Замена фона в видеозвонке

Николай Васильчук

VideoTech

2023

Николай Васильчук

@vasilchuk

- 13 лет с видео
- 9 лет в VK
- 5 лет со звонками

О чём этот доклад?

- Как работают звонки и что такое WebRTC
- Как разобрать видео на кадры и собрать обратно
- Обработка кадров
- Canvas, нейронки, WebGL

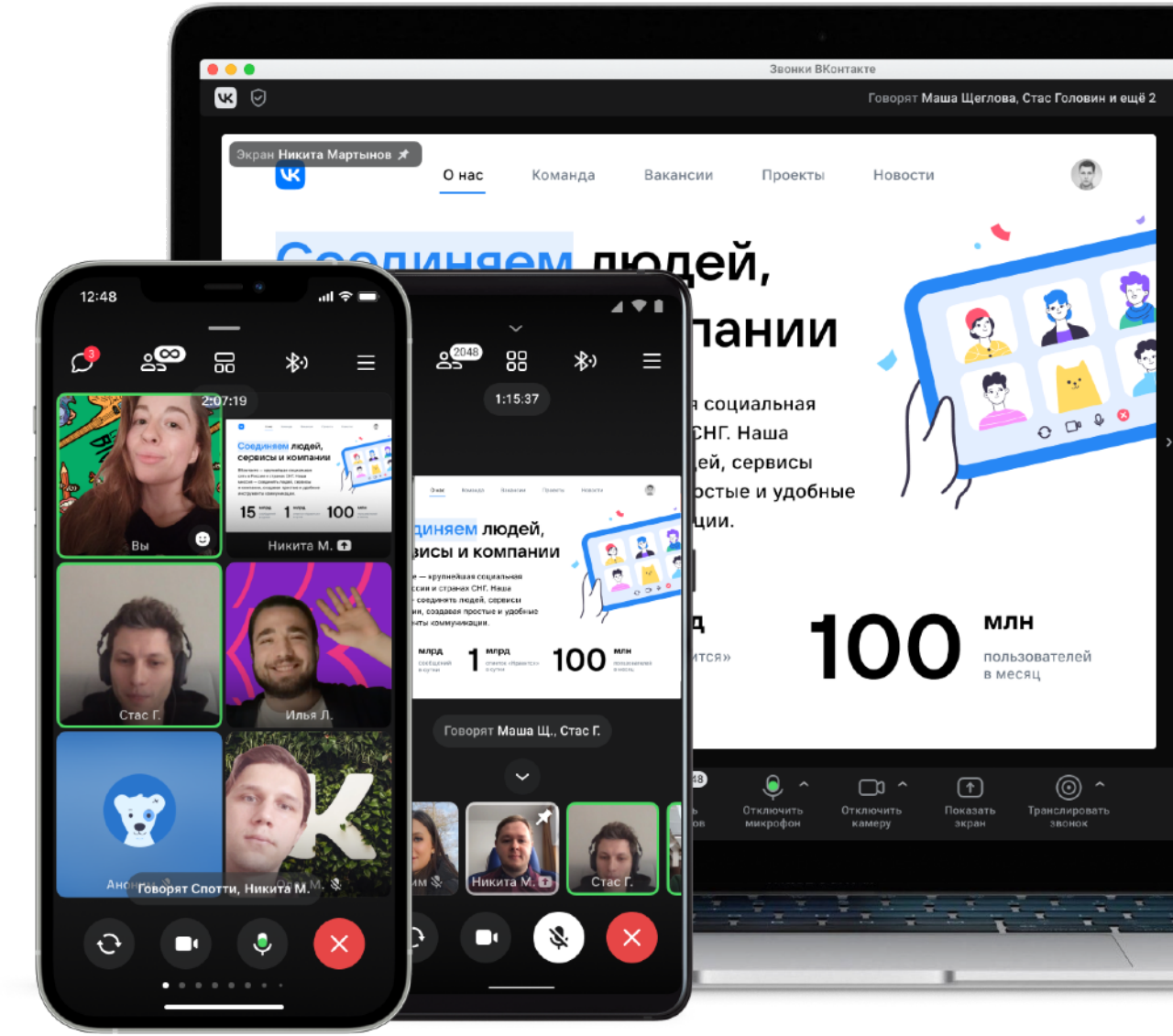
VK Звонки

Бесплатные звонки без ограничений по времени
и количеству участников.

Для работы и учёбы
Демонстрация экрана в 4K, трансляция, планирование и запись.

Управление звонками
Зал ожидания, управление микрофонами, функция «Поднять руку» и другие возможности модерации.

Технологичность
Интеллектуальное шумоподавление, собственная AR-технология замены фона.



20 млн

пользователей общаются в VK Звонках ежемесячно

Источник: ВКонтакте, апрель 2022 г.

2012	2018	05.2020	09.2020	02.2021
Start: web	mobile	**8** group calls	**128**	

07.2021	08.2021	10.2021	11.2021	04.2022
4K	**VK Звонки** desktop **2048**	**SDK**	∞ участников	**VK Звонки** mobile app

...2023

- Замена фона
- API для управления звонками
- Поддержка переговорных
- Совместный просмотр
- vmoji
- Сессионные залы
- Расшифровка звонков и автосубтитры

VK Звонки

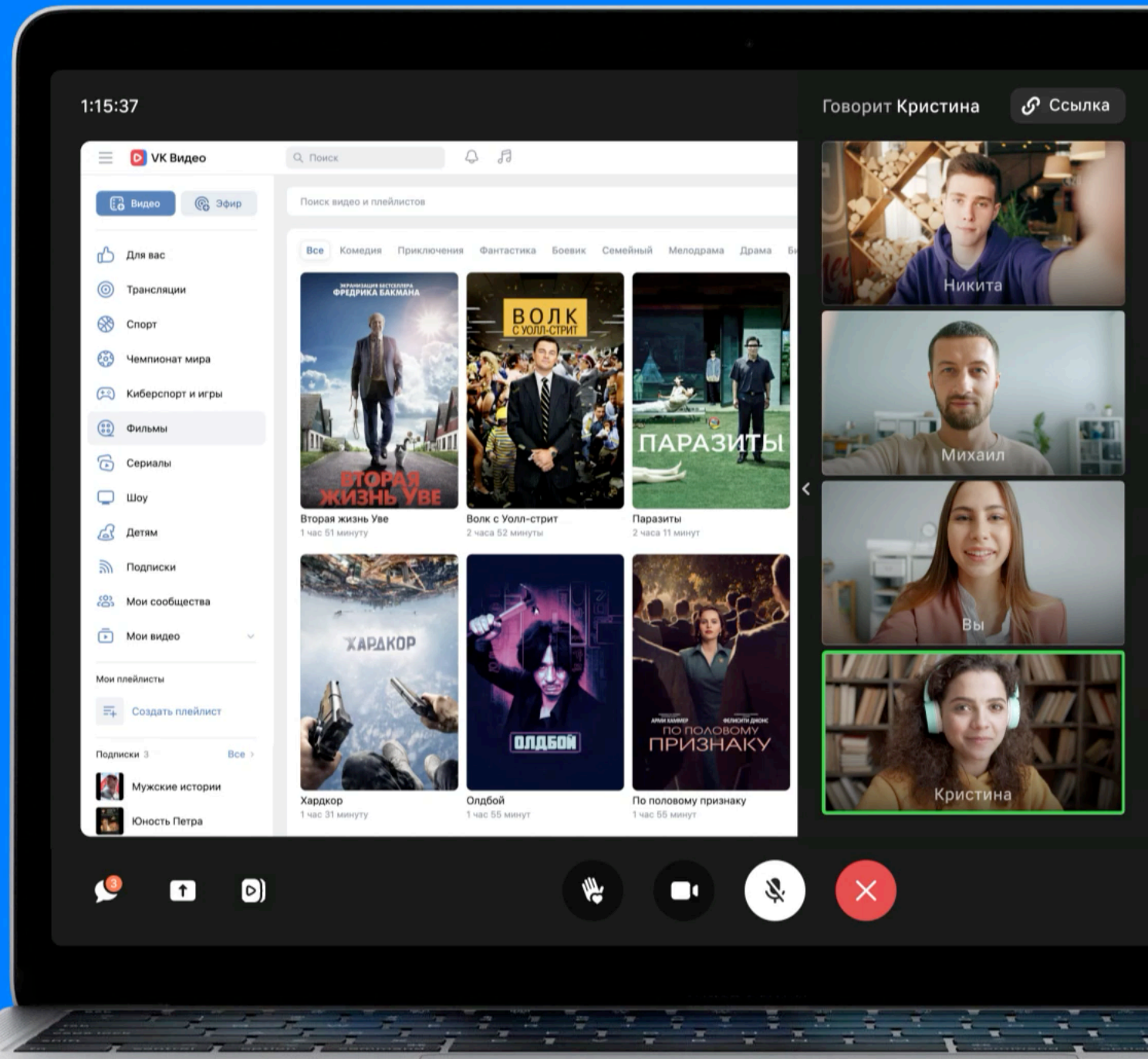
новое

vmoji
вместо вас

VK Звонки

новое

совместный
просмотр
и реакции



Как устроены звонки

WebRTC — Web Real Time Communication

10 Years after Inception, WebRTC Becomes an Official Web Standard

APR 07, 2021 • 3 MIN READ

Web Real-Time Communications (WebRTC) recently became a World Wide Web Consortium (W3C) recommendation and Internet Engineering Task Force (IETF) standard. This is a major milestone on a long journey for WebRTC that started in 2011 with Google open-sourcing key communication technologies and Ericsson implementing the ConnectionPeer API. The new standard will continue to evolve as the WebRTC Working Group strives to integrate new use cases — live processing of audio and video feeds, Internet of Things use cases, and more.

https://www.infoq.com/news/2021/04/webrtc-official-web-standard/

WebRTC Peer-to-peer connections

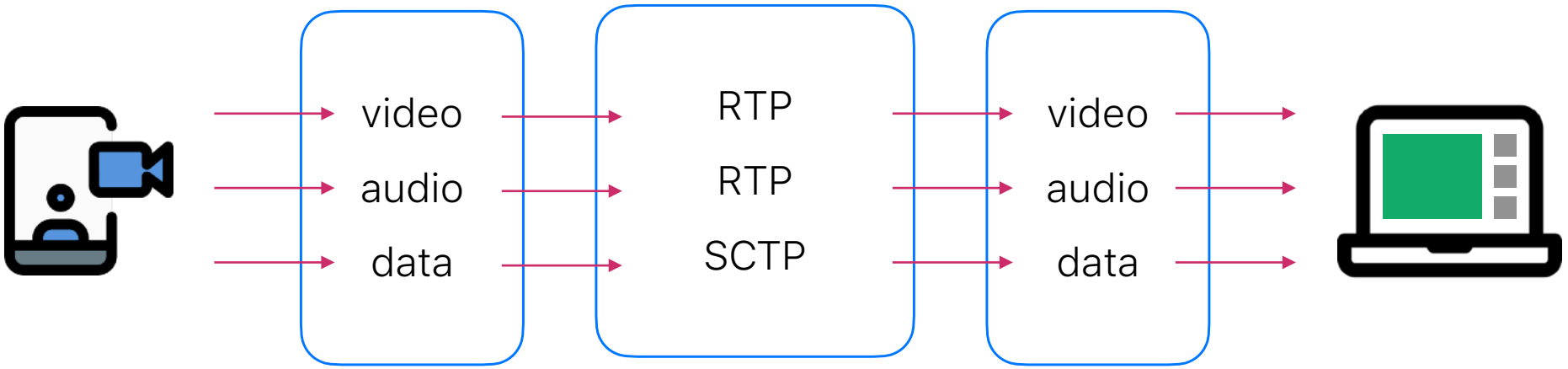
- WD

Method of allowing two users to communicate directly, browser to browser using the RTCPeerConnection API.

Usage	% of	all users	?
Global	94.49% + 0.02% =	94.51%	
unprefixed:	93.73% + 0.02% =	93.75%	

Current aligned | Usage relative | Date relative | Filtered | All

Chrome	Edge *	Safari	Firefox	Opera	Chrome for Android	Safari on iOS *	Opera Mini *	UC Browser for Android
109						15.7		
114	114	15.6	115			16.1		
115	115	16.5	116	100		16.3		
116	116	16.6	117	102	116	16.5	all	15.5
117		17	118			16.6		
118		TP	119			17		
119			120					



Real-time Transport Protocol

Stream Control Transmission Protocol

```
const stream = await navigator.mediaDevices.getUserMedia({
    audio: true,
    video: true,
});
const pc = new RTCPeerConnection();
stream.getTracks().forEach((track) => pc.addTrack(track, stream));
const offer = await pc.createOffer({
    offerToReceiveAudio: true,
    offerToReceiveVideo: true,
});
await pc.setLocalDescription(offer);

const socket = new WebSocket(endpoint);
socket.send({
    type: 'offer',
    data: offer,
});
```

```
const stream = await navigator.mediaDevices.getUserMedia({
    audio: true,
    video: true,
});
const pc = new RTCPeerConnection();
stream.getTracks().forEach((track) => pc.addTrack(track, stream));
const offer = await pc.createOffer({
    offerToReceiveAudio: true,
    offerToReceiveVideo: true,
});
await pc.setLocalDescription(offer);

const socket = new WebSocket(endpoint);
socket.send({
    type: 'offer',
    data: offer,
});
```

```
const stream = await navigator.mediaDevices.getUserMedia({
    audio: true,
    video: true,
});
const pc = new RTCPeerConnection();
stream.getTracks().forEach((track) => pc.addTrack(track, stream));
const offer = await pc.createOffer({
    offerToReceiveAudio: true,
    offerToReceiveVideo: true,
});
await pc.setLocalDescription(offer);

const socket = new WebSocket(endpoint);
socket.send({
    type: 'offer',
    data: offer,
});
```

```
const stream = await navigator.mediaDevices.getUserMedia({
    audio: true,
    video: true,
});
const pc = new RTCPeerConnection();
stream.getTracks().forEach((track) => pc.addTrack(track, stream));
const offer = await pc.createOffer({
    offerToReceiveAudio: true,
    offerToReceiveVideo: true,
});
await pc.setLocalDescription(offer);

const socket = new WebSocket(endpoint);
socket.send({
    type: 'offer',
    data: offer,
});
```

```
const stream = await navigator.mediaDevices.getUserMedia({
    audio: true,
    video: true,
});
const pc = new RTCPeerConnection();
stream.getTracks().forEach((track) => pc.addTrack(track, stream));
const offer = await pc.createOffer({
    offerToReceiveAudio: true,
    offerToReceiveVideo: true,
});
await pc.setLocalDescription(offer);

const socket = new WebSocket(endpoint);
socket.send({
    type: 'offer',
    data: offer,
});
```

```
const stream = await navigator.mediaDevices.getUserMedia({
    audio: true,
    video: true,
});
const pc = new RTCPeerConnection();
stream.getTracks().forEach((track) => pc.addTrack(track, stream));
const offer = await pc.createOffer({
    offerToReceiveAudio: true,
    offerToReceiveVideo: true,
});
await pc.setLocalDescription(offer);

const socket = new WebSocket(endpoint);
socket.send({
    type: 'offer',
    data: offer,
});
```

```
const stream = await navigator.mediaDevices.getUserMedia({
    audio: true,
    video: true,
});
const pc = new RTCPeerConnection();
stream.getTracks().forEach((track) => pc.addTrack(track, stream));
await pc.setRemoteDescription(offer);
const answer = await pc.createAnswer();
await pc.setLocalDescription(answer);

socket.send({
    type: 'answer',
    data: answer,
});
```

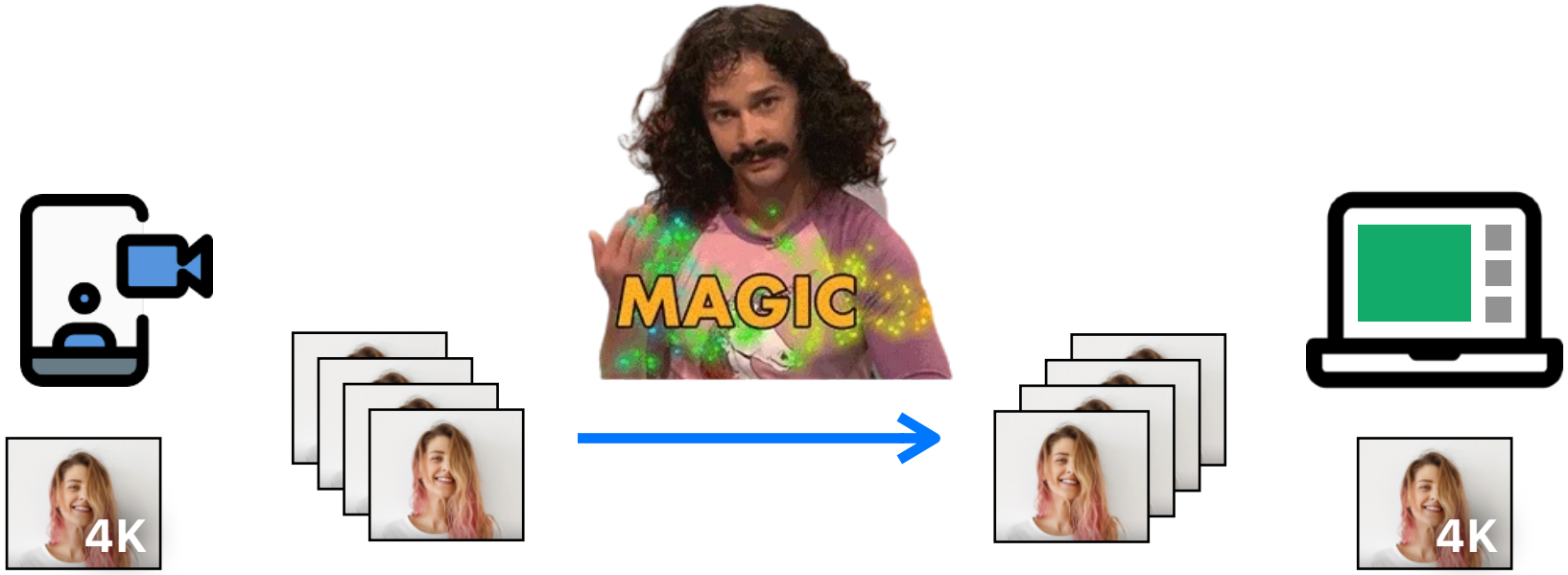
```
const stream = await navigator.mediaDevices.getUserMedia({
    audio: true,
    video: true,
});
const pc = new RTCPeerConnection();
stream.getTracks().forEach((track) => pc.addTrack(track, stream));
await pc.setRemoteDescription(offer);
const answer = await pc.createAnswer();
await pc.setLocalDescription(answer);

socket.send({
    type: 'answer',
    data: answer,
});
```

```
const stream = await navigator.mediaDevices.getUserMedia({
    audio: true,
    video: true,
});
const pc = new RTCPeerConnection();
stream.getTracks().forEach((track) => pc.addTrack(track, stream));
await pc.setRemoteDescription(offer);
const answer = await pc.createAnswer();
await pc.setLocalDescription(answer);

socket.send({
    type: 'answer',
    data: answer,
});
```

```
const stream = await navigator.mediaDevices.getUserMedia({
    audio: true,
    video: true,
});
const pc = new RTCPeerConnection();
stream.getTracks().forEach((track) => pc.addTrack(track, stream));
await pc.setRemoteDescription(offer);
const answer = await pc.createAnswer();
await pc.setLocalDescription(answer);

socket.send({
    type: 'answer',
    data: answer,
});
```

```
const stream = await navigator.mediaDevices.getUserMedia({
    audio: true,
    video: true,
});
const pc = new RTCPeerConnection();
stream.getTracks().forEach((track) => pc.addTrack(track, stream));
await pc.setRemoteDescription(offer);
const answer = await pc.createAnswer();
await pc.setLocalDescription(answer);

socket.send({
    type: 'answer',
    data: answer,
});
```

Insertable Streams


MAGIC
4K
4K

Insertable Streams API

https://developer.mozilla.org/en-US/docs/Web/API/Insertable_Streams_for_MediaStreamTrack_API

https://www.w3.org/TR/mediacapture-transform/

	Chrome	Edge	Firefox	Opera	Safari	Chrome Android	Firefox for Android	Opera Android	Safari on iOS	Samsung Internet	WebView Android
`MediaStreamTrackGenerator`	✓ 94	✓ 94	✗ No	✓ 80	✗ No	✓ 94	✗ No	✓ 66	✗ No	✓ 17.0	✓ 94
`MediaStreamTrackGenerator()` constructor	✓ 94	✓ 94	✗ No	✓ 80	✗ No	✓ 94	✗ No	✓ 66	✗ No	✓ 17.0	✓ 94
`writable`	✓ 94	✓ 94	✗ No	✓ 80	✗ No	✓ 94	✗ No	✓ 66	✗ No	✓ 17.0	✓ 94

MediaStreamTrackProcessor

Insertable Streams

MediaStreamTrackGenerator

```
const stream = await navigator.getUserMedia({ video: true });
const [videoTrack] = stream.getVideoTracks();

const trackProcessor = new MediaStreamTrackProcessor(videoTrack);
const frameReader = trackProcessor.readable.getReader();

const {done, value: videoFrame} = await frameReader.read();

const newFrame = await applyEffect(videoFrame, effect);
```

```
const stream = await navigator.getUserMedia({ video: true });
const [videoTrack] = stream.getVideoTracks();

const trackProcessor = new MediaStreamTrackProcessor(videoTrack);
const frameReader = trackProcessor.readable.getReader();

const {done, value: videoFrame} = await frameReader.read();

const newFrame = await applyEffect(videoFrame, effect);
```

```
const stream = await navigator.getUserMedia({ video: true });
const [videoTrack] = stream.getVideoTracks();

const trackProcessor = new MediaStreamTrackProcessor(videoTrack);
const frameReader = trackProcessor.readable.getReader();

const {done, value: videoFrame} = await frameReader.read();

const newFrame = await applyEffect(videoFrame, effect);
```

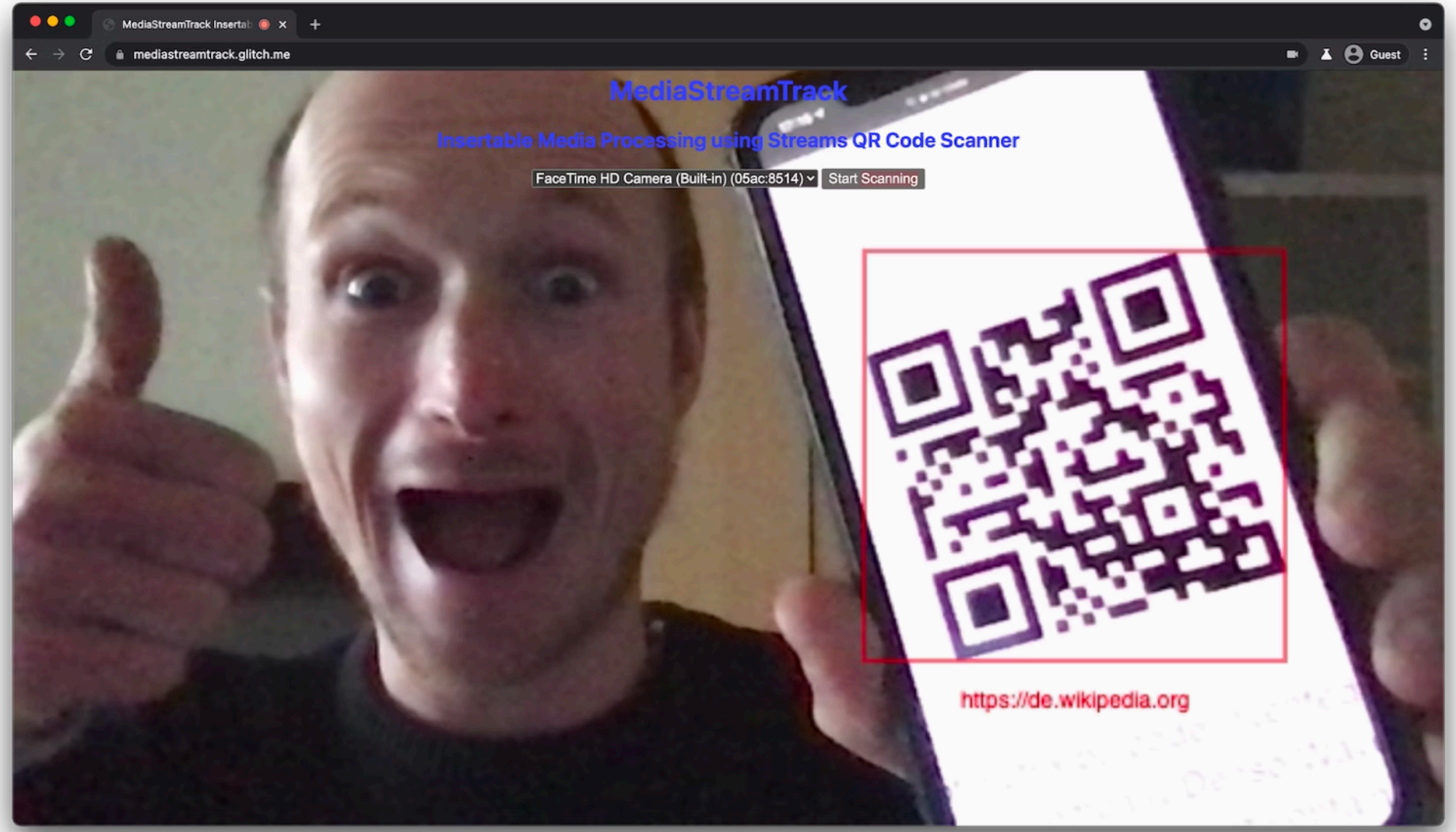
```
const stream = await navigator.getUserMedia({ video: true });
const [videoTrack] = stream.getVideoTracks();

const trackProcessor = new MediaStreamTrackProcessor(videoTrack);
const frameReader = trackProcessor.readable.getReader();

const {done, value: videoFrame} = await frameReader.read();

const newFrame = await applyEffect(videoFrame, effect);
```

```
const trackGenerator = new MediaStreamTrackGenerator({ kind: 'video' });
const frameWriter = trackGenerator.writable.getWriter();

await frameWriter.write(newFrame);

const video = document.createElement('video');
video.srcObject = trackGenerator;
video.play();
```

```
const trackGenerator = new MediaStreamTrackGenerator({ kind: 'video' });
const frameWriter = trackGenerator.writable.getWriter();

await frameWriter.write(newFrame);

const video = document.createElement('video');
video.srcObject = trackGenerator;
video.play();
```

```
const trackGenerator = new MediaStreamTrackGenerator({ kind: 'video' });
const frameWriter = trackGenerator.writable.getWriter();

await frameWriter.write(newFrame);

const video = document.createElement('video');
video.srcObject = trackGenerator;
video.play();
```

```
async function applyEffect(videoFrame) {
    const barcodes = await detectBarcodes(videoFrame);
    return highlightBarcodes(videoFrame, barcodes);
}
```


MediaStreamTrack Insertab
mediastreamtrack.glitch.me
Guest
MediaStreamTrack
Insertable Media Processing using Streams QR Code Scanner
FaceTime HD Camera (Built-in) (05ac:8514)
Start Scanning
https://de.wikipedia.org

Видеоэффекты


SONY
ON AIR

Filter

Overlay

Background

Blur

Filter + Overlay

Filter + Background + Overlay

VideoFrame → Canvas → VideoFrame

Filter Effect

canvasContext.filter

blur() brightness() contrast() drop-shadow()
grayscale() hue-rotate() invert() saturate() sepia()

https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D/filter

```
const canvas = new OffscreenCanvas(displayWidth, displayHeight);
const canvasCtx = canvas.getContext('2d', {willReadFrequently: true});

canvasCtx.filter = 'sepia(0.8)';
canvasCtx.drawImage(videoFrame, 0, 0);

const newFrame = new VideoFrame(canvas, {
    timestamp,
    displayWidth,
    displayHeight,
    duration,
});
```

```
const canvas = new OffscreenCanvas(displayWidth, displayHeight);
const canvasCtx = canvas.getContext('2d', {willReadFrequently: true});

canvasCtx.filter = 'sepia(0.8)';
canvasCtx.drawImage(videoFrame, 0, 0);

const newFrame = new VideoFrame(canvas, {
    timestamp,
    displayWidth,
    displayHeight,
    duration,
});
```

```
const canvas = new OffscreenCanvas(displayWidth, displayHeight);
const canvasCtx = canvas.getContext('2d', {willReadFrequently: true});

canvasCtx.filter = 'sepia(0.8)';
canvasCtx.drawImage(videoFrame, 0, 0);

const newFrame = new VideoFrame(canvas, {
    timestamp,
    displayWidth,
    displayHeight,
    duration,
});
```

Overlay Effect

canvasContext.drawImage()
x2

https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D/drawImage

```
const canvas = new OffscreenCanvas(width, height);
const canvasCtx = canvas.getContext('2d', {willReadFrequently: true});

const overlayImageSource = await loadImage(url);
const overlayImageData = resize(overlayImageSource, width, height);

canvasCtx.drawImage(videoFrame, 0, 0);
canvasCtx.drawImage(overlayImageData, 0, 0);

const newFrame = new VideoFrame(canvas, {
    timestamp,
    displayWidth,
    displayHeight,
    duration,
});
```

```
const canvas = new OffscreenCanvas(width, height);
const canvasCtx = canvas.getContext('2d', {willReadFrequently: true});

const overlayImageSource = await loadImage(url);
const overlayImageData = resize(overlayImageSource, width, height);

canvasCtx.drawImage(videoFrame, 0, 0);
canvasCtx.drawImage(overlayImageData, 0, 0);

const newFrame = new VideoFrame(canvas, {
    timestamp,
    displayWidth,
    displayHeight,
    duration,
});
```

```
const canvas = new OffscreenCanvas(width, height);
const canvasCtx = canvas.getContext('2d', {willReadFrequently: true});

const overlayImageSource = await loadImage(url);
const overlayImageData = resize(overlayImageSource, width, height);

canvasCtx.drawImage(videoFrame, 0, 0);
canvasCtx.drawImage(overlayImageData, 0, 0);

const newFrame = new VideoFrame(canvas, {
    timestamp,
    displayWidth,
    displayHeight,
    duration,
});
```

```
const canvas = new OffscreenCanvas(width, height);
const canvasCtx = canvas.getContext('2d', {willReadFrequently: true});

const overlayImageSource = await loadImage(url);
const overlayImageData = resize(overlayImageSource, width, height);

canvasCtx.drawImage(videoFrame, 0, 0);
canvasCtx.drawImage(overlayImageData, 0, 0);

const newFrame = new VideoFrame(canvas, {
    timestamp,
    displayWidth,
    displayHeight,
    duration,
});
```

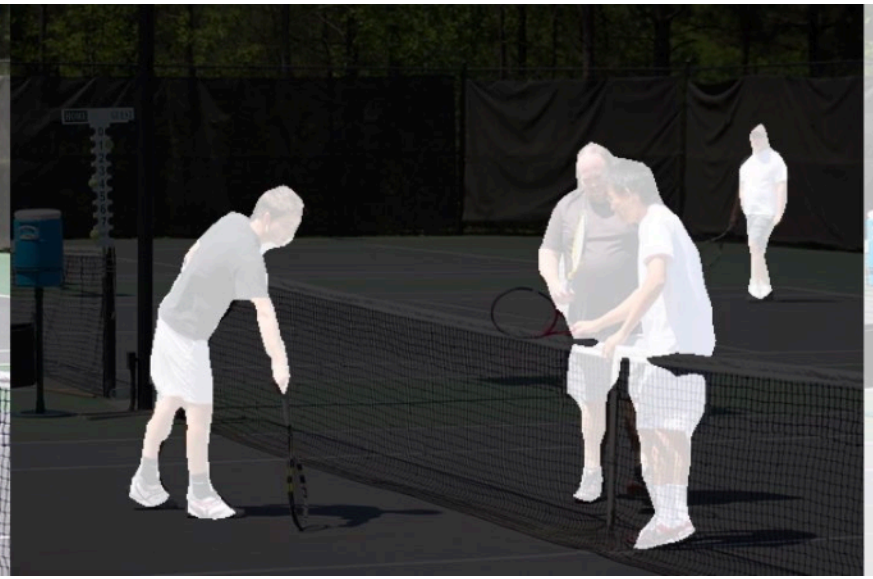
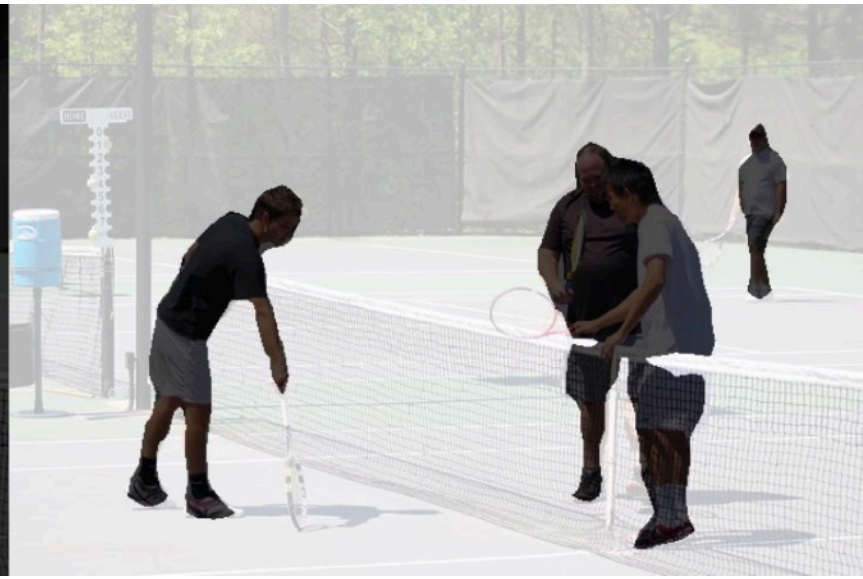
```
function resize(image, width, height) {
    const canvas = new OffscreenCanvas(width, height);
    const canvasCtx = canvas.getContext('2d');

    canvasCtx.drawImage(image, 0, 0, width, height);
    return canvasCtx.getImageData(0, 0, width, height);
}
```

ML

canvasContext.drawImage() x2

tfjs-models/body-segmentation

https://github.com/tensorflow/tfjs-models/tree/master/body-segmentation

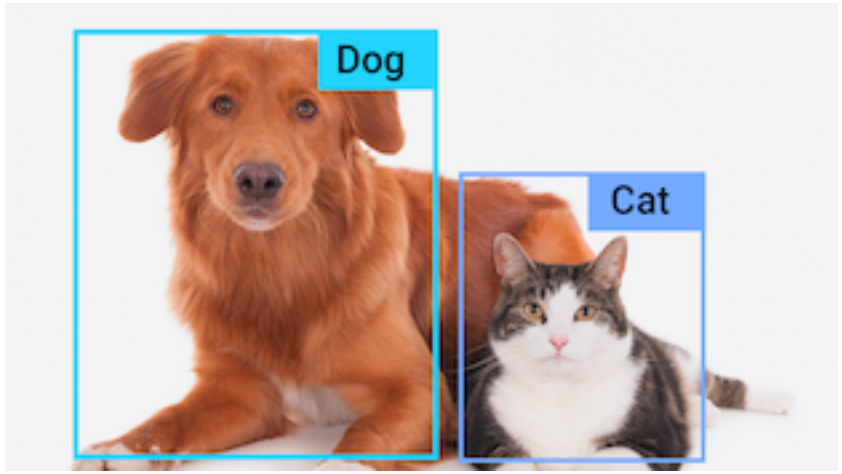
Body Segmentation

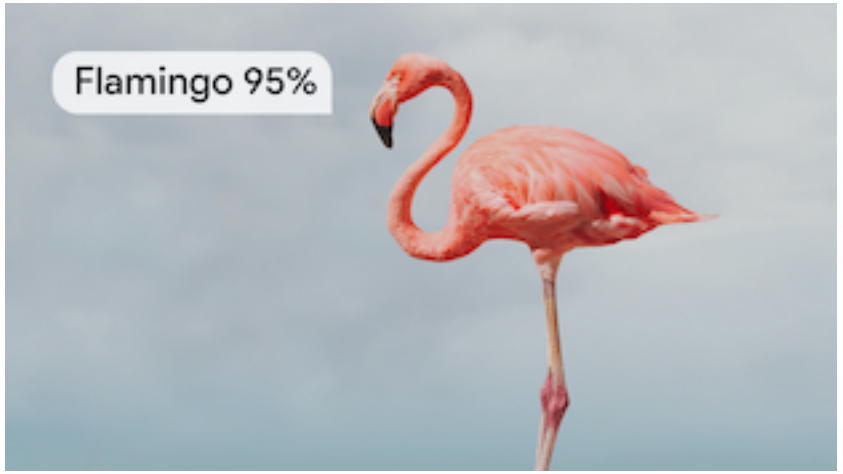
+ Работает
+ Много возможностей
+ Разные модели

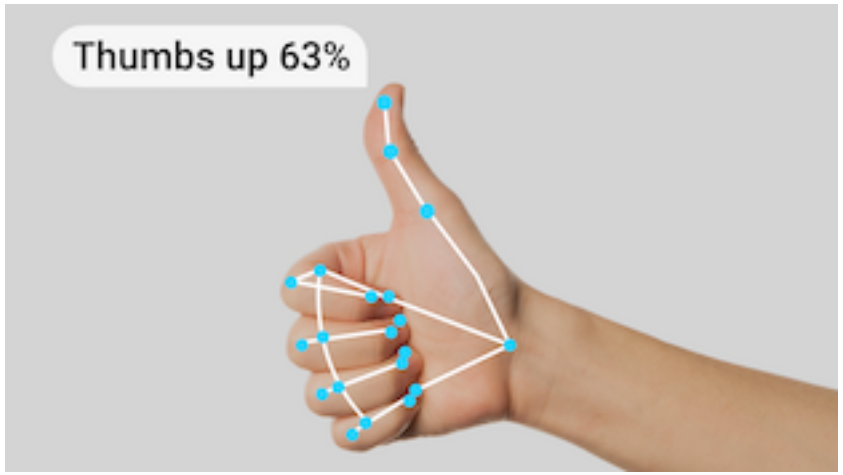
- Медленный
- Тяжелый

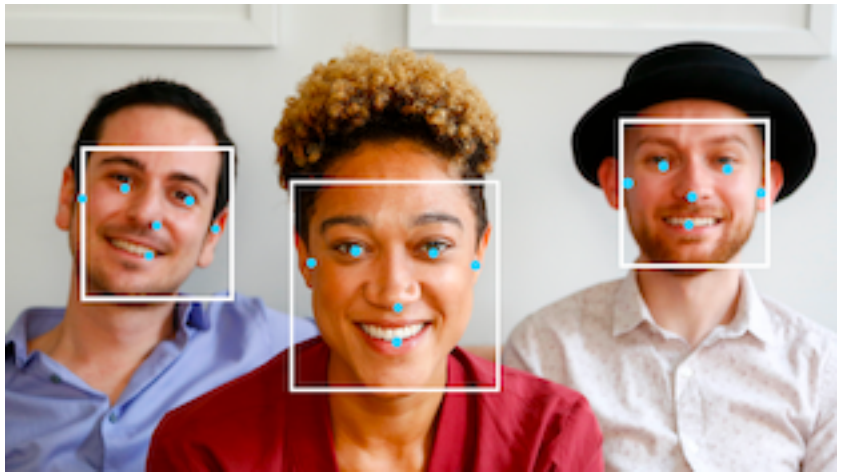
MediaPipe

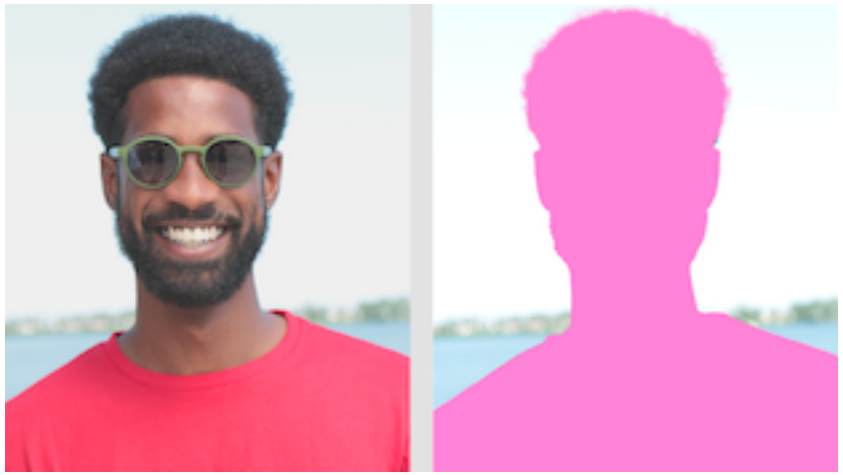
https://developers.google.com/mediapipe


Dog
Cat


Flamingo 95%


Thumbs up 63%



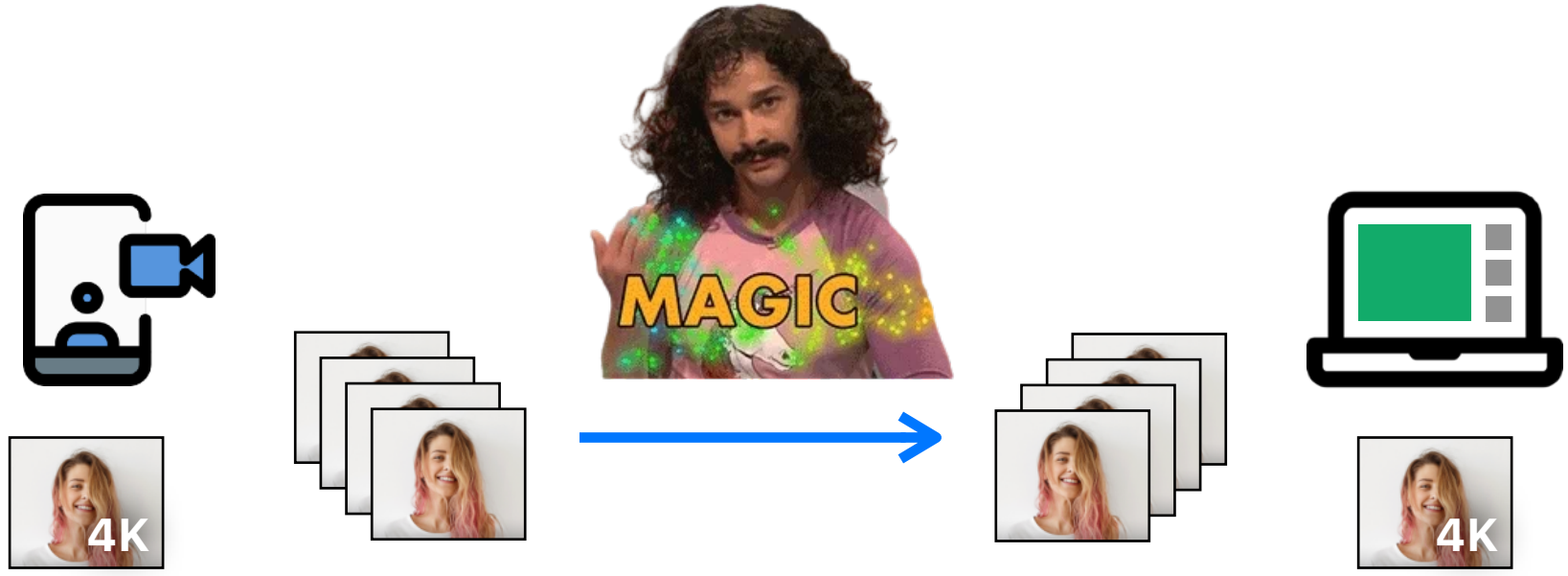



SONY
ON AIR

MediaPipe

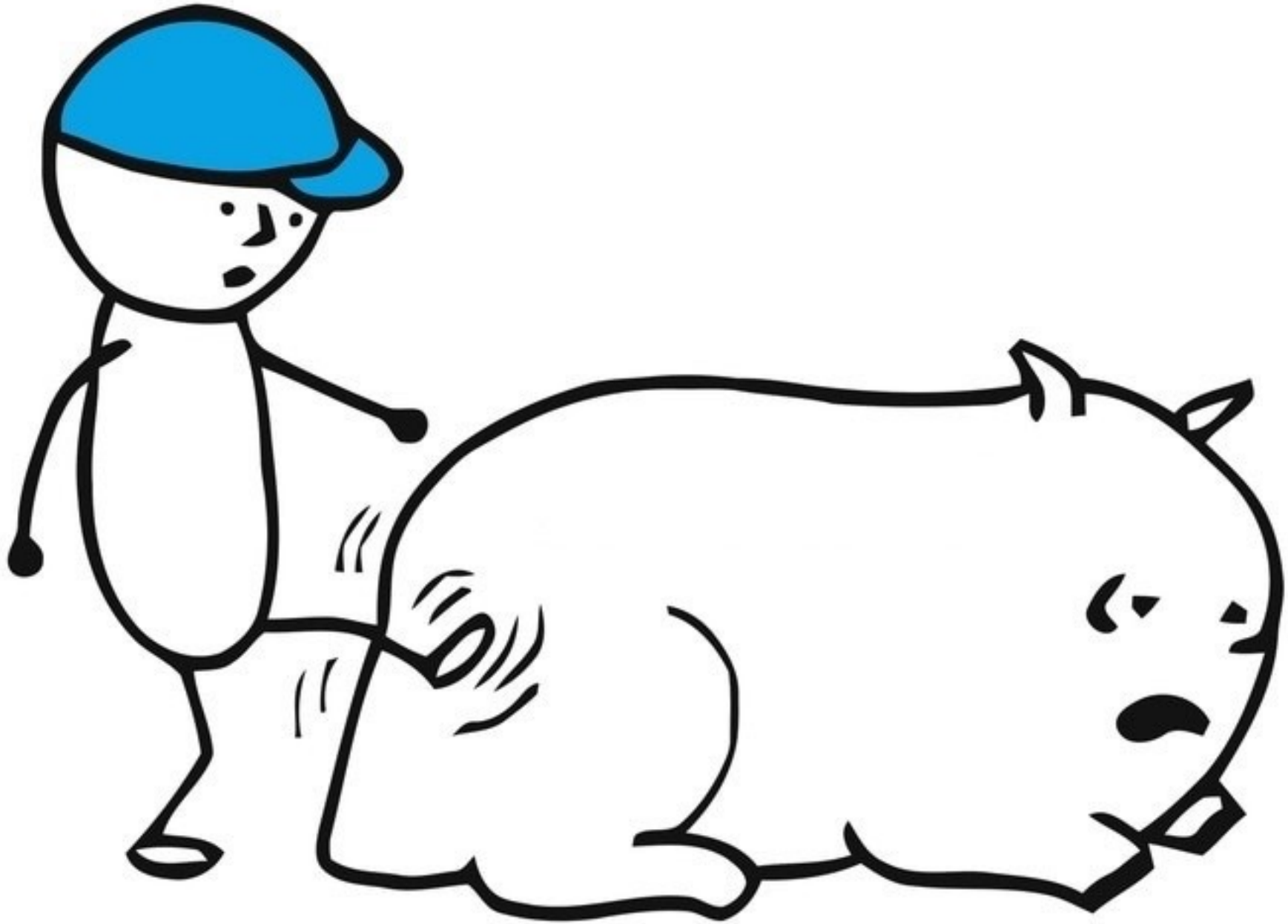
+ Очень удобный
+ Очень много возможностей
+ Документация
+ Быстрый

- Тяжелый



25 fps

40 ms



https://www.tensorflow.org/js

https://onnxruntime.ai/


ON AIR


ON AIR


MAGIC

Background Effect


ON AIR
SONY


MAGIC

```
const canvas = new OffscreenCanvas(width, height);
const canvasCtx = canvas.getContext('2d', {willReadFrequently: true});

canvasCtx.clearRect(width, height);
canvasCtx.drawImage(videoFrame, 0, 0);

canvasCtx.globalCompositeOperation = 'destination-in';
canvasCtx.drawImage(segmentationMask, 0, 0);

canvasCtx.globalCompositeOperation = 'destination-over';
canvasCtx.drawImage(background, 0, 0);
```

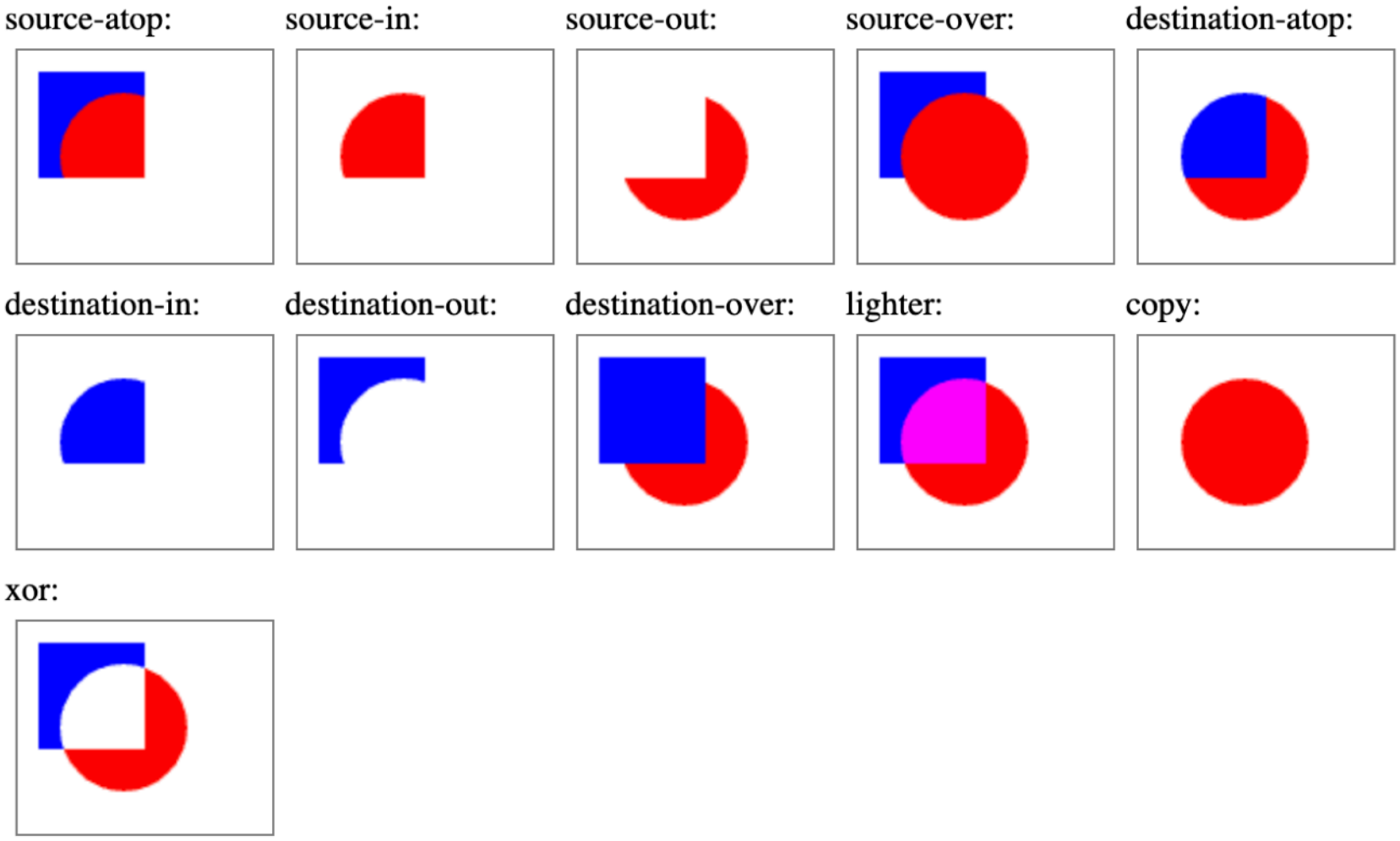
```
const canvas = new OffscreenCanvas(width, height);
const canvasCtx = canvas.getContext('2d', {willReadFrequently: true});

canvasCtx.clearRect(width, height);
canvasCtx.drawImage(videoFrame, 0, 0);

canvasCtx.globalCompositeOperation = 'destination-in';
canvasCtx.drawImage(segmentationMask, 0, 0);

canvasCtx.globalCompositeOperation = 'destination-over';
canvasCtx.drawImage(background, 0, 0);
```

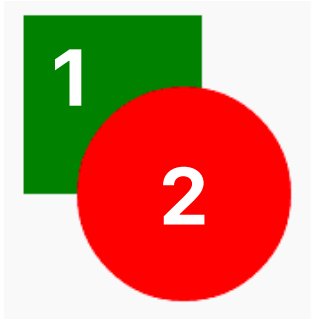
```
const canvas = new OffscreenCanvas(width, height);
const canvasCtx = canvas.getContext('2d', {willReadFrequently: true});

canvasCtx.clearRect(width, height);
canvasCtx.drawImage(videoFrame, 0, 0);

canvasCtx.globalCompositeOperation = 'destination-in';
canvasCtx.drawImage(segmentationMask, 0, 0);

canvasCtx.globalCompositeOperation = 'destination-over';
canvasCtx.drawImage(background, 0, 0);
```

```
const canvas = new OffscreenCanvas(width, height);
const canvasCtx = canvas.getContext('2d', {willReadFrequently: true});

canvasCtx.clearRect(width, height);
canvasCtx.drawImage(videoFrame, 0, 0);

canvasCtx.globalCompositeOperation = 'destination-in';
canvasCtx.drawImage(segmentationMask, 0, 0);

canvasCtx.globalCompositeOperation = 'destination-over';
canvasCtx.drawImage(background, 0, 0);
```

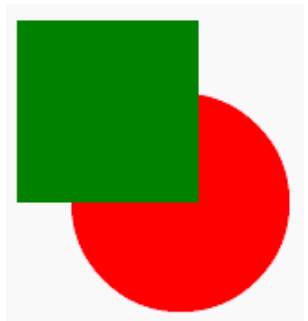
globalCompositeOperation

https://www.w3resource.com/html5-canvas/html5-canvas-compositing.php



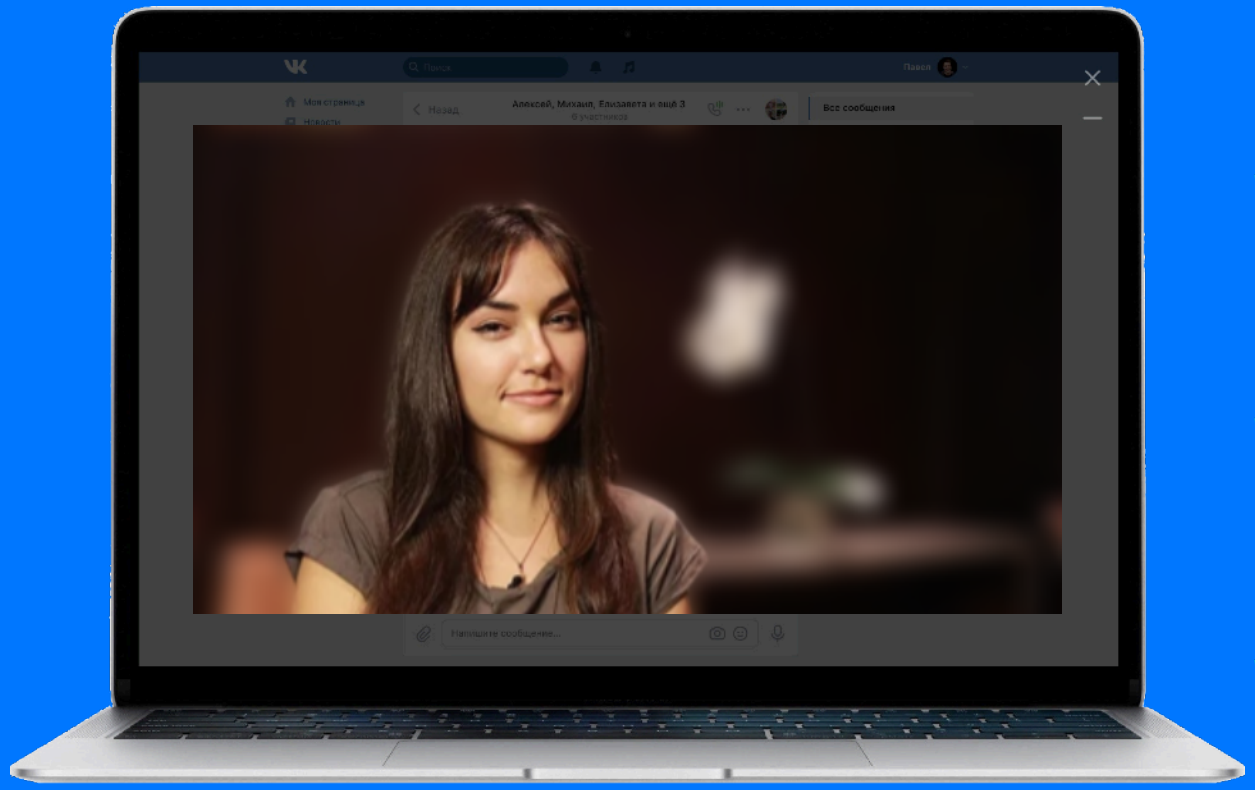


destination-in



destination-over

Blur Effect



```
const canvas = new OffscreenCanvas(width, height);
const canvasCtx = canvas.getContext('2d', {willReadFrequently: true});

canvasCtx.clearRect(width, height);
canvasCtx.drawImage(videoFrame, 0, 0);

canvasCtx.globalCompositeOperation = 'destination-in';
canvasCtx.drawImage(segmentationMask, 0, 0);

canvasCtx.globalCompositeOperation = 'destination-over';
canvasCtx.filter = 'blur(20px)';
canvasCtx.drawImage(videoFrame, 0, 0);
canvasCtx.filter = 'none';
```

```
const canvas = new OffscreenCanvas(width, height);
const canvasCtx = canvas.getContext('2d', {willReadFrequently: true});

canvasCtx.clearRect(width, height);
canvasCtx.drawImage(videoFrame, 0, 0);

canvasCtx.globalCompositeOperation = 'destination-in';
canvasCtx.drawImage(segmentationMask, 0, 0);

canvasCtx.globalCompositeOperation = 'destination-over';
canvasCtx.filter = 'blur(20px)';
canvasCtx.drawImage(videoFrame, 0, 0);
canvasCtx.filter = 'none';
```

```
const canvas = new OffscreenCanvas(width, height);
const canvasCtx = canvas.getContext('2d', {willReadFrequently: true});

canvasCtx.clearRect(width, height);
canvasCtx.drawImage(videoFrame, 0, 0);

canvasCtx.globalCompositeOperation = 'destination-in';
canvasCtx.drawImage(segmentationMask, 0, 0);

canvasCtx.globalCompositeOperation = 'destination-over';
canvasCtx.filter = 'blur(20px)';
canvasCtx.drawImage(videoFrame, 0, 0);
canvasCtx.filter = 'none';
```

Combined Effect


ON AIR
SONY


Background


Filter


Overlay

```
const canvas = new OffscreenCanvas(width, height);
const canvasCtx = canvas.getContext('2d', {willReadFrequently: true});

const effects = [background, filter, overlay];
let nextFrame = videoFrame;

for (const effect of effects) {
    effect.draw(nextFrame);
    nextFrame = effect.getCanvas();
}

canvasCtx.drawImage(nextFrame, 0, 0);
```

```
const canvas = new OffscreenCanvas(width, height);
const canvasCtx = canvas.getContext('2d', {willReadFrequently: true});

const effects = [background, filter, overlay];
let nextFrame = videoFrame;

for (const effect of effects) {
    effect.draw(nextFrame);
    nextFrame = effect.getCanvas();
}

canvasCtx.drawImage(nextFrame, 0, 0);
```

```
const canvas = new OffscreenCanvas(width, height);
const canvasCtx = canvas.getContext('2d', {willReadFrequently: true});

const effects = [background, filter, overlay];
let nextFrame = videoFrame;

for (const effect of effects) {
    effect.draw(nextFrame);
    nextFrame = effect.getCanvas();
}

canvasCtx.drawImage(nextFrame, 0, 0);
```

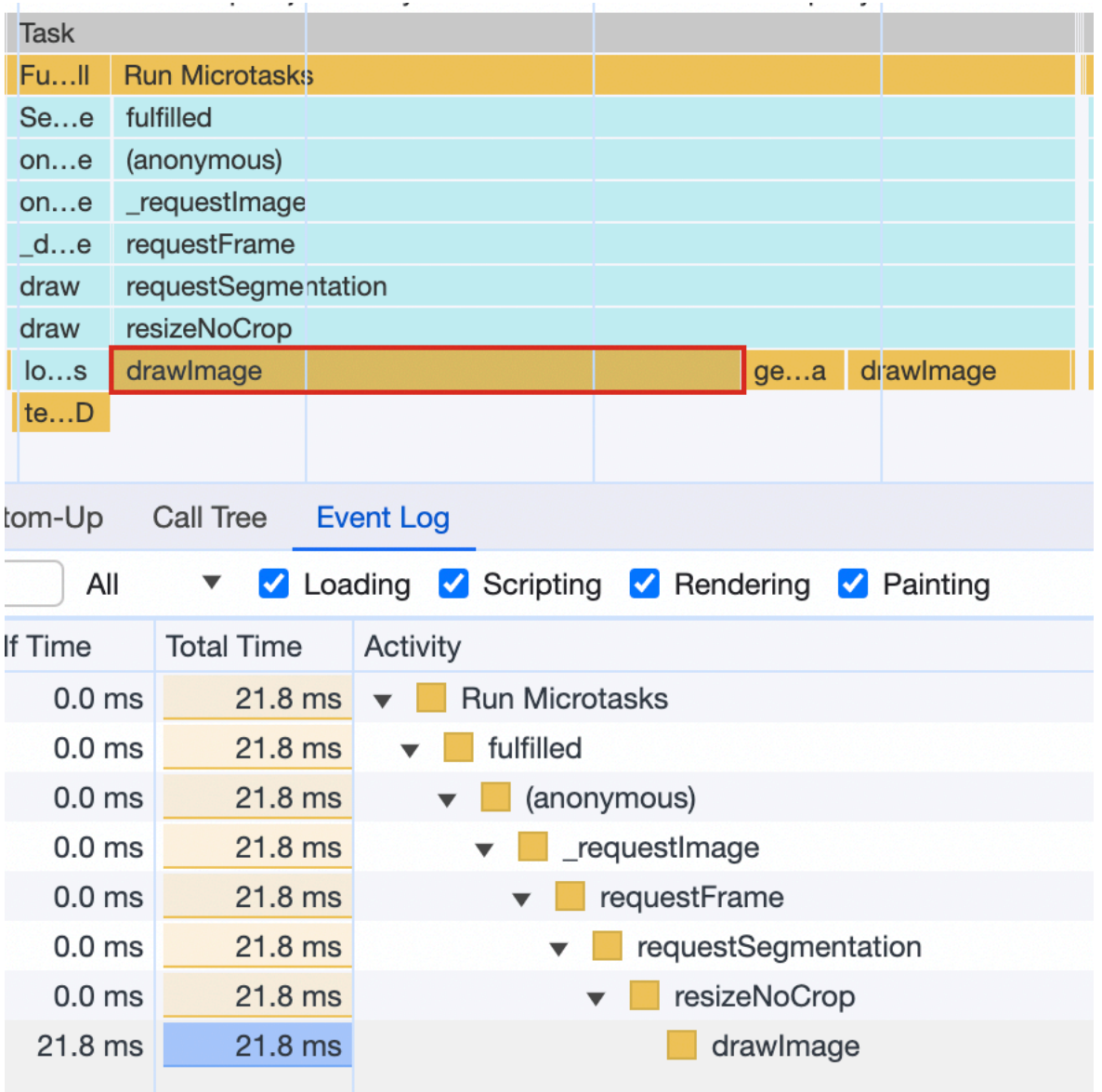
```
const canvas = new OffscreenCanvas(width, height);
const canvasCtx = canvas.getContext('2d', {willReadFrequently: true});

const effects = [background, filter, overlay];
let nextFrame = videoFrame;

for (const effect of effects) {
    effect.draw(nextFrame);
    nextFrame = effect.getCanvas();
}

canvasCtx.drawImage(nextFrame, 0, 0);
```

Проблемы

Ресайз


ON AIR


ON AIR


Task
Fu…ll
Run Microtasks
Se…e
fulfilled
on…e
(anonymous)
on…e
_requestImage
_d…e
requestFrame
draw
requestSegmentation
draw
resizeNoCrop
lo…s
drawImage
ge…a
drawImage
te…D
tom-Up
Call Tree
Event Log
All
Loading
Scripting
Rendering
Painting
lf Time
Total Time
Activity
0.0 ms
21.8 ms
Run Microtasks
0.0 ms
21.8 ms
fulfilled
0.0 ms
21.8 ms
(anonymous)
0.0 ms
21.8 ms
_requestImage
0.0 ms
21.8 ms
requestFrame
0.0 ms
21.8 ms
requestSegmentation
0.0 ms
21.8 ms
resizeNoCrop
21.8 ms
21.8 ms
drawImage

```
function resize(image, width, height) {
    const canvas = new OffscreenCanvas(width, height);
    const canvasCtx = canvas.getContext('2d');

    canvasCtx.drawImage(image, 0, 0, width, height);
    return canvasCtx.getImageData(0, 0, width, height);
}
```

```
canvasCtx.getContext('2d')
```

2d webgl webgl2 webgpu bitmaprenderer

```
canvasCtx.getContext('webgl')
```

```
// VERTEX_SHADER_SRC

precision mediump float;
attribute vec4 a_position;
attribute vec2 a_texCoord;
varying vec2 v_texCoord;

void main() {
    gl_Position = a_position;
    v_texCoord = a_texCoord;
}
```

```
// FRAGMENT_SHADER_SRC

precision mediump float;
uniform sampler2D u_image;
varying vec2 v_texCoord;

void main() {
    gl_FragColor = texture2D(u_image, v_texCoord);
}
```

```
// VERTEX_SHADER_SRC

precision mediump float;
attribute vec4 a_position;
attribute vec2 a_texCoord;
varying vec2 v_texCoord;

void main() {
    gl_Position = a_position;
    v_texCoord = a_texCoord;
}
```

```
// FRAGMENT_SHADER_SRC

precision mediump float;
uniform sampler2D u_image;
varying vec2 v_texCoord;

void main() {
    gl_FragColor = texture2D(u_image, v_texCoord);
}
```

```
const canvas = new OffscreenCanvas(160, 90);
const gl = canvas.getContext('webgl');

const vertexShader = gl.createShader(gl.VERTEX_SHADER)!;
gl.shaderSource(vertexShader, VERTEX_SHADER_SRC);
gl.compileShader(vertexShader);

const fragmentShader = gl.createShader(gl.FRAGMENT_SHADER)!;
gl.shaderSource(fragmentShader, FRAGMENT_SHADER_SRC);
gl.compileShader(fragmentShader);

const program = gl.createProgram()!;
gl.attachShader(program, vertexShader);
gl.attachShader(program, fragmentShader);

gl.linkProgram(program);
```

```
const canvas = new OffscreenCanvas(160, 90);
const gl = canvas.getContext('webgl');

const vertexShader = gl.createShader(gl.VERTEX_SHADER)!;
gl.shaderSource(vertexShader, VERTEX_SHADER_SRC);
gl.compileShader(vertexShader);

const fragmentShader = gl.createShader(gl.FRAGMENT_SHADER)!;
gl.shaderSource(fragmentShader, FRAGMENT_SHADER_SRC);
gl.compileShader(fragmentShader);

const program = gl.createProgram()!;
gl.attachShader(program, vertexShader);
gl.attachShader(program, fragmentShader);

gl.linkProgram(program);
```

```
const canvas = new OffscreenCanvas(160, 90);
const gl = canvas.getContext('webgl');

const vertexShader = gl.createShader(gl.VERTEX_SHADER)!;
gl.shaderSource(vertexShader, VERTEX_SHADER_SRC);
gl.compileShader(vertexShader);

const fragmentShader = gl.createShader(gl.FRAGMENT_SHADER)!;
gl.shaderSource(fragmentShader, FRAGMENT_SHADER_SRC);
gl.compileShader(fragmentShader);

const program = gl.createProgram()!;
gl.attachShader(program, vertexShader);
gl.attachShader(program, fragmentShader);

gl.linkProgram(program);
```

```
const positionLocation = gl.getAttribLocation(program, 'a_position');
gl.enableVertexAttribArray(positionLocation);

const positionBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, positionBuffer);
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array([
    -1.0, 1.0,
    1.0, 1.0,
    -1.0, -1.0,
    1.0, -1.0,
]), gl.STATIC_DRAW);

gl.vertexAttribPointer(positionLocation, 2, gl.FLOAT, false, 0, 0);
```

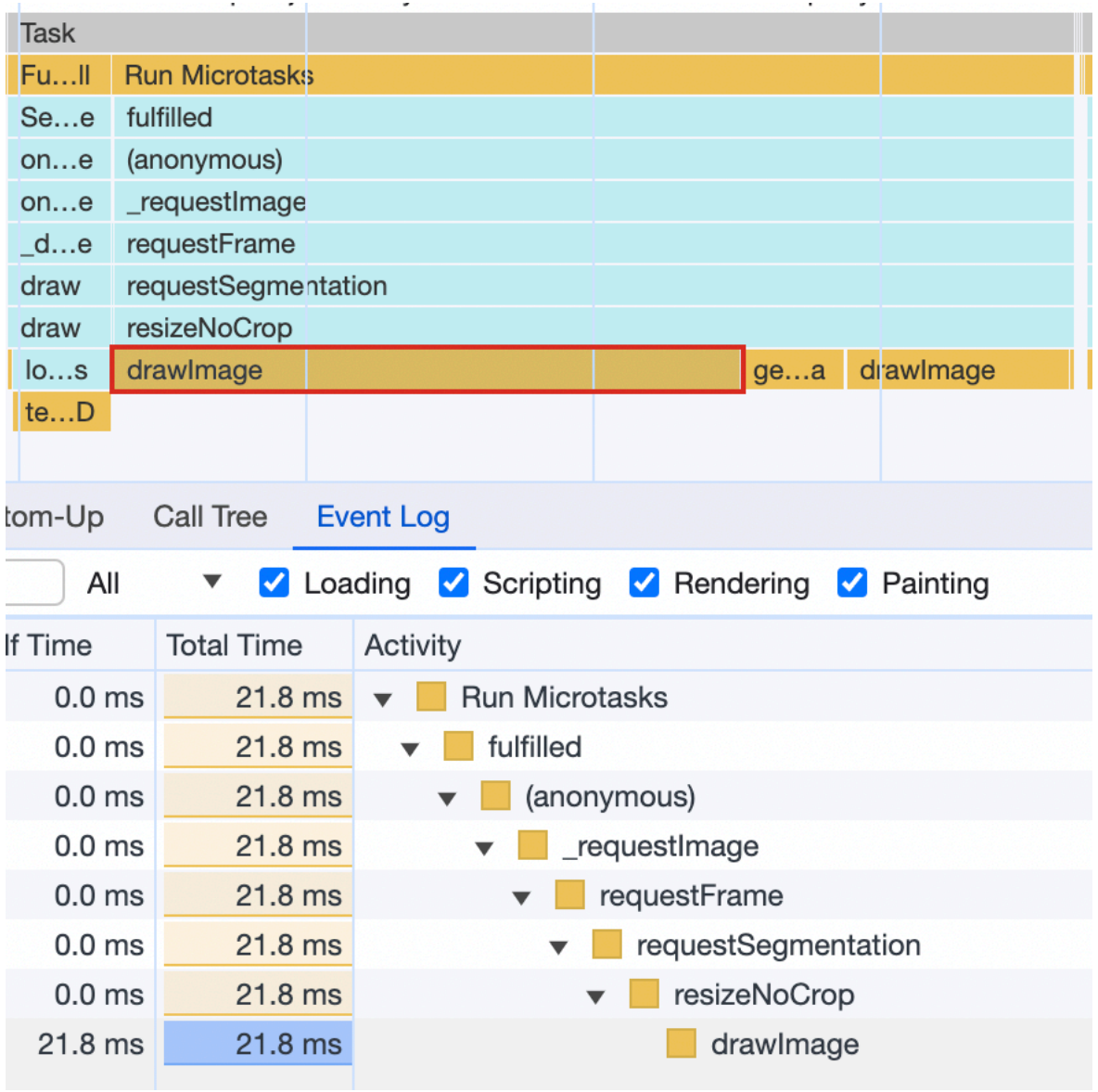
```
const texcoordLocation = gl.getAttribLocation(program, 'a_texCoord');
gl.enableVertexAttribArray(texcoordLocation);

const texcoordBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, texcoordBuffer);
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array([
    0.0, 1.0,
    1.0, 1.0,
    0.0, 0.0,
    1.0, 0.0,
]), gl.STATIC_DRAW);

gl.vertexAttribPointer(texcoordLocation, 2, gl.FLOAT, false, 0, 0);
```

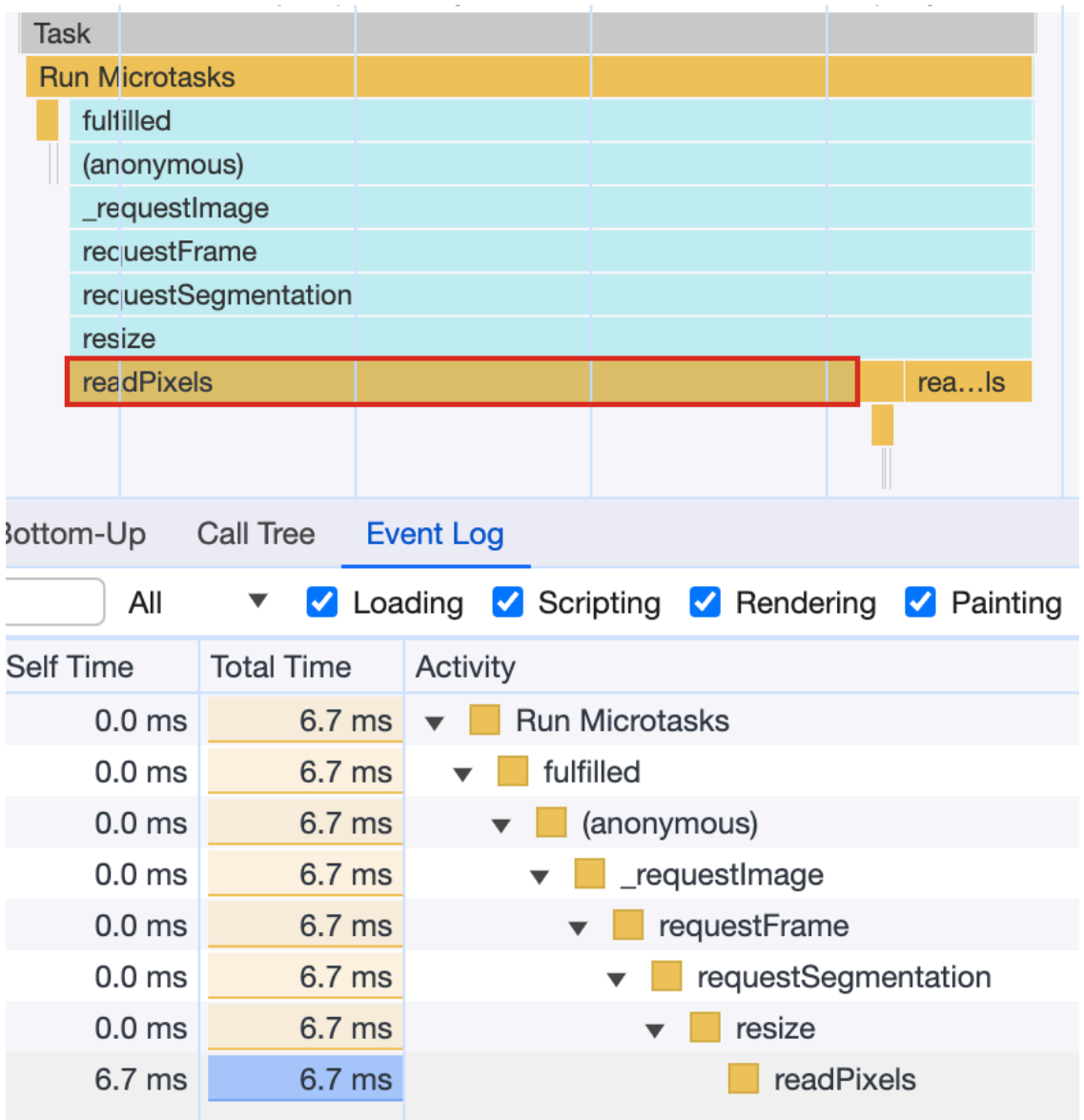
```
const dstTexture = gl.createTexture();
const srcTexture = gl.createTexture();

gl.bindTexture(gl.TEXTURE_2D, dstTexture);
gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, width, height, 0, gl.RGBA, gl.UNSIGNED_BYTE, null);
gl.framebufferTexture2D(gl.FRAMEBUFFER, gl.COLOR_ATTACHMENT0, gl.TEXTURE_2D, dstTexture, 0);

gl.bindTexture(gl.TEXTURE_2D, srcTexture);
gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, image);
```

```
gl.viewport(0, 0, width, height);
gl.clearColor(0, 0, 0, 0);
gl.clear(gl.COLOR_BUFFER_BIT);

gl.useProgram(program);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, 4);

const pixels = new Uint8ClampedArray(4 * width * height);
gl.readPixels(0, 0, width, height, gl.RGBA, gl.UNSIGNED_BYTE, pixels);

return new ImageData(pixels, width, height);
```

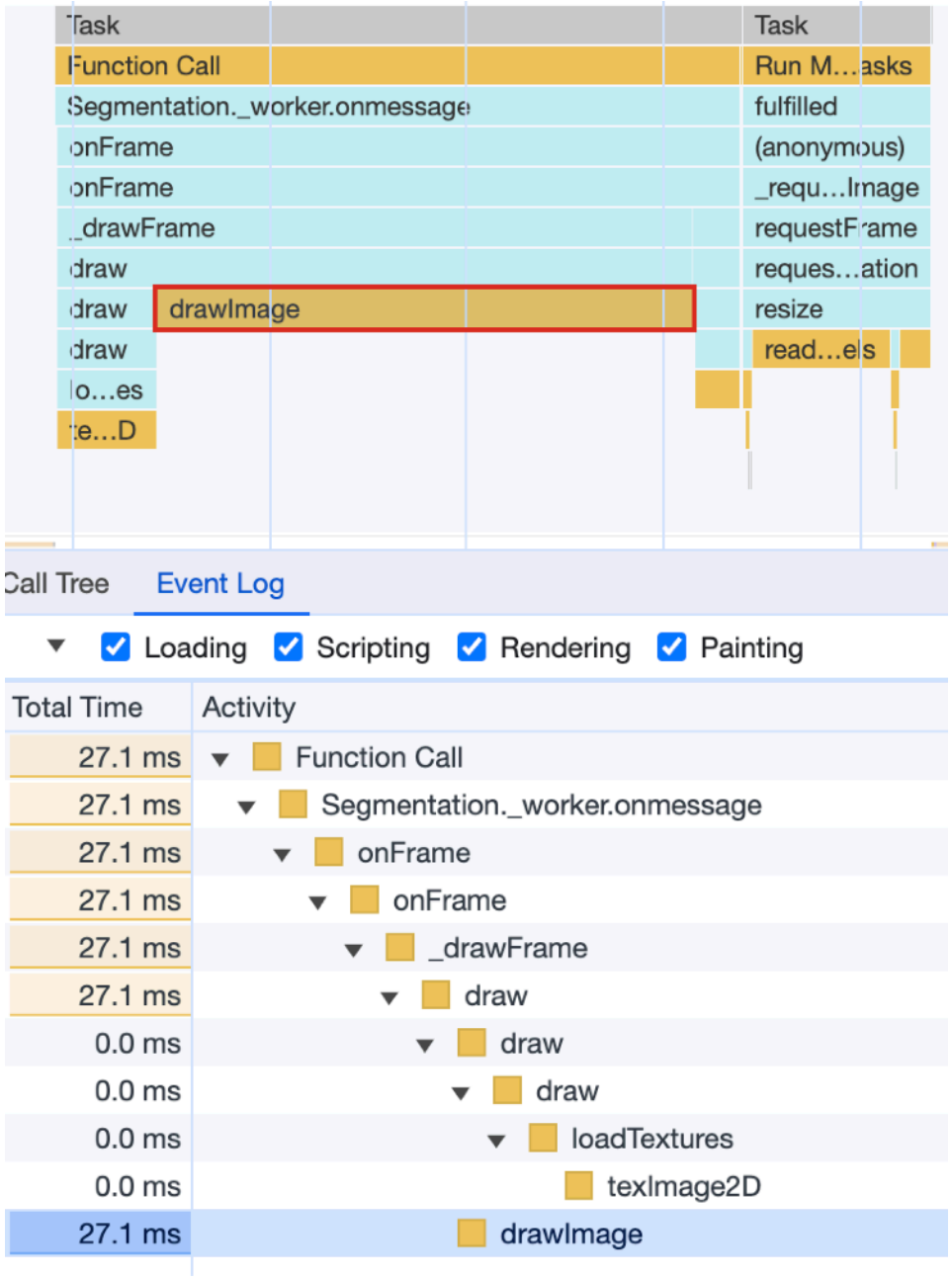
```
gl.viewport(0, 0, width, height);
gl.clearColor(0, 0, 0, 0);
gl.clear(gl.COLOR_BUFFER_BIT);

gl.useProgram(program);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, 4);

const pixels = new Uint8ClampedArray(4 * width * height);
gl.readPixels(0, 0, width, height, gl.RGBA, gl.UNSIGNED_BYTE, pixels);

return new ImageData(pixels, width, height);
```

```
gl.viewport(0, 0, width, height);
gl.clearColor(0, 0, 0, 0);
gl.clear(gl.COLOR_BUFFER_BIT);

gl.useProgram(program);
gl.drawArrays(gl.TRIANGLE_STRIP, 0, 4);

const pixels = new Uint8ClampedArray(4 * width * height);
gl.readPixels(0, 0, width, height, gl.RGBA, gl.UNSIGNED_BYTE, pixels);

return new ImageData(pixels, width, height);
```


Task
Fu…ll
Run Microtasks
Se…e
fulfilled
on…e
(anonymous)
on…e
_requestImage
_d…e
requestFrame
draw
requestSegmentation
draw
resizeNoCrop
lo…s
drawImage
ge…a
drawImage
te…D
tom-Up
Call Tree
Event Log
All
Loading
Scripting
Rendering
Painting
lf Time
Total Time
Activity
0.0 ms
21.8 ms
Run Microtasks
0.0 ms
21.8 ms
fulfilled
0.0 ms
21.8 ms
(anonymous)
0.0 ms
21.8 ms
_requestImage
0.0 ms
21.8 ms
requestFrame
0.0 ms
21.8 ms
requestSegmentation
0.0 ms
21.8 ms
resizeNoCrop
21.8 ms
21.8 ms
drawImage


Task
Run Microtasks
fulfilled
(anonymous)
_requestImage
requestFrame
requestSegmentation
resize
readPixels
rea…ls
Bottom-Up
Call Tree
Event Log
All
Loading
Scripting
Rendering
Painting
Self Time
Total Time
Activity
0.0 ms
6.7 ms
Run Microtasks
0.0 ms
6.7 ms
fulfilled
0.0 ms
6.7 ms
(anonymous)
0.0 ms
6.7 ms
_requestImage
0.0 ms
6.7 ms
requestFrame
0.0 ms
6.7 ms
requestSegmentation
0.0 ms
6.7 ms
resize
6.7 ms
6.7 ms
readPixels

Композиция


Task
Task
Function Call
Run M…asks
Segmentation._worker.onmessage
fulfilled
onFrame
(anonymous)
onFrame
_requ…Image
_drawFrame
requestFrame
draw
reques…ation
draw
drawImage
resize
draw
read…es
lo…es
te…D
Call Tree
Event Log
Loading
Scripting
Rendering
Painting
Total Time
Activity
27.1 ms
Function Call
27.1 ms
Segmentation._worker.onmessage
27.1 ms
onFrame
27.1 ms
onFrame
27.1 ms
_drawFrame
27.1 ms
draw
0.0 ms
draw
0.0 ms
draw
0.0 ms
loadTextures
0.0 ms
texImage2D
27.1 ms
drawImage

```
const canvas = new OffscreenCanvas(width, height);
const canvasCtx = canvas.getContext('2d', {willReadFrequently: true});

canvasCtx.clearRect(width, height);
canvasCtx.drawImage(videoFrame, 0, 0);

canvasCtx.globalCompositeOperation = 'destination-in';
canvasCtx.drawImage(segmentationMask, 0, 0);

canvasCtx.globalCompositeOperation = 'destination-over';
canvasCtx.drawImage(background, 0, 0);
```

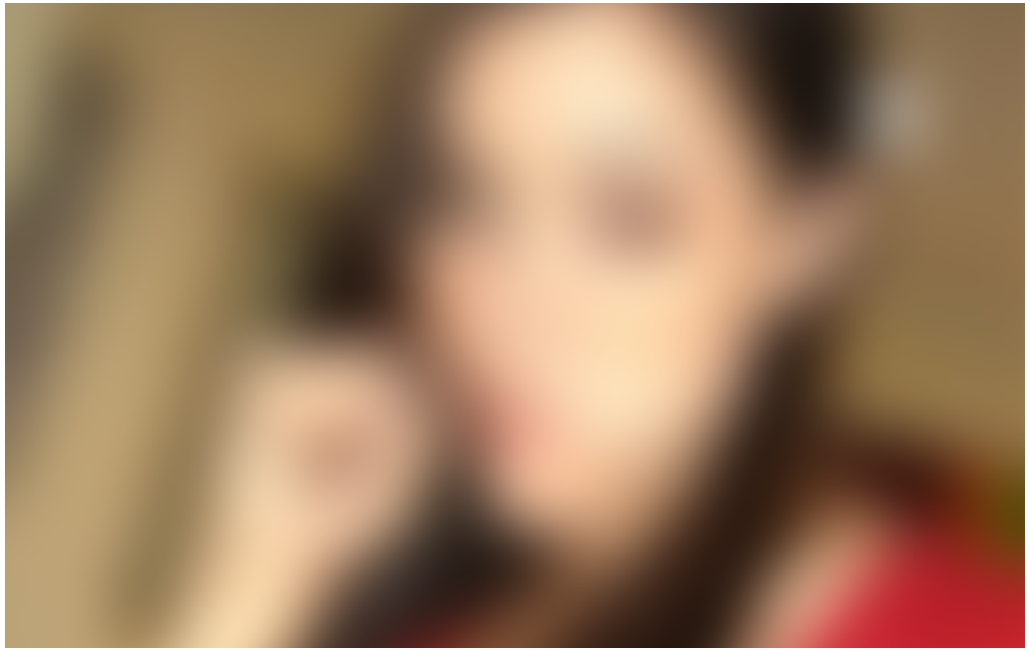
```
const dstTexture = gl.createTexture();
const srcTexture1 = gl.createTexture();
const srcTexture2 = gl.createTexture();
const srcTexture3 = gl.createTexture();

gl.bindTexture(gl.TEXTURE_2D, srcTexture1);
gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, image1);

gl.bindTexture(gl.TEXTURE_2D, srcTexture2);
gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, image2);

gl.bindTexture(gl.TEXTURE_2D, srcTexture3);
gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, image3);
```

Image + Image

VideoFrame → Canvas

VideoFrame → Canvas → Canvas → Canvas

Blur



```
const canvas = new OffscreenCanvas(width, height);
const canvasCtx = canvas.getContext('2d', {willReadFrequently: true});

// ...

canvasCtx.filter = 'blur(20px)';
canvasCtx.drawImage(videoFrame, 0, 0);
canvasCtx.filter = 'none';
```

glsl-fast-gaussian-blur

https://github.com/Experience-Monks/glsl-fast-gaussian-blur

WebGL🔥

Итоги

- Рисовать на canvas просто
- Быстро рисовать – сложно
- Используйте WebGL
- Избегайте перемещения текстур
- Не бойтесь изобретать сложные решения
- https://webglfundamentals.org/

Всё

Николай Васильчук
@vasilchuk

VideoTech
2023

@vasilchuk

VK Team

Хабр