import React, { useRef, useEffect, useState } from "react";

import Webcam from "react-webcam";

import "@tensorflow/tfjs-backend-webgl";

import \* as faceLandmarksDetection from "@tensorflow-models/face-landmarks-detection";

const videoDims = { width: 640, height: 480 };

const baseFilters = [

  { value: "all", label: "All", image: "/all-filter.png" },

  { value: "hat", label: "Hat Only", image: "/hat.png" },

  { value: "shades", label: "Shades Only", image: "/shades.png" },

  { value: "eyes", label: "Eye Color", image: "/eye-color.png" },

];

const repeatedFilters = Array(50).fill(baseFilters).flat(); // Repeat 50x for long scroll

export default function FaceFilterApp() {

  const webcamRef = useRef(null);

  const carouselRef = useRef(null);

  const [model, setModel] = useState(null);

  const [faces, setFaces] = useState([]);

  const [selectedFilter, setSelectedFilter] = useState("all");

  const [loadingProgress, setLoadingProgress] = useState(0);

  useEffect(() => {

    async function loadModel() {

      let progress = 0;

      const interval = setInterval(() => {

        progress += Math.random() \* 20;

        if (progress >= 100) progress = 100;

        setLoadingProgress(Math.min(Math.round(progress), 100));

      }, 200);

      const detector = await faceLandmarksDetection.createDetector(

        faceLandmarksDetection.SupportedModels.MediaPipeFaceMesh,

        { runtime: "tfjs", maxFaces: 10 }

      );

      setModel(detector);

      setLoadingProgress(100);

      clearInterval(interval);

    }

    loadModel();

  }, []);

  useEffect(() => {

    let rafId;

    async function detect() {

      if (model && webcamRef.current?.video?.readyState === 4) {

        const video = webcamRef.current.video;

        const predictions = await model.estimateFaces(video);

        setFaces(predictions);

      }

      rafId = requestAnimationFrame(detect);

    }

    detect();

    return () => cancelAnimationFrame(rafId);

  }, [model]);

  // Center to middle group on mount

  useEffect(() => {

    const carousel = carouselRef.current;

    if (carousel) {

      const middleIndex = Math.floor(repeatedFilters.length / 2);

      const button = carousel.children[middleIndex];

      setTimeout(() => {

        if (button) {

          const scrollLeft =

            button.offsetLeft - (carousel.offsetWidth - button.offsetWidth) / 2;

          carousel.scrollLeft = scrollLeft;

        }

      }, 100);

    }

  }, []);

  // Swipe-to-center select

  useEffect(() => {

    const carousel = carouselRef.current;

    if (!carousel) return;

    let timeoutId;

    const handleScroll = () => {

      clearTimeout(timeoutId);

      timeoutId = setTimeout(() => {

        const scrollLeft = carousel.scrollLeft;

        const centerX = scrollLeft + carousel.offsetWidth / 2;

        let closest = null;

        let minDist = Infinity;

        repeatedFilters.forEach((filter, i) => {

          const btn = carousel.children[i];

          const btnCenter = btn.offsetLeft + btn.offsetWidth / 2;

          const dist = Math.abs(centerX - btnCenter);

          if (dist < minDist) {

            minDist = dist;

            closest = filter.value;

          }

        });

        if (closest !== selectedFilter) setSelectedFilter(closest);

      }, 100);

    };

    carousel.addEventListener("scroll", handleScroll);

    return () => {

      clearTimeout(timeoutId);

      carousel.removeEventListener("scroll", handleScroll);

    };

  }, [selectedFilter]);

  const overlays = faces.map((face, idx) => {

    const lm = face?.keypoints;

    if (!lm || lm.length < 264) return null;

    const [lx, ly] = [lm[33].x, lm[33].y];

    const [rx, ry] = [lm[263].x, lm[263].y];

    const angle = (Math.atan2(ry - ly, rx - lx) \* 180) / Math.PI;

    const glassesW = Math.hypot(rx - lx, ry - ly) \* 1.8;

    const glassesH = glassesW / 2;

    const glassesX = lx + (rx - lx) / 2 - glassesW / 2;

    const glassesY = ly - glassesH / 3;

    const hatW = glassesW \* 1.6;

    const hatH = hatW \* 0.8;

    const hatX = glassesX - glassesW \* 0.28;

    const hatY = glassesY - hatH \* 0.8;

    const eyeStyle = (x, y) => ({

      left: `${x - 25}px`,

      top: `${y - 25}px`,

      width: "50px",

      height: "50px",

      borderRadius: "50%",

      backgroundColor: "green",

      position: "absolute",

      opacity: 0.7,

    });

    return (

      <React.Fragment key={idx}>

        {(selectedFilter === "hat" || selectedFilter === "all") && (

          <img

            src="/hat.png"

            alt="hat"

            style={{

              position: "absolute",

              left: `${hatX}px`,

              top: `${hatY}px`,

              width: `${hatW}px`,

              height: `${hatH}px`,

              transform: `rotate(${angle}deg)`,

              transformOrigin: "bottom center",

              pointerEvents: "none",

            }}

          />

        )}

        {(selectedFilter === "shades" || selectedFilter === "all") && (

          <img

            src="/shades.png"

            alt="shades"

            style={{

              position: "absolute",

              left: `${glassesX}px`,

              top: `${glassesY}px`,

              width: `${glassesW}px`,

              height: `${glassesH}px`,

              transform: `rotate(${angle}deg)`,

              transformOrigin: "center",

              pointerEvents: "none",

            }}

          />

        )}

        {(selectedFilter === "eyes" || selectedFilter === "all") && (

          <>

            <div style={eyeStyle(lx, ly)} />

            <div style={eyeStyle(rx, ry)} />

          </>

        )}

      </React.Fragment>

    );

  });

  return (

    <div

      style={{

        position: "relative",

        width: videoDims.width,

        height: videoDims.height + 100,

        margin: "auto",

        fontFamily: "sans-serif",

      }}

    >

      {/\* Webcam + overlays \*/}

      <div

        style={{

          width: `${videoDims.width}px`,

          height: `${videoDims.height}px`,

          position: "relative",

          backgroundColor: "#000",

        }}

      >

        <Webcam

          ref={webcamRef}

          audio={false}

          width={videoDims.width}

          height={videoDims.height}

          style={{ position: "absolute", left: 0, top: 0 }}

        />

        {overlays}

        {!model && (

          <div

            style={{

              position: "absolute",

              top: 0,

              left: 0,

              width: "100%",

              height: "100%",

              backgroundColor: "rgba(0, 0, 0, 0.5)",

              color: "#fff",

              display: "flex",

              justifyContent: "center",

              alignItems: "center",

              flexDirection: "column",

              zIndex: 10,

            }}

          >

            <div

              style={{

                width: "50px",

                height: "50px",

                border: "5px solid #fff",

                borderTop: "5px solid #3498db",

                borderRadius: "50%",

                animation: "spin 1s linear infinite",

              }}

            />

            <span style={{ marginTop: "10px" }}>{loadingProgress}%</span>

          </div>

        )}

      </div>

      {/\* Filter carousel \*/}

      <div

        ref={carouselRef}

        style={{

          marginTop: "10px",

          width: `${videoDims.width}px`,

          overflowX: "scroll",

          display: "flex",

          padding: "10px 0",

          scrollSnapType: "x mandatory",

          scrollBehavior: "smooth",

          WebkitOverflowScrolling: "touch",

          touchAction: "pan-x",

          scrollbarWidth: "none",

          msOverflowStyle: "none",

        }}

      >

        {repeatedFilters.map((filter, i) => (

          <button

            key={`${filter.value}-${i}`}

            style={{

              flex: "0 0 auto",

              width: "70px",

              height: "70px",

              margin: "0 10px",

              borderRadius: "50%",

              background: `url(${filter.image}) center/cover no-repeat`,

              border:

                filter.value === selectedFilter

                  ? "3px solid #3498db"

                  : "2px solid #ccc",

              scrollSnapAlign: "center",

              position: "relative",

              cursor: "grab",

            }}

            title={filter.label}

          >

            <span

              style={{

                position: "absolute",

                bottom: "5px",

                left: "50%",

                transform: "translateX(-50%)",

                fontSize: "10px",

                color: "#fff",

                background: "rgba(0,0,0,0.5)",

                padding: "2px 5px",

                borderRadius: "4px",

              }}

            >

              {filter.label}

            </span>

          </button>

        ))}

      </div>

      {/\* Center indicator \*/}

      <div

        style={{

          position: "absolute",

          bottom: "95px",

          left: "50%",

          transform: "translateX(-50%)",

          width: "2px",

          height: "60px",

          backgroundColor: "rgba(255,255,255,0.3)",

        }}

      />

      <style>{`

        @keyframes spin {

          0% { transform: rotate(0deg); }

          100% { transform: rotate(360deg); }

        }

        div::-webkit-scrollbar {

          display: none;

        }

      `}</style>

    </div>

  );

}