendering
 - Studio
 - Server-Side Rendering (SSR)
 - Client-Side Rendering
 - Remotion Player
 - AWS Lambda
 - Cloud Run
 - Media Parser
 - Building Apps
 - AI Integration
 - Captions
 - Tooling
 - Troubleshooting
 - Terminology
 - Migration Guides
 - Miscellaneous
-

Getting Started

Prerequisites

To use Remotion, you need at least **Node 16** or **Bun 1.0.3**.

Linux Requirements:

- Linux distros need at least version 2.35 of Libc
- Additional packages must be installed
- Alpine Linux and nixOS are unsupported

Prompting Videos with Claude Code

If you plan on prompting videos with **Claude Code**, see the dedicated section on AI integration.

Scaffolding a New Project

Use one of the following commands based on your package manager:

npm:

```
npx create-video@latest
```

pnpm:

```
pnpm create video
```

Yarn:

```
yarn create video
```

Bun:

```
bun create video
```

Bun as a runtime is mostly supported.

Template Selection

Choose the template that is most suitable for you. For your first project, we recommend the **Hello World template**.

Available templates include:

- Regular templates
- Next.js + React Router 7 templates

Starting Remotion Studio

After the project has been scaffolded, open the project in your text editor and start the Remotion Studio:

For regular templates:

```
npm run dev
```

For Next.js and React Router 7 templates:

```
npm run remotion
```

This is a shorthand for the studio command of the Remotion CLI:

```
npx remotion studio
```

A server will be started on port 3000 (or a higher port if it isn't available) and the Remotion Studio should open in the browser.

Note: In older projects, npm start was used over npm run dev.

Installation in Existing Projects

Want to install Remotion in an existing project? See the dedicated installation guide for existing projects.

The Fundamentals

React Components

The idea of Remotion is to give you a frame number and a blank canvas, to which you can render anything you want using React. This is an example React component that renders the current frame as text:

MyComposition.tsx

```
import { AbsoluteFill, useCurrentFrame } from "remotion";

export const MyComposition = () => {
  const frame = useCurrentFrame();

  return (
    <AbsoluteFill
      style={{
        justifyContent: "center",
        alignItems: "center",
        fontSize: 100,
        backgroundColor: "white",
      }}
    >
    The current frame is {frame}.
    </AbsoluteFill>
  );
}
```

A video is a function of images over time. If you change content every frame, you'll end up with an animation.

Video Properties

A video has 4 properties:

- **width** - in pixels
- **height** - in pixels
- **durationInFrames** - the total number of frames in the video
- **fps** - framerate of the video

You can obtain these values from the `useVideoConfig()` hook:

```
import { AbsoluteFill, useVideoConfig } from 'remotion';

export const MyComposition = () => {
  const { fps, durationInFrames, width, height } = useVideoConfig();
```

```
return (
<AbsoluteFill
style={{
justifyContent: 'center',
alignItems: 'center',
fontSize: 60,
backgroundColor: 'white',
}}
>
This {width}x{height}px video is {durationInFrames / fps} seconds long.
</AbsoluteFill>
);
};
```

A video's first frame is 0 and its last frame is durationInFrames - 1.

Compositions

A composition is the combination of a React component and video metadata.

By rendering a <Composition> component in src/Root.tsx, you can register a renderable video and make it visible in the Remotion sidebar.

src/Root.tsx

```
export const RemotionRoot: React.FC = () => {
return (
);
```

You can register multiple compositions in src/Root.tsx by wrapping them in a React Fragment:

```
<>
```

```
</>
```

Animating Properties

Animation works by changing properties over time.

Basic Fade In Animation

If we want to fade the text in over 60 frames, we need to gradually change the opacity over time so that it goes from 0 to 1.

FadeIn.tsx

```
export const FadeIn = () => {
const frame = useCurrentFrame();
const opacity = Math.min(1, frame / 60);
```

```
return (
<AbsoluteFill
style={{
justifyContent: "center",
alignItems: "center",
backgroundColor: "white",
fontSize: 80,
}}
>
<div style={{ opacity: opacity }}>Hello World!</div>
</AbsoluteFill>
);
};
```

Using the interpolate() Helper Function

Using the `interpolate()` function can make animations more readable. The above animation can also be written as:

```
import { interpolate } from "remotion";

const opacity = interpolate(
frame,
[0, 60], // Input range
[0, 1], // Output range
{
extrapolateRight: "clamp",
}
);
```

In this example, we map the frames 0 to 60 to their opacity values (0, 0.0166, 0.033, 0.05 ...) and use the `extrapolateRight` setting to clamp the output so that it never becomes bigger than 1.

Using Spring Animations

Spring animations are a natural animation primitive. By default, they animate from 0 to 1 over time. This time, let's animate the scale of the text.

```
import { spring, useCurrentFrame, useVideoConfig } from "remotion";

export const MyVideo = () => {
const frame = useCurrentFrame();
const { fps } = useVideoConfig();

const scale = spring({
fps,
frame,
});

return (
<div
style={{

```

```
flex: 1,  
textAlign: "center",  
fontSize: "7em",  
}  
>  
<div style={{ transform: scale(${scale}) }}>Hello World!</div>  
</div>  
);  
};
```

You should see the text jump in.

The default spring configuration leads to a little bit of overshoot, meaning the text will bounce a little bit. See the documentation page for `spring()` to learn how to customize it.

Always Animate Using `useAnimationFrame()`

Watch out for flickering issues during rendering that arise if you write animations that are not driven by `useAnimationFrame()` – for example CSS transitions.

Read more about how Remotion's rendering works - understanding it will help you avoid issues down the road.

Making Components Reusable

React components allow us to encapsulate video logic and reuse the same visuals multiple times.

Consider a title - to make it reusable, factor it out into its own component:

`MyComposition.tsx`

```
import { AbsoluteFill, interpolate, useAnimationFrame } from 'remotion';

const Title: React.FC<{ title: string }> = ({ title }) => {
  const frame = useAnimationFrame();
  const opacity = interpolate(
    frame,
    [0, 20],
    [0, 1],
    {
      extrapolateRight: 'clamp',
    }
  );

  return (
    <div style={{ opacity, textAlign: 'center', fontSize: '7em' }}>
      {title}
    </div>
  );
};
```

```
export const MyVideo = () => {
  return (
    );
  };
};
```

Using Sequences

To render multiple instances of the title, duplicate the <Title> component.

You can also use the <Sequence> component to limit the duration of the first title and time-shift the appearance of the second title.

```
import { Sequence } from 'remotion';

export const MyVideo = () => {
  return (
    );
  };
};
```

You should see two titles appearing after each other.

Note that the value of `useCurrentFrame()` has been shifted in the second instance, so that it returns 0 only when the absolute time is 40. Before that, the sequence was not mounted at all.

Sequences by default are absolutely positioned - you can use `layout="none"` to make them render like a regular <div>.

Preview Your Video

You can preview your video by starting the Remotion Studio:

Regular templates:

`npm run dev`

Next.js and React Router 7 templates:

`npm run remotion`

This is a shorthand for the studio command of the Remotion CLI:

`npx remotion studio`

A server will be started on port 3000 (or a higher port if it isn't available) and the Remotion Studio should open in the browser.

Transforms

Animation occurs when the visual properties of an element transform over time. Let's look at five common ways to transform an element.

The 5 Basic Transformations

From left to right: **Opacity, Scale, Skew, Translate, Rotate**

Opacity

The opacity determines how visible the element is.

- 0 means fully invisible
- 1 means fully visible
- Values inbetween will make the element semi-transparent

You can set the opacity of an element using the opacity property.

MyComponent.tsx

```
<div  
style={{  
height: 100,  
width: 100,  
backgroundColor: 'red',  
opacity: 0.5,  
}}  
/>
```

Values below 0 and above 1 are accepted, but have no further effect.

Scale

The scale determines how big an element is.

- 1 is the natural size
- 2 will make the element twice as tall and wide
- Values below 1 will make the element smaller
- 0 makes the element invisible
- Values below 0 are accepted and will make the element bigger again, but mirrored

You can set the scale of an element using the scale property.

MyComponent.tsx

```
<div  
style={{  
height: 100,  
width: 100,  
backgroundColor: 'red',  
scale: 0.5,  
}}  
/>
```

The difference to changing the size of the element using height and width is that using scale() will not change the layout of the other elements.

Skew

Skewing an element will lead to a distorted appearance as if the element has been stretched on two corners of the element. Skew takes an angle that can be specified using rad (radians) and deg (degrees).

You can set the skew of an element using the transform property.

MyComponent.tsx

```
<div  
style={{  
height: 100,  
width: 100,  
backgroundColor: 'red',  
transform: skew(20deg),  
}}  
/>
```

Translate

Translating an element means moving it. A translation can be done on the X, Y or even Z axis. The transformation can be specified in px.

You can set the translation of an element using the transform property.

MyComponent.tsx

```
<div  
style={{  
height: 100,  
width: 100,  
backgroundColor: 'red',  
transform: translateX(100px),  
}}  
/>
```

As opposed to changing the position of an element using margin-top and margin-left, using translate() will not change the position of the other elements.

Rotate

By rotating an element, you can make it appear as if it has been turned around its center. The rotation can be specified in rad (radians) or deg (degrees) and you can rotate an element around the Z axis (the default) but also around the X and Y axis.

You can set the translation of an element using the transform property.

MyComponent.tsx

```
<div  
style={{  
height: 100,  
width: 100,
```

```
backgroundColor: 'red',
transform: rotate(45deg), // the same as rotateZ(45deg)
}}
/>>
```

If you want to rotate an element around the X or Y axis, you should apply the perspective property to the parent element.

If you don't want to rotate around the center, you can use the transform-origin property to change the origin of the rotation.

Note: When rotating SVG elements, the transform origin is the top left corner by default. You can get the same behavior as for the other elements by adding style={{transformBox: 'fill-box', transformOrigin: 'center center'}}.

Multiple Transformations

Oftentimes, you want to combine multiple transformations. If they use different CSS properties like transform and opacity, simply specify both properties in the style object.

If both transformations use the transform property, specify multiple transformations separated by a space.

MyComponent.tsx

```
<div
style={{
height: 100,
width: 100,
backgroundColor: 'red',
transform: translateX(100px) scale(2),
}}
/>
```

Note that the order matters. The transformations are applied in the order they are specified.

Using the makeTransform() Helper

Install `@remotion/animation-utils` to get a type-safe helper function to generate transform strings.

```
import { makeTransform, rotate, translate } from '@remotion/animation-utils';

const transform = translate(50, 50);
// => "translate(50px, 50px)"

const multiTransform = makeTransform([
rotate(45),
translate(50, 50)
]);
// => "rotate(45deg) translate(50px, 50px)"
```

More Ways to Transform Objects

These are just some of the basic transformations. Here are some more transformations that are possible:

- The height and width of a <div>
 - The rounded edges of an element using border-radius
 - The shadow of an element using box-shadow
 - The color of something using color and interpolateColors()
 - The evolution of a SVG path using evolvePath()
 - The weight and slant of a dynamic font
 - The stops of a linear-gradient
 - The values of a CSS filter()
-

Importing Assets

To import assets in Remotion, create a public/ folder in your project and use staticFile() to import it.

Project structure:

```
my-video/
  └── node_modules/
  └── public/
    └── logo.png
  └── src/
    ├── MyComp.tsx
    ├── Root.tsx
    └── index.ts
  └── package.json
```

src/MyComp.tsx

```
import { Img, staticFile } from 'remotion';

export const MyComp: React.FC = () => {
  return <Img src={staticFile('logo.png')} />;
};
```

Using Images

Use the tag from Remotion.

MyComp.tsx

```
import { Img, staticFile } from 'remotion';

export const MyComp: React.FC = () => {
  return <Img src={staticFile('logo.png')} />;
};
```

You can also pass a URL:

MyComp.tsx

```
import { Img } from 'remotion';
```

```
export const MyComp: React.FC = () => {
  return
```



```
;  
};
```

Using Image Sequences

If you have a series of images, for example exported from another program like After Effects or Rotato, you can interpolate the path to create a dynamic import.

Project structure:

```
my-video/  
  └── public/  
    └── frame1.png  
    └── frame2.png  
    └── frame3.png  
  └── package.json
```

```
import { Img, staticFile, useCurrentFrame } from 'remotion';

const MyComp: React.FC = () => {
  const frame = useCurrentFrame();
  return <Img src={staticFile(`/frame${frame}.png`)} />;
};
```

Using Videos

Use the `<OffthreadVideo />`, `<Video />` or `<Html5Video />` component to keep the timeline and your video in sync.

```
import { OffthreadVideo, staticFile } from 'remotion';

export const MyComp: React.FC = () => {
  return <OffthreadVideo src={staticFile('vid.webm')} />;
};
```

Loading videos via URL is also possible:

```
import { OffthreadVideo } from 'remotion';

export const MyComp: React.FC = () => {
  return (
    );
  };
}
```

See also: Which video formats does Remotion support?

Using Audio

Use the <Html5Audio> component.

```
import { Html5Audio, staticFile } from 'remotion';

export const MyComp: React.FC = () => {
  return <Html5Audio src={staticFile('tune.mp3')} />;
};
```

Loading audio from an URL is also possible:

```
import { Html5Audio } from 'remotion';

export const MyComp: React.FC = () => {
  return (
    );
  };
}
```

See the audio guide for guidance on including audio.

Using CSS

Put the .css file alongside your JavaScript source files and use an import statement.

Project structure:

```
my-video/
  └── node_modules/
  └── src/
    └── style.css
    └── MyComp.tsx
    └── Root.tsx
    └── index.ts
  └── package.json
```

MyComp.tsx

```
import './style.css';
```

Want to use SASS, Tailwind or similar? See examples on how to override the Webpack configuration.

Using Fonts

Read the separate page for fonts.

Import Statements

As an alternative way to import files, Remotion allows you to import or require() several types of files in your project:

- Images (.png, .svg, .jpg, .jpeg, .webp, .gif, .bmp)
- Videos (.webm, .mov, .mp4)
- Audio (.mp3, .wav, .aac, .m4a)
- Fonts (.woff, .woff2, .otf, .ttf, .eot)

For example:

MyComp.tsx

```
import { Img } from 'remotion';
import logo from './logo.png';

export const MyComp: React.FC = () => {
  return ;
};
```

Caveats

While this was previously the main way of importing files, we now recommend against it because of the following reasons:

- Only the above listed file extensions are supported
- The maximum file size is 2GB
- Dynamic imports such as require('img' + frame + '.png') are funky

Prefer importing using staticFile() if possible.

Important Notes

Use Remotion Components:

- Use or <Gif /> over the native tag, <Image> from Next.js and CSS background-image
- Use <OffthreadVideo />, <Video /> or <Html5Video /> over the native <video> tag
- Use <Audio /> or <Html5Audio /> over the native <audio> tag
- Use <IFrame /> over the native <iframe> tag

By using the components from Remotion, you ensure that:

1. The assets are fully loaded before the the frame is rendered
 2. The images and videos are synchronized with Remotion's timeline
-

Layers

Unlike normal websites, a video has fixed dimensions. That means, it is okay to use position: "absolute"!

In Remotion, you often want to "layer" elements on top of each other.

This is a common pattern in video editors, and in Photoshop.

An easy way to do it is using the <AbsoluteFill> component:

MyComp.tsx

```
import React from 'react';
import { AbsoluteFill, Img, staticFile } from 'remotion';

export const MyComp: React.FC = () => {
  return (
    <Img src={staticFile('bg.png')} />
```

This text appears on top of the video!

```
);  
};
```

This will render the text on top of the image.

If you want to only show an element for a certain duration, you can use a <Sequence> component, which by default is also absolutely positioned.

MyComp.tsx

```
import React from 'react';
import { AbsoluteFill, Img, staticFile, Sequence } from 'remotion';

export const MyComp: React.FC = () => {
  return (
    <Img src={staticFile('bg.png')} />
```

This text appears after 60 frames!

```
);  
};
```

The lower the absolutely positioned element is in the tree, the higher it will be in the layer stack.

If you are aware of this behavior, you can use it to your advantage and avoid using z-index in most cases.

Transitions

To transition between two types of absolutely positioned content, you can use the `<TransitionSeries>` component.

SlideTransition.tsx

```
import { linearTiming, TransitionSeries } from "@remotion/transitions";
import { slide } from "@remotion/transitions/slide";

const BasicTransition = () => {
  return (
    <TransitionSeries.Sequence durationInFrames={40}>
      A
    </TransitionSeries.Sequence>
    <TransitionSeries.Transition
      presentation={slide()}
      timing={linearTiming({ durationInFrames: 30 })}>
    />
    <TransitionSeries.Sequence durationInFrames={60}>
      B
    </TransitionSeries.Sequence>
  );
}
```

Enter and Exit Animations

You don't necessarily have to use `<TransitionSeries>` for transitions between scenes. You can also use it to animate the entrance or exit of a scene by putting a transition first or last in the `<TransitionSeries>`.

Duration

In the above example, A is visible for 40 frames, B for 60 frames and the duration of the transition is 30 frames.

During this time, both slides are rendered. This means the total duration of the animation is $60 + 40 - 30 = 70$.

Presentations

A presentation determines the visual of animation.

Built-in Presentations:

- `fade()`
- `slide()`
- `wipe()`
- `flip()`
- `clockWipe()`
- `iris()`

- `cube()` (Paid)
- `none()`

Custom presentations are also available.

Timings

A timing determines how long the animation takes and the animation curve.

- `springTiming()` - `spring()`
- `linearTiming()`
- Custom timings

Getting the Duration of a Transition

You can get the duration of a transition by calling `getDurationInFrames()` on the timing:

Assuming a framerate of 30fps:

```
import { springTiming } from "@remotion/transitions";  
  
springTiming({ config: { damping: 200 } }).getDurationInFrames({ fps: 30 }); // 23
```

Using Audio

Remotion provides powerful capabilities for working with audio in your compositions.

Audio Features:

- **Importing audio** - Add audio to your compositions
- **Delaying audio** - Delay the start time of audio elements
- **Creating audio from video** - Use audio from videos
- **Controlling volume** - Control the volume of audio elements
- **Muting audio** - Mute audio elements
- **Controlling speed** - Change the speed of audio elements
- **Controlling pitch** - Change the pitch of audio elements
- **Visualizing audio** - Visualize audio elements
- **Exporting audio** - Export audio
- **Order of operations** - Control the order of operations for audio elements

Rendering

There are various ways to render your video.

Remotion Studio

To render a video, click the "Render" button.

Choose your preferred settings, then confirm using the Render video button.

Remotion Studio Deployment

It's possible to deploy the Remotion Studio onto a long-running server in the cloud, which can then be accessed by your non-technical team members using just a URL. Check out the Deploy the Remotion Studio guide to learn how to do this.

CLI

Render a video using render CLI command:

```
npx remotion render HelloWorld
```

Modify the composition ID to select a different video to render, or add an output path at the end if you want to override the default.

You can leave out the composition name and a picker will be shown:

```
npx remotion render
```

SSR

Remotion has a full-featured server-side rendering API. Read more about it on the server-side rendering API.

AWS Lambda

Check out Remotion Lambda.

GitHub Actions

You can also render a video using a GitHub Action.

Google Cloud Run

An alpha version of Remotion for Cloud Run is available.

We plan to change it in the future so it shares a runtime with Remotion Lambda.

Rendering Variants

Audio-only:

Instead of rendering a video, you can also just export the audio.

Image Sequence:

Instead of encoding as a video, you can use the --sequence command to output a series of image.

Still images:

If you want a single image, you can do so using the CLI or Node.JS API.

GIF:

See: Render as GIF

Transparent videos:

See: Creating overlays

Encoding Guide

Backed by FFmpeg, Remotion allows you to configure a variety of encoding settings. The goal of this page is to help you navigate through the settings and to help you choose the right one.

Choosing a Codec

Remotion supports 5 video codecs: h264 (*default*), h265, vp8, vp9 and prores. While H264 will work well in most cases, sometimes it's worth going for a different codec. Refer to the table below to see the advantages and drawbacks of each codec.

Codec	File extension	File size	Encoding time	Browser compatibility	Hardware acceleration
H.264	.mp4, .mov or .mkv	Medium	Very fast	Very good	No
H.265	.mp4 or .hevc	Medium	Fast	Very poor	No
VP8	.webm	Small	Slow	Okay	No
VP9	.webm	Very small	Very slow	Okay	No
ProRes	.mov	Large	Fast	None	On macOS

Table 1: Codec comparison

You can set a config using `Config.setCodec()` in the config file or the `--codec` CLI flag.

Hardware acceleration is available from Remotion 4.0.228.

Controlling Quality Using the CRF Setting

Applies only to h264, h265, vp8 and vp9.

No matter which codec you end up using, there's always a tradeoff between file size and video quality. You can control it by setting the so called CRF (Constant Rate Factor). The **lower the number, the better the quality**, the higher the number, the smaller the file is – of course at the cost of quality.

Be cautious: Every codec has its own range of acceptable values and a different default. So while 23 will look very good on a H264 video, it will look terrible on a WebM video. Use this chart to determine which CRF value to use:

Codec	Minimum - Best quality	Maximum - Best compression	Default
H264	1	51	18
H265	0	51	23
VP8	4	63	9
VP9	0	63	28

Table 2: CRF values by codec

You can set a CRF in the config file using the `Config.setCrf()` function or use the `--crf` command line flag.

If you enable hardware acceleration, you cannot set a crf.

Controlling Quality Using Video and Audio Bitrate

Use the following options to set the video and audio bitrate:

- In the Studio: Set video and audio bitrate in the Render Dialog
- In the CLI: Use the `--video-bitrate` and `--audio-bitrate` flags
- In SSR, Lambda and Cloud Run APIs: Use the `videoBitrate` and `audioBitrate` options

This option is incompatible with other quality options.

Controlling Quality Using ProRes Profile

Applies only to prores codec.

For ProRes, there is no CRF option, but there are profiles which you can set using the `--prores-profile` flag or the `setProResProfile` config file option.

Value	FFmpeg setting	Bitrate	Supports alpha channel
"proxy"	0	~45Mbps	No
"light"	1	~102Mbps	No
""	2	~147Mbps	No
"hq"	3	~220Mbps	No
"4444"	4	~330Mbps	Yes
"4444-xq"	4	~500Mbps	Yes

Table 3: ProRes profiles

Higher bitrate means higher quality and higher file size.

Audio-Only Export

You can pass mp3, wav or aac as a codec. If you do it, an audio file will be output in the corresponding format. Quality settings will be ignored.

Audio Codec

available from v3.3.42

Using the --audio-codec flag, you can set the format of the audio that is embedded in the video. Not all codec and audio codec combinations are supported and certain combinations require a certain file extension and container format.

The container format will be automatically derived based on the file extension.

Video codec	Default	Supported audio codecs	Possible file extensions
h264	aac	aac, pcm-16, mp3	.mp4 (default), .mkv, .mov
h265	aac	aac, pcm-16	.mp4 (default), .mkv, .hevc
prores	pcm-16	aac, pcm-16	.mov (default), .mkv, .mxf
vp8	opus	opus, pcm-16	.webm (default), .mkv
vp9	opus	opus, pcm-16	.webm (default), .mkv
wav	pcm-16	pcm-16	.wav (default)

Table 4: Audio codec compatibility

GIFs don't support audio.

Note: In versions before v4.0.0 the default audio codec for ProRes was aac. Now it's pcm-16.

Server-Side Rendering

Remotion's rendering engine is built with Node.JS, which makes it easy to render a video in the cloud.

Render a Video on AWS Lambda

The easiest and fastest way to render videos in the cloud is to use [@remotion/lambda](#).

Render a Video Using Node.js APIs

We provide a set of APIs to render videos using Node.js and Bun.

See an example or the API reference for more information.

Render Using GitHub Actions

You can render a video on GitHub actions. The following workflow assumes a composition ID of MyComp

```
name: Render video
on: workflow_dispatch:
jobs:
  render:
    name: Render video
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@main
      - uses: actions/setup-node@main
      - run: npm i
      - run: npx remotion render MyComp out/video.mp4
      - uses: actions/upload-artifact@v4
        with:
          name: out.mp4
          path: out/video.mp4
```

With Input Props

If you have props, you can ask for them using the GitHub Actions input fields.

Here we assume a shape of {titleText: string; titleColor: string}.

```
name: Render video
on:
  workflow_dispatch:
inputs:
  titleText:
    description: 'Which text should it say?'
    required: true
    default: 'Welcome to Remotion'
  titleColor:
    description: 'Which color should it be in?'
    required: true
    default: 'black'
jobs:
  render:
    name: Render video
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@main
      - uses: actions/setup-node@main
      - run: npm i
```

```
- run: echo $WORKFLOW_INPUT > input-props.json
env:
  WORKFLOW_INPUT: ${{ toJson(github.event.inputs) }}
- run: npx remotion render MyComp out/video.mp4 --props="./input-props.json"
- uses: actions/upload-artifact@v4
  with:
    name: out.mp4
    path: out/video.mp4
```

1. Commit the template to a GitHub repository
2. On GitHub, click the Actions tab
3. Select the Render video workflow on the left
4. A Run workflow button should appear. Click it
5. Fill in the props of the root component and click Run workflow

Note that running the workflow may incur costs. However, the workflow will only run if you actively trigger it.

Render a Video Using Docker

See: Dockerizing a Remotion project

Render a Video Using GCP Cloud Run (Alpha)

Check out the experimental Cloud Run package.

Note: It is not actively being developed - our plan is to port the Lambda runtime to Cloud Run instead of maintaining a separate implementation.

API Reference

Available SSR APIs:

- `getCompositions()`
 - `selectComposition()`
 - `renderMedia()`
 - `renderFrames()`
 - `renderStill()`
 - `stitchFramesToVideo()`
 - `openBrowser()`
 - `ensureBrowser()`
 - `makeCancelSignal()`
 - `getVideoMetadata()`
 - `getSilentParts()`
 - `combineChunks()`
 - `ensureFfmpeg()`
 - `ensureFfprobe()`
 - `getCanExtractFramesFast()`
-

AWS Lambda (@remotion/lambda)

Integrate Remotion Lambda into your app

Render Remotion videos on AWS Lambda. This is the fastest and most scalable way to render Remotion videos.

You only pay while you are rendering, making it very cost-effective.

When Should I Use It?

- Your videos are less than 80 minutes long at Full HD (approximately until the 15min AWS Timeout limit is hit)
- You stay within the AWS Lambda Concurrency Limit or you are requesting an increase from AWS
- You are fine with using Amazon Web Services in one of the supported regions

If one of those constraints is a dealbreaker for you, resort to normal server-side rendering or consider using Cloud Run.

See also: Comparison of server-side rendering options

How It Works

1. A Lambda function and a S3 bucket is created on AWS
2. A Remotion project gets deployed to a S3 bucket as a website
3. The Lambda function gets invoked and opens the Remotion project
4. A lot of Lambda functions are created in parallel which each render a small part of the video
5. The initial Lambda function downloads the videos and stitches them together
6. The final video gets uploaded to S3 and is available for download

See in more detail: How Remotion Lambda works

Architecture

Lambda function: Requires a layer with Chromium, currently hosted by Remotion. Only one Lambda function is required, but it can execute different actions.

S3 bucket: Stores the projects, the renders, and render metadata.

CLI: Allows to control the overall architecture from the command line. Is installed by adding @remotion/lambda to a project.

Node.JS API: Has the same features as the CLI but is easier to use programmatically.

Region Selection

The following regions are available for Remotion Lambda:

eu-central-1, eu-central-2, eu-west-1, eu-west-2, eu-west-3, eu-south-1, eu-north-1, us-east-1, us-east-2, us-west-1, us-west-2, af-south-1, ap-south-1, ap-east-1, ap-southeast-1, ap-southeast-2, ap-northeast-1, ap-northeast-2, ap-northeast-3, ap-southeast-4, ap-southeast-5, ca-central-1, me-south-1, sa-east-1

Limitations

- You only have up to 10GB of storage available in a Lambda function. This must be sufficient for both the separated chunks and the concatenated output, therefore the output file can only be about 5GB maximum, limiting the maximum video length to around 2 hours of Full HD video
- Lambda has a global limit of 1000 concurrent lambdas per region by default, although it can be increased

Cost

Most of our users render multiple minutes of video for just a few pennies. The exact cost is dependent on the region, assigned memory, type of video, parallelization and other parameters. For each render, we estimate a cost and display it to you. You might also need a Remotion license (see below).

AWS Permissions

Remotion Lambda requires you to create an AWS account and create a user with some permissions attached to it. We require only the minimal amount of permissions required for operating Remotion Lambda.

CLI

You can control Remotion Lambda using the `npx remotion lambda` command.

Node.JS API

Everything you can do using the CLI, you can also control using Node.JS APIs. See the reference here.

License

The standard Remotion license applies: <https://github.com/remotion-dev/remotion/blob/main/LICENSE.md>

Companies needing a license and using cloud rendering must set it up with Cloud Rendering Units. Please visit: <https://remotion.pro/license>

Uninstalling

We make it easy to remove all Remotion resources from your AWS account without leaving any traces or causing further costs.

Remotion Player

A component which can be rendered in a regular React App (for example: Next.JS, Vite.js, Create React App) to display a Remotion video.

MyApp.tsx

```
import { Player } from '@remotion/player';
import { MyVideo } from './remotion/MyVideo';
```

```
export const App: React.FC = () => {
  return (
    );
  };
}
```

Key API Properties

component or lazyComponent:

Pass a React component in directly **or** pass a function that returns a dynamic import. Passing neither or both of the props is an error.

If you use lazyComponent, wrap it in a useCallback() to avoid constant rendering.

The Player does not use <Composition>'s. Pass your component directly and do not wrap it in a <Composition> component.

durationInFrames:

The duration of the video in frames. Must be an integer and greater than 0.

fps:

The frame rate of the video. Must be a number.

compositionWidth:

The width you would like the video to have when rendered as an MP4. Use style={{width: <width>}} to define a width to be assumed in the browser.

compositionHeight:

The height you would like the video to have when rendered as an MP4. Use style={{height: <height>}} to define a height to be assumed in the browser.

inputProps:

Pass props to the component that you have specified using the component prop. The Typescript definition takes the shape of the props that you have given to your component.

loop?:

Whether the video should restart when it ends. Default false.

autoPlay?:

Whether the video should start immediately after loaded. Default false.

controls?:

Whether the video should display a seek bar and a play/pause button. Default false.

showVolumeControls?:

Whether the video should display a volume slider and a mute button. Only has an effect if controls is also set to true. Default true.

allowFullscreen?:

Whether the video can go fullscreen. By default true.

clickToPlay?:

A boolean property defining whether you can play, pause or resume the video with a single click into the player. Default true if controls are true, otherwise false.

playbackRate?:

A number between -4 and 4 (excluding 0) for the speed that the Player will run the media.

A playbackRate of 2 means the video plays twice as fast. A playbackRate of 0.5 means the video plays twice as slow. A playbackRate of -1 means the video plays in reverse.

Default 1.

Player Ref

You may attach a ref to the player and control it in an imperative manner.

```
import { Player, PlayerRef } from '@remotion/player';
import { useEffect, useRef } from 'react';
import { MyComposition } from './MyComposition';

const MyComp: React.FC = () => {
  const playerRef = useRef<PlayerRef>(null);

  useEffect(() => {
    if (playerRef.current) {
      console.log(playerRef.current.getCurrentFrame());
    }
  }, []);

  return (
    );
};
```

Available Methods on PlayerRef:

- `pause()` - Pause the video
- `play()` - Play the video
- `toggle()` - Pauses if playing, plays if paused
- `getCurrentFrame()` - Gets the current position as frame number
- `isPlaying()` - Returns boolean indicating if video is playing
- `getContainerNode()` - Gets the container HTMLDivElement
- `mute()` - Mutes the video
- `unmute()` - Unmutes the video
- `getVolume()` - Gets the volume (0 to 1)
- `setVolume(volume)` - Set the volume (0 to 1)
- `isMuted()` - Returns boolean if video is muted
- `seekTo(frame)` - Move to a specific frame
- `isFullscreen()` - Returns boolean if in fullscreen
- `requestFullscreen()` - Requests fullscreen mode
- `exitFullscreen()` - Exit fullscreen mode
- `getScale()` - Returns how much content is scaled down
- `addEventListener()` - Start listening to an event
- `removeEventListener()` - Stop listening to an event

Player Events

Using a player ref, you can bind event listeners to get notified of certain events:

Available Events:

- play - Fires when video starts playing
- pause - Fires when video has paused or ended
- seeked - Fired when time position is changed by user
- ended - Fires when video has ended and looping is disabled
- timeupdate - Fires periodic time updates when playing
- frameupdate - Fires whenever current time has changed
- ratechange - Fires when playbackRate has changed
- scalechange - Fires when scale has changed
- volumechange - Fires when volume has changed
- fullscreenchange - Fires when entering/exiting fullscreen
- mutechange - Fires when muted/unmuted
- error - Fires when error or exception happens
- waiting - Fires when entered buffering state
- resume - Fires when exited buffering state

Handling Errors

Since videos are written in React, they are prone to crashing.

When a video throws an exception, you may handle the error using the error event.

The video will unmount and show an error UI, but the host application (The React app which is embedding the player) will not crash.

You can customize the error message that is shown if a video crashes:

```
const MyApp: React.FC = () => {
  const errorFallback: ErrorFallback = useCallback(({ error }) => {
    return (
      <AbsoluteFill
        style={{
          backgroundColor: 'yellow',
          justifyContent: 'center',
          alignItems: 'center',
        }}
      >
        Sorry about this! An error occurred: {error.message}
      </AbsoluteFill>
    );
  }, []);
}

return (
);
```

AI Integration - Claude Code

Creating videos just from prompting

You can create videos just from prompting using Claude Code.

This is an easy way to get started with Remotion!

Prerequisites

You first need to install Claude Code and Node.js.

Claude Code requires a paid subscription.

Start a New Project

Create a new project using the following command:

```
npx create-video@latest
```

This will create a new project - we recommend the following settings:

- Select the Blank template
- Say yes to use TailwindCSS
- Say yes to install Skills

Start the Preview

First, go into the directory that was created.

If you named your project my-video, you would run the following command:

```
cd my-video
```

Install dependencies:

```
npm install
```

Start the project using the following command:

```
npm run dev
```

Start Claude

Open a separate terminal window and start Claude:

```
cd my-video  
claude
```

You can now prompt a video! See our video above for a few ideas on how to start.

Additional Documentation Sections

This document contains the core documentation sections from Remotion. The complete documentation includes 150+ pages covering:

Advanced Topics:

- Captions (importing, transcribing, displaying, exporting)
- Media Parser (parsing media files, extracting metadata)
- WebCodecs (video conversion and manipulation)
- Cloud Run (Google Cloud deployment)
- Client-side rendering
- Building custom apps
- Tooling (TailwindCSS, Webpack, TypeScript, Testing)
- Troubleshooting guides
- Migration guides
- Performance optimization
- Security considerations

For the most up-to-date and complete documentation, visit:

<https://www.remotion.dev/docs/>

References

All documentation content sourced from <https://www.remotion.dev/docs/> (accessed January 30, 2026)

Primary Sections Referenced:

1. Getting Started - <https://www.remotion.dev/docs/>
2. The Fundamentals - <https://www.remotion.dev/docs/the-fundamentals>
3. Animating Properties - <https://www.remotion.dev/docs/animating-properties>
4. Reusability - <https://www.remotion.dev/docs/reusability>
5. Preview - <https://www.remotion.dev/docs/preview>
6. Transforms - <https://www.remotion.dev/docs/transforms>
7. Assets - <https://www.remotion.dev/docs/assets>
8. Layers - <https://www.remotion.dev/docs/layers>
9. Transitions - <https://www.remotion.dev/docs/transitioning>
10. Audio - <https://www.remotion.dev/docs/using-audio>
11. Rendering - <https://www.remotion.dev/docs/render>
12. Encoding - <https://www.remotion.dev/docs/encoding>
13. Server-Side Rendering - <https://www.remotion.dev/docs/ssr>
14. AWS Lambda - <https://www.remotion.dev/docs/lambda>
15. Remotion Player - <https://www.remotion.dev/docs/player/player>
16. AI Integration - <https://www.remotion.dev/docs/ai/clause-code>