Visión por Computador

Virtual Fitting Room

Autores

Javier Castilla Moreno Asmae Ez Zaim Driouch

Índice general

Capítulo 1

Introducción

La industria del comercio electrónico ha experimentado un crecimiento exponencial en las últimas décadas, transformando radicalmente la manera en que los consumidores adquieren productos. Sin embargo, el sector de la moda y la vestimenta enfrenta un desafío particular: la imposibilidad de que los usuarios prueben físicamente las prendas antes de realizar una compra. Esta limitación genera tasas de devolución superiores al 30 % en ventas online de ropa, incrementando significativamente los costes operativos para los comercios electrónicos y deteriorando la experiencia del usuario.

Los avances recientes en visión por computador, procesamiento de imágenes en tiempo real y renderizado 3D en navegadores web han abierto nuevas posibilidades para abordar esta problemática. Las tecnologías de detección de pose mediante redes neuronales, como MediaPipe, combinadas con motores de renderizado 3D como Three.js, permiten crear experiencias interactivas directamente en el navegador sin necesidad de hardware especializado o aplicaciones nativas complejas.

1.1. Motivación del trabajo

La motivación principal de este proyecto radica en desarrollar una solución tecnológica que reduzca la brecha entre la experiencia de compra física y digital. Un probador virtual (*Virtual Fitting Room*) no solo mejora la confianza del consumidor al momento de la compra, permitiéndole visualizar cómo le quedaría una prenda antes de adquirirla, sino que también representa beneficios tangibles para los comercios electrónicos mediante la reducción de devoluciones y el incremento en las tasas de conversión.

Desde una perspectiva técnica y académica, este proyecto representa una oportunidad única para integrar múltiples disciplinas de la ingeniería informática: visión por computador, gráficos 3D, desarrollo web full-stack, procesamiento de gestos y diseño de interfaces de usuario. La convergencia de estas áreas permite aplicar conocimientos teóricos adquiridos durante el grado en un contexto práctico y actual, enfrentando desafíos reales de implementación que incluyen procesamiento en tiempo real, optimización de rendimiento y diseño de experiencia de usuario.

Adicionalmente, el desarrollo de interfaces basadas en gestos representa un paradigma de interacción innovador que elimina la necesidad de dispositivos de entrada tradicionales. En el contexto de un probador virtual, esto resulta especialmente relevante ya que el usuario puede interactuar con el sistema de manera natural mientras se visualiza a sí mismo, sin necesidad de interrumpir la experiencia para utilizar teclado o ratón.

1.2. Contexto tecnológico

El ecosistema tecnológico actual proporciona herramientas maduras y accesibles para el desarrollo de aplicaciones de realidad aumentada en navegadores web. MediaPipe, desarrollado por Google, ofrece soluciones de visión por computador pre-entrenadas que detectan 33 landmarks corporales en tiempo real con alta precisión. Three.js, por su parte, es una biblioteca consolidada que abstrae la complejidad de WebGL, facilitando el renderizado de escenas 3D complejas con rendimiento optimizado.

El desarrollo de aplicaciones web progresivas (PWA) utilizando frameworks modernos como Angular permite crear experiencias ricas e interactivas que funcionan directamente en el navegador, eliminando barreras de instalación y garantizando compatibilidad multiplataforma. La combinación de estas tecnologías hace posible desarrollar un probador virtual accesible desde cualquier dispositivo con cámara y navegador web moderno.

1.3. Alcance del proyecto

Este trabajo se centra en el diseño e implementación de un sistema de probador virtual funcional que permite a los usuarios visualizar prendas de vestir tridimensionales sobre su propia figura en tiempo real. El sistema incluye un catálogo de prendas organizadas por categorías y género, una interfaz de navegación basada en gestos manuales, y modos de depuración para desarrollo.

El proyecto no contempla la implementación de funcionalidades de comercio electrónico (gestión de inventario, carrito de compras, procesamiento de pagos), centrándose exclusivamente en la experiencia de prueba virtual y la validación técnica de la viabilidad de este tipo de sistemas en entornos web.

1.4. Estructura del documento

El presente documento se organiza de la siguiente manera: el Capítulo 2 detalla los objetivos específicos y la descripción funcional de la aplicación desarrollada; el Capítulo 3 aborda las decisiones de diseño del sistema; el Capítulo 4 describe la arquitectura de software implementada; el Capítulo 5 presenta el modelo de datos utilizado; el Capítulo 6 documenta el proceso de desarrollo y las tecnologías empleadas; el Capítulo 7 muestra las interfaces de usuario implementadas; finalmente, el Capítulo 8 expone las conclusiones, lecciones aprendidas y propuestas de trabajo futuro.

Capítulo 2

Descripción de la aplicación incluyendo objetivos

2.1. Descripción general de la aplicación

Virtual Fitting Room es una aplicación web progresiva diseñada para permitir a los usuarios probarse prendas de vestir virtuales en tiempo real utilizando únicamente la cámara de su dispositivo. El sistema captura el video del usuario, detecta su postura corporal mediante algoritmos de visión por computador, y superpone modelos tridimensionales de prendas que se ajustan dinámicamente a su anatomía mientras se mueve.

La aplicación está desarrollada con Angular y TypeScript, utilizando MediaPipe para la detección de landmarks corporales y Three.js para el renderizado de escenas 3D. La interfaz de usuario está completamente controlada por gestos manuales, permitiendo una experiencia "manos libres" donde el usuario puede navegar por el catálogo de prendas, seleccionar artículos y probar combinaciones sin necesidad de tocar el dispositivo.

El flujo principal de uso consiste en: (1) selección inicial de género para filtrar el catálogo de prendas, (2) exploración de categorías de ropa (parte superior e inferior del cuerpo), (3) navegación por prendas específicas mediante gestos de apuntar y (4) visualización en tiempo real de las prendas sobre el cuerpo del usuario.

2.2. Objetivos de la aplicación

2.2.1. Objetivo general

Desarrollar un sistema de probador virtual basado en tecnologías web que permita visualizar prendas de vestir tridimensionales sobre la figura del usuario en tiempo real, proporcionando una experiencia interactiva, fluida y accesible desde cualquier dispositivo con cámara y navegador moderno.

2.2.2. Objetivos específicos

Sistema de captura y detección de pose

Implementar un pipeline de procesamiento de video en tiempo real que utilice MediaPipe Pose para detectar 33 landmarks corporales del usuario con una frecuencia mínima de 30 FPS. El sistema debe proporcionar coordenadas tridimensionales normalizadas de

puntos anatómicos clave (hombros, caderas, rodillas, etc.) que sirvan como anclajes para el posicionamiento de las prendas virtuales.

Este módulo debe ser capaz de manejar variaciones en condiciones de iluminación, distancias variables del usuario a la cámara, y diferentes constituciones físicas, garantizando robustez en la detección incluso con movimientos naturales del usuario.

Motor de renderizado 3D

Desarrollar un sistema de renderizado basado en Three.js capaz de:

- Cargar y gestionar modelos 3D de prendas en formato GLTF/GLB
- Posicionar y escalar las prendas dinámicamente según los landmarks detectados
- Mantener la alineación vertical y horizontal entre prendas de la parte superior e inferior del cuerpo
- Superponer correctamente las prendas sobre el video en tiempo real mediante composición de capas
- Optimizar el rendimiento del renderizado para mantener fluidez visual

El motor debe implementar lógica específica para cada tipo de prenda, mapeando landmarks corporales a puntos de anclaje apropiados (por ejemplo, hombros para camisetas, caderas para pantalones).

Catálogo de prendas estructurado

Diseñar e implementar un sistema de gestión de prendas que:

- Organice las prendas por categorías jerárquicas (parte superior del cuerpo, parte inferior del cuerpo)
- Clasifique las prendas por género (masculino, femenino)
- Soporte subcategorías específicas (camisetas, camisas, pantalones, faldas, etc.)
- Incluya metadatos para cada prenda (nombre, ruta del modelo 3D, imagen de vista previa)
- Permita la selección independiente de prendas por categoría
- Incorpore la opción de "sin prenda" para cada categoría

El catálogo debe generarse automáticamente mediante scripts que escanean la estructura de carpetas de modelos 3D, facilitando la adición de nuevas prendas sin modificar código manualmente.

Interfaz de usuario basada en gestos

Implementar un sistema de reconocimiento de gestos manuales que permita:

- **Navegación por categorías**: Detección de gesto de apuntar (*pointing*) sobre zonas específicas de la interfaz para cambiar entre categorías de ropa o volver a la selección de género
- **Selección de prendas**: Navegación horizontal por el carrusel de prendas mediante apuntar a áreas laterales, con retroalimentación visual del elemento actualmente señalado
- **Confirmación de selección**: Detección automática cuando el usuario apunta al centro del carrusel donde se encuentra la prenda destacada

El sistema debe implementar zonas de interacción configurables, umbrales de confianza para evitar falsos positivos, y mecanismos de debounce para prevenir activaciones múltiples accidentales.

Funcionalidades complementarias

Desarrollar características adicionales que mejoren la usabilidad y el desarrollo del sistema:

- **Modo debug**: Visualización de landmarks detectados, skeleton overlay, zonas de interacción para gestos, y métricas de rendimiento en tiempo real
- **Selección de género inicial**: Pantalla de bienvenida para filtrar el catálogo según el género del usuario

2.3. Requisitos funcionales

1. El sistema debe detectar la pose del usuario en tiempo real
2. Las prendas virtuales deben actualizarse en sincronía con el movimiento del usuario
3. La interfaz debe responder a gestos en tiempo real
4. El usuario debe poder cambiar de prenda sin interrupciones visuales perceptibles
5. La aplicación debe mantener un rendimiento mínimo de 30 FPS en dispositivos modernos
6. Los modelos 3D deben cargarse de forma asíncrona sin bloquear la interfaz
7. El sistema debe proporcionar feedback visual claro del estado actual (prenda seleccionada, gesto detectado)

2.4. Requisitos no funcionales

1. **Usabilidad**: La interfaz debe ser intuitiva y no requerir instrucciones extensas para su uso básico
2. **Rendimiento**: El sistema debe funcionar fluidamente en navegadores modernos (Chrome, Firefox, Safari) en dispositivos con cámara web o frontal
3. **Escalabilidad**: La arquitectura debe facilitar la adición de nuevas prendas y categorías sin modificaciones estructurales
4. **Mantenibilidad**: El código debe seguir principios SOLID, estar modularizado en servicios independientes y contar con separación clara de responsabilidades
5. **Accesibilidad**: La aplicación debe ser accesible mediante navegador web sin requerir instalación de software adicional
6. **Robustez**: El sistema debe manejar adecuadamente casos edge como pérdida temporal de detección

Capítulo 3

Diseño

Este capítulo presenta las decisiones de diseño fundamentales que guiaron el desarrollo del sistema Virtual Fitting Room. Se abordan desde el diseño arquitectónico de alto nivel hasta las decisiones específicas de implementación, justificando cada elección técnica en función de los requisitos establecidos.

3.1. Decisiones de diseño principales

3.1.1. Aplicación web vs. aplicación nativa

Se optó por desarrollar una aplicación web progresiva (PWA) en lugar de aplicaciones nativas por las siguientes razones:

- **Accesibilidad multiplataforma**: Una aplicación web funciona en cualquier dispositivo con navegador moderno (desktop, móvil, tablet) sin necesidad de desarrollar versiones específicas para cada plataforma
- **Distribución inmediata**: Los usuarios pueden acceder al sistema mediante una URL sin proceso de instalación, eliminando barreras de entrada
- **Actualizaciones transparentes**: Las mejoras y correcciones se despliegan instantáneamente sin requerir actualizaciones manuales del usuario
- **Madurez tecnológica**: Las APIs Web modernas (WebRTC para cámara, WebGL para 3D, Web Workers para procesamiento paralelo) ofrecen capacidades suficientes para implementar todas las funcionalidades requeridas

3.1.2. Framework de desarrollo

Se seleccionó Angular como framework de desarrollo por:

- **Arquitectura basada en servicios**: Angular promueve la inyección de dependencias y la separación de responsabilidades mediante servicios singleton, facilitando la modularización del código
- **TypeScript nativo**: El tipado estático de TypeScript reduce errores en tiempo de desarrollo y mejora la mantenibilidad del código

- **Gestión de estado reactiva**: El uso de RxJS y observables permite manejar flujos de datos asíncronos de manera elegante, esencial para el procesamiento en tiempo real
- **Ecosistema maduro**: Angular cuenta con herramientas consolidadas de CLI, testing y optimización de producción

3.1.3. Pipeline de visión por computador

Para la detección de pose se eligió MediaPipe Pose sobre alternativas como PoseNet o MoveNet debido a:

- **Precisión superior**: MediaPipe detecta 33 landmarks con alta precisión, incluyendo puntos faciales y de manos que permiten reconocimiento de gestos
- **Coordenadas 3D**: Proporciona coordenadas normalizadas en tres dimensiones (x, y, z), facilitando el cálculo de distancias y profundidad relativa
- **Optimización para web**: MediaPipe está optimizado para ejecución en navegadores mediante WebAssembly y aceleración GPU con WebGL
- **Modelo único**: Un solo modelo detecta pose completa y landmarks de manos, simplificando la integración

3.1.4. Motor de renderizado 3D

Three.js fue seleccionado como motor de renderizado 3D por:

- **Abstracción de WebGL**: Simplifica la complejidad de WebGL manteniendo control granular cuando es necesario
- **Soporte GLTF/GLB**: Manejo nativo de formatos estándar de la industria para modelos 3D
- **Sistema de escenas**: Estructura jerárquica de objetos 3D que facilita la organización y transformación de múltiples prendas
- **Rendimiento optimizado**: Sistema de frustum culling, gestión eficiente de materiales y geometrías
- **Comunidad activa**: Amplia documentación, ejemplos y soporte de la comunidad

3.2. Arquitectura de alto nivel

El sistema sigue una arquitectura de capas claramente diferenciadas:

3.2.1. Capa de presentación

Componentes Angular responsables de la interfaz de usuario y la interacción visual. Esta capa incluye:

- Componente principal de cámara y renderizado
- Sidebar de categorías con navegación por gestos
- Carrusel de prendas con selector circular
- Overlay de debug con métricas y visualizaciones
- Pantalla de selección de género inicial

3.2.2. Capa de lógica de negocio

Servicios singleton que encapsulan la lógica de la aplicación:

- **GarmentCatalogService**: Gestión del catálogo de prendas, filtrado por género y categoría
- **PoseDetectionService**: Procesamiento de video con MediaPipe y detección de landmarks
- **GestureRecognitionService**: Reconocimiento de gestos manuales (pointing, swipe) con zonas configurables
- **ThreeService**: Gestión de la escena 3D, carga de modelos y posicionamiento de prendas
- **ClothingPositionService**: Cálculo de transformaciones (posición, rotación, escala) según landmarks
- **StateManagementService**: Gestión del estado de la aplicación (género, categoría, prendas seleccionadas)
- **StabilizationService**: Aplicación de filtros de suavizado (EWMA/Kalman) sobre landmarks

3.2.3. Capa de datos

Modelos e interfaces TypeScript que definen la estructura de datos:

- Interfaces de prendas (Garment, GarmentCategory, GarmentGender)
- Modelos de landmarks y pose (PoseLandmark, NormalizedLandmark)
- Tipos de gestos y eventos de interacción
- Configuración del sistema (zonas de gestos, parámetros de filtrado)

3.3. Patrones de diseño aplicados

3.3.1. Singleton (Servicios Angular)

Todos los servicios se registran con `providedIn: 'root'`, garantizando una única instancia compartida en toda la aplicación. Esto es esencial para mantener estado consistente y evitar múltiples instancias de procesamiento intensivo (MediaPipe, Three.js).

3.3.2. Observer (RxJS Observables)

El flujo de datos entre servicios utiliza observables de RxJS para comunicación reactiva:

- Emisión de nuevos frames de video procesados
- Notificación de gestos detectados
- Cambios en el estado de la aplicación
- Actualización de landmarks filtrados

Este patrón desacopla productores y consumidores de datos, facilitando la extensibilidad.

3.3.3. Strategy (Posicionamiento de prendas)

Cada tipo de prenda implementa una estrategia específica de posicionamiento:

- Prendas superiores: Ancladas a landmarks de hombros (11, 12) y caderas (23, 24)
- Prendas inferiores: Ancladas a landmarks de caderas (23, 24) y rodillas (25, 26)

Esto permite añadir nuevos tipos de prendas (accesorios, calzado) implementando nuevas estrategias sin modificar código existente.

3.3.4. Factory (Generación de modelos)

Se implementa un script generador (`generate-models-list.mjs`) que actúa como factory, escaneando automáticamente la estructura de carpetas de modelos 3D y generando código TypeScript con el catálogo completo. Este patrón facilita la adición de nuevas prendas sin edición manual de código.

3.4. Flujo de datos del sistema

3.4.1. Pipeline de procesamiento en tiempo real

El flujo principal de procesamiento sigue esta secuencia:

1. **Captura de video**: La cámara del dispositivo proporciona frames continuos mediante MediaStream API
2. **Detección de pose**: MediaPipe procesa cada frame y extrae 33 landmarks con coordenadas 3D normalizadas

3. **Reconocimiento de gestos**: Se analizan landmarks de manos (índices 15-22) para detectar gestos específicos
4. **Cálculo de transformaciones**: Según los landmarks estabilizados, se calculan posiciones, rotaciones y escalas para cada prenda activa
5. **Actualización de escena 3D**: Los modelos 3D se transforman en la escena de Three.js
6. **Renderizado**: La escena 3D se renderiza y se compone sobre el video de fondo
7. **Presentación**: El frame compuesto se muestra al usuario

Este pipeline se ejecuta a 30-60 FPS, garantizando fluidez visual.

3.4.2. Flujo de interacción del usuario

La interacción sigue este esquema:

1. Usuario inicia la aplicación y selecciona género mediante gesto de pointing
2. El catálogo se filtra mostrando solo prendas del género seleccionado
3. Usuario navega entre categorías (superior/inferior) mediante gestos en sidebar
4. Dentro de una categoría, el usuario apunta a zonas laterales para desplazar el carrusel de prendas
5. Al apuntar al centro durante un período, la prenda destacada se selecciona y se aplica automáticamente
6. El proceso se repite para otras categorías, construyendo un outfit completo

3.5. Diseño de la interfaz de usuario

3.5.1. Distribución espacial

La interfaz se divide en zonas funcionales:

- **Zona central**: Visualización del video con prendas superpuestas (ocupa la mayor parte de la pantalla)
- **Sidebar derecho**: Selector de categorías y botón de cambio de género
- **Carrusel inferior**: Galería horizontal de prendas de la categoría activa con disposición circular 3D
- **Panel de debug**: Visualización de landmarks, zonas de gestos y métricas (activable opcionalmente)

3.5.2. Zonas de interacción por gestos

Se definen zonas virtuales de interacción para gestos de pointing:

- **Zona superior derecha**: Cambio de categoría a parte superior del cuerpo
- **Zona inferior derecha**: Cambio de categoría a parte inferior del cuerpo
- **Zona de cambio de género**: Regreso a selección de género (en sidebar)
- **Zona lateral izquierda del carrusel**: Desplazamiento hacia prendas anteriores
- **Zona lateral derecha del carrusel**: Desplazamiento hacia prendas siguientes
- **Zona central del carrusel**: Selección de prenda destacada

Cada zona tiene umbrales de distancia configurables y requiere tiempo de permanencia mínimo para evitar activaciones accidentales.

3.5.3. Feedback visual

El sistema proporciona retroalimentación clara mediante:

- Cambio de color/brillo en elementos señalados
- Escalado suave del elemento destacado en el carrusel
- Indicador de progreso circular al mantener un gesto (para confirmaciones)
- Animaciones de transición entre prendas

3.6. Decisiones técnicas específicas

3.6.1. Gestión de modelos 3D

Los modelos GLTF/GLB se organizan jerárquicamente por:

```
/assets/models/
   male/
      UPPER_BODY/
         T_SHIRT/
         SHIRT/
      LOWER_BODY/
          PANTS/
   female/
       UPPER_BODY/
       LOWER_BODY/
```

Esta estructura permite escalabilidad y organización clara, facilitando la automatización mediante el script generador.

3.6.2. Estrategia de carga de recursos

Para optimizar el rendimiento:

- Carga lazy de modelos 3D (solo se cargan cuando se seleccionan)
- Caché de modelos ya cargados para evitar descargas repetidas
- Precarga de texturas e imágenes de vista previa
- Uso de formatos comprimidos (GLB en lugar de GLTF con assets separados)

3.6.3. Manejo de errores y casos edge

El sistema implementa manejo robusto de situaciones excepcionales:

- Pérdida temporal de detección de pose: Las prendas mantienen última posición conocida
- Rendimiento bajo: Ajuste dinámico de calidad de renderizado

3.7. Consideraciones de rendimiento

3.7.1. Optimizaciones aplicadas

Para garantizar fluidez en el renderizado en tiempo real:

- Throttling del procesamiento de MediaPipe a 30 FPS cuando el dispositivo no soporta 60 FPS
- Reducción de resolución del video de entrada si se detecta bajo rendimiento
- Geometrías simplificadas para prendas (bajo número de polígonos)
- Reutilización de materiales compartidos
- Debounce en detección de gestos para reducir procesamiento innecesario
- RequestAnimationFrame para sincronización con el refresh rate del navegador

3.7.2. Estrategia de filtrado

Capítulo 4

Arquitectura

Este capítulo documenta el proceso de desarrollo del sistema Virtual Fitting Room, describiendo las tecnologías utilizadas, los servicios implementados y los retos técnicos superados durante la construcción del proyecto.

4.1. Tecnologías utilizadas

4.1.1. Frontend y Framework

Angular 20

- Framework principal para desarrollo de la aplicación web
- Standalone components sin necesidad de NgModules
- Control flow nativo (@if, @for) para templates
- Sistema de inyección de dependencias con `providedIn: 'root'`
- Change detection optimizada con `ChangeDetectionStrategy.OnPush`
- Zone management con `NgZone.runOutsideAngular()` para loops de animación

TypeScript 5.4+

- Tipado estático fuerte con strict mode
- Interfaces y tipos personalizados para landmarks y transformaciones
- Enums para categorías, géneros y tipos de prendas
- Type guards para validación en runtime

RxJS 7+

- Observables y Subjects para comunicación reactiva entre servicios
- Operadores: `interval`, `subscribe`, `firstValueFrom`
- Gestión de subscripciones para prevenir memory leaks

4.1.2. Visión por Computador

MediaPipe (Google)

- **Pose Landmarker**: Detección de 33 landmarks corporales en 3D
- **Hand Landmarker**: Detección de 21 puntos por mano
- **Gesture Recognizer**: Reconocimiento de gestos predefinidos
- Procesamiento optimizado mediante WebAssembly y WebGL
- Landmarks normalizados en coordenadas `[0-1]` para X e Y
- Coordenadas Z relativas para profundidad

Configuración aplicada:

```
{
  modelComplexity: 1,  // Balance precisi n/rendimiento
  smoothLandmarks: true,
  minDetectionConfidence: 0.5,
  minTrackingConfidence: 0.5
}
```

4.1.3. Renderizado 3D

Three.js r160+

- Motor de renderizado 3D basado en WebGL
- Cámara perspectiva con FOV 75°
- Iluminación: Ambient Light (intensidad 1.0) + Directional Light (intensidad 0.5)
- Renderer con canal alpha para transparencia
- Antialiasing habilitado para calidad visual
- Soporte de formato GLTF/GLB mediante GLTFLoader

three-stdlib

- GLTFLoader para carga de modelos 3D
- Soporte de animaciones y skeletons en modelos rigged

4.1.4. APIs Web

- **MediaStream API**: Captura de video de cámara a 1280x720px
- **WebGL**: Renderizado 3D acelerado por GPU
- **Canvas 2D API**: Overlay de debug con landmarks y skeleton
- **requestAnimationFrame**: Sincronización de loops de animación
- **HttpClient (Angular)**: Carga de catálogo desde JSON

4.2. Arquitectura de servicios implementada

4.2.1. MediapipeService

Servicio responsable de la integración con MediaPipe.
Funcionalidades principales:

- Inicialización de PoseLandmarker, HandLandmarker y GestureRecognizer
- Detección de pose con método `detectPose(video, timestamp)`
- Emisión de landmarks 2D (`poseLandmarks$`) y 3D (`poseWorldLandmarks$`)
- Procesamiento frame-by-frame con timestamps precisos

4.2.2. GestureDetectorService

Sistema personalizado de reconocimiento de gestos.
Gestos implementados:

- **Pointing**: Índice extendido, otros dedos cerrados
- **Swipe Left/Right**: Movimiento lateral de la mano

Lógica de detección:

1. Análisis de landmarks de mano (21 puntos)
2. Cálculo de distancias entre puntos clave
3. Verificación de configuración de dedos
4. Emisión de eventos mediante `gestureDetected$`
5. Estado de gesto en tiempo real mediante `gestureState$`

4.2.3. ThreejsService

Gestor de la escena 3D.
Inicialización:

```
initScene(canvas: HTMLCanvasElement, transparent: boolean) {
  this.scene = new THREE.Scene();
  this.camera = new THREE.PerspectiveCamera(75, ...);
  this.camera.position.z = 5;

  this.renderer = new THREE.WebGLRenderer({
    canvas: canvas,
    alpha: transparent,
    antialias: true
  });

  // Iluminaci n
  const ambientLight = new THREE.AmbientLight(0xffffff, 1);
```

```
14    const directionalLight =
15      new THREE.DirectionalLight(0xffffff, 0.5);
16    directionalLight.position.set(5, 5, 5);
17
18    this.scene.add(ambientLight, directionalLight);
19  }
```

4.2.4. GarmentManagerService

Gestor centralizado de prendas cargadas.
Responsabilidades:

- Carga asíncrona de modelos GLB mediante ModelLoaderService
- Caché de modelos ya cargados en memoria
- Añadir/remover prendas de la escena Three.js
- Actualización de transformaciones en cada frame
- Coordinación con SkeletonRetargetService para animación de huesos

4.2.5. SkeletonRetargetService

Servicio avanzado de retargeting de skeleton (característica destacada del proyecto).
Funcionalidad: Este servicio no solo posiciona las prendas, sino que anima los huesos (bones) de los modelos 3D rigged para que se deformen según los movimientos del usuario.
Proceso de retargeting:

1. **Búsqueda de huesos**: Localiza huesos en el skeleton del modelo mediante patrones de nombres:
   - Superiores: LeftShoulder, LeftElbow, RightShoulder, RightElbow
   - Inferiores: LeftHip, LeftKnee, RightHip, RightKnee
2. **Caché de bind poses**: Almacena quaternions iniciales de cada hueso
3. **Cálculo de rotaciones**:
   - Convierte landmarks de MediaPipe a coordenadas del rig
   - Calcula vectores de dirección de extremidades del usuario
   - Obtiene vectores actuales de huesos del modelo
   - Computa quaternion de rotación entre vectores
   - Aplica suavizado mediante `slerp` (factor 0.3)
4. **Actualización por extremidad**: Anima shoulder→elbow y hip→knee independientemente

Código simplificado:

```
private animateLimb(
  upperBone, lowerBone,
  upperLM, midLM, lowerLM
): void {
  // Convertir landmarks a sistema de coordenadas del rig
  const U = this.toRig(upperLM);
  const M = this.toRig(midLM);

  // Direcci n objetivo del usuario
  const targetDir = M.sub(U).normalize();

  // Direcci n actual del hueso
  const currentDir = /* posici n mundial del hueso */;

  // Quaternion de rotaci n
  const rotation = new THREE.Quaternion()
    .setFromUnitVectors(currentDir, targetDir);

  // Suavizado
  upperBone.quaternion.slerp(rotation, this.smoothing);

  // Actualizar skeleton
  skeleton.update();
}
```

4.2.6. GarmentTransformService

Servicio de transformación de posición, rotación y escala de prendas.
Estrategias por categoría:
UPPER_BODY:

- Anclaje: Hombros (landmarks 11, 12)
- Escala: Distancia entre hombros × 2.0
- Rotación: Basada en vector hombro-cadera

LOWER_BODY:

- Anclaje: Caderas (landmarks 23, 24)
- Escala: Distancia entre caderas × 1.8
- Rotación: Basada en vector cadera-rodilla

Suavizado de transformaciones:

```
private readonly SMOOTHING = 0.15;

root.position.lerp(targetPosition, this.SMOOTHING);
root.scale.lerp(targetScale, this.SMOOTHING);
root.rotation.x = THREE.MathUtils.lerp(
  root.rotation.x, targetRotation.x, this.SMOOTHING
);
```

4.2.7. CoordinateTransformerService

Servicio de conversión de coordenadas entre sistemas.
Transformación MediaPipe → Three.js:

```
worldToThreeJS(landmark: Landmark3D): THREE.Vector3 {
  return new THREE.Vector3(
    -landmark.x * 2.5,  // Invertir X y escalar
    -landmark.y * 2.5,  // Invertir Y y escalar
    -landmark.z * 2.5   // Invertir Z y escalar
  );
}
```

Cálculos geométricos:

- `calculateCentroid()`: Punto medio de múltiples landmarks
- `calculateDistance()`: Distancia euclidiana 3D
- `calculateOrientation()`: Euler angles basados en vectores de hombro y columna

4.2.8. DebugLoggerService

Sistema de logging detallado para desarrollo.
Funcionalidades:

- Logging cada N frames (configurable, por defecto 30)
- Comparación de landmarks MediaPipe vs posiciones Three.js
- Información de transformaciones (posición, escala, rotación)
- Distancias y ratios de escala
- Agrupación jerárquica en consola del navegador

4.2.9. GarmentCatalogService

Gestor del catálogo de prendas.
Inicialización:

1. Carga `/assets/models-list.json` mediante HttpClient
2. Parsea estructura jerárquica: `gender → category → type → models`
3. Genera objetos Garment con metadatos completos
4. Elimina duplicados por ID
5. Formatea nombres para visualización

Métodos de filtrado:

- `getGarmentsByCategory(category, gender?)`: Filtro por categoría y opcionalmente género
- `getGarmentsByCategoryAndGender()`: Filtro combinado incluyendo UNISEX
- `getGarmentById(id)`: Búsqueda por identificador único

4.3. Componentes de interfaz

4.3.1. AppComponent

Componente raíz que orquesta la aplicación.
Gestión de estado:

- `selectedGender`: Género seleccionado (null si no se ha elegido)
- `selectedCategory`: Categoría activa (por defecto UPPER_BODY)
- `selectedGarments`: Map categoría → prenda seleccionada

Detección de pointing: Implementa un sistema sofisticado de detección de pointing con zonas expandidas (50px de margen) y tiempo de permanencia (1200ms) antes de activación.

```
private checkPointingGesture(): void {
  if (!this.currentGestureState.isPointing) {
    this.resetPointingStateWithGracePeriod();
    return;
  }

  const handPos = this.currentGestureState.handPosition;
  const screenPos = this.normalizeHandPosition(handPos);

  // Detectar género/categoría/botón en posición
  const pointedGender = this.detectGenderAtPosition(...);
  const pointedCategory = this.detectCategoryAtPosition(...);

  // Sistema de progreso con temporizador
  if (pointedCategory !== null) {
    if (this.pointingStartTime === 0) {
      this.pointingStartTime = Date.now();
    }

    const elapsed = Date.now() - this.pointingStartTime;
    this.pointingProgress = Math.min(
      elapsed / this.POINTING_SELECTION_DELAY, 1
    );

    if (this.pointingProgress >= 1) {
      this.onCategorySelected(pointedCategory);
      this.forceResetPointingState();
    }
  }
}
```

4.3.2. CameraFeedComponent

Componente de captura y visualización de video con overlay de debug.
Características:

- Solicitud de permisos de cámara (1280x720, facing: user)
- Loop de procesamiento con `requestAnimationFrame`
- Detección de pose y gestos en cada frame
- Modo debug activable con tecla 'L'
- Overlay de canvas con skeleton y landmarks
- Ejecución fuera de Angular zone para optimización

Pipeline de procesamiento:

```
private processFrame = (): void => {
  const video = this.videoElement.nativeElement;
  if (video.readyState === video.HAVE_ENOUGH_DATA) {
    const ts = performance.now();

    // Detecci n de pose
    const pose = this.mediapipeService.detectPose(video, ts);

    // Detecci n de manos y gestos
    const handResults = this.mediapipeService
      .handLandmarker?.detectForVideo(video, ts);
    const gestureResults = this.mediapipeService
      .gestureRecognizer?.recognizeForVideo(video, ts);

    // Procesamiento de gestos
    if (handsLandmarks.length > 0) {
      this.gestureDetector.detectGesture(
        handsLandmarks, gestures
      );
    }

    // Dibujar debug si est  activo
    if (this.debugMode) {
      this.drawOverlay(handsLandmarks, pose.poseLandmarks);
    }
  }

  this.animationId = requestAnimationFrame(this.processFrame);
};
```

4.3.3. SceneViewerComponent

Componente de visualización 3D de prendas sobre el usuario.
Ciclo de actualización:

1. Suscripción a `poseWorldLandmarks$` y `poseLandmarks$`
2. Almacenamiento de últimos landmarks en cache

3. Loop de animación independiente con `requestAnimationFrame`
4. Verificación de pose válida (33 landmarks)
5. Si hay pose: actualizar prendas mediante GarmentManager
6. Si no hay pose: ocultar todas las prendas
7. Renderizado de escena Three.js

Gestión de visibilidad:

```
private startAnimationLoop(): void {
  const animate = (): void => {
    const hasPose = !!(
      this.latestWorld?.length >= 33 &&
      this.latestPose2d?.length >= 33
    );

    if (hasPose !== this.lastPoseState) {
      if (!hasPose) {
        this.forceHideAllGarments();
      }
      this.lastPoseState = hasPose;
    }

    if (hasPose) {
      this.garmentManager.updateGarments(
        this.latestPose2d!,
        this.latestWorld!
      );
      this.ensureGarmentsVisible();
    }

    this.threeService.renderer.render(
      this.threeService.scene,
      this.threeService.camera
    );

    this.animationId = requestAnimationFrame(animate);
  };
  animate();
}
```

4.3.4. GenderSelectorComponent

Pantalla inicial de selección de género.
Funcionalidades:

- Tres opciones: Masculino, Femenino, Unisex
- Iconos emoji para representación visual

- Indicador de pointing con barra de progreso
- Emisión de evento `genderSelected` al confirmar

4.3.5. Otros componentes

CategorySidebarComponent:

- Navegación vertical entre categorías de prendas
- Botón para regresar a selección de género
- Highlighting de categoría activa
- Detección de pointing sobre botones

GalleryBarComponent:

- Carrusel horizontal de prendas de la categoría activa
- Navegación mediante gestos o swipes
- Elemento central destacado
- Opción "Sin prendaçomo primer elemento

4.4. Flujo de datos completo

4.4.1. Inicialización

1. Usuario accede a la aplicación
2. `AppComponent.ngOnInit()`: Inicializa `GarmentCatalogService`
3. Se muestra `GenderSelectorComponent`
4. Paralelamente, `CameraFeedComponent` solicita permisos de cámara
5. `MediapipeService` inicializa modelos de ML
6. Usuario selecciona género mediante pointing gesture
7. Catálogo se filtra por género seleccionado
8. Se muestra interfaz principal con cámara, sidebar y galería

4.4.2. Loop principal (30-60 FPS)

1. **CameraFeedComponent.processFrame()**:
   - Captura frame de video
   - Envía a MediaPipe para detección
   - Obtiene 33 landmarks de pose en 2D y 3D
   - Obtiene landmarks de manos
   - Detecta gestos mediante GestureDetector
2. **MediapipeService**:
   - Emite `poseLandmarks$` (coordenadas 2D normalizadas)
   - Emite `poseWorldLandmarks$` (coordenadas 3D en metros)
3. **SceneViewerComponent.animate()**:
   - Lee últimos landmarks de cache
   - Verifica validez de detección (33 puntos)
   - Llama a `GarmentManager.updateGarments()`
4. **GarmentManager.updateGarments()**:
   - Itera sobre prendas cargadas
   - Para cada prenda:
     - Llama a `GarmentTransformService.updateTransform()`
     - Llama a `SkeletonRetargetService.updateSkeleton()`
5. **GarmentTransformService**:
   - Calcula centroide de landmarks de anclaje
   - Calcula escala basada en distancias corporales
   - Calcula rotación según orientación del torso
   - Aplica suavizado con `lerp/slerp`
   - Actualiza `position`, `scale`, `rotation` del root
6. **SkeletonRetargetService**:
   - Convierte landmarks a coordenadas del rig
   - Calcula quaternions de rotación para cada hueso
   - Aplica suavizado con `slerp`
   - Actualiza skeleton del modelo 3D
7. **ThreeService**:
   - Renderiza escena actualizada
   - Compone sobre canvas transparente

4.4.3. Selección de prenda

1. Usuario apunta a zona lateral de GalleryBar
2. `GestureDetectorService` detecta pointing
3. `AppComponent.checkPointingGesture()` verifica posición
4. Si apunta a zona de navegación:
   - Inicia temporizador de pointing
   - Actualiza barra de progreso visual
   - Al completar 1200ms, desplaza galería
5. Usuario apunta a prenda en centro del carrusel
6. Al mantener pointing 1200ms sobre prenda:
   - `AppComponent.onGarmentSelected()` se ejecuta
   - Actualiza `selectedGarments` Map
   - `SceneViewerComponent` detecta cambio via `ngOnChanges`
   - `updateVisibleGarments()` se ejecuta
   - `GarmentManager.loadGarment()` carga modelo si no está en caché
   - Prenda anterior se remueve de escena
   - Nueva prenda se añade y se posiciona inmediatamente

4.5. Optimizaciones implementadas

4.5.1. Rendimiento

- **Zone management**: Loops de animación fuera de Angular zone con `NgZone.runOutsideAngular`
- **Change detection**: `OnPush` strategy en SceneViewerComponent
- **Caché de modelos**: Modelos GLB se cargan una vez y se reutilizan
- **Throttling de debug**: Logs cada 30 frames en lugar de cada frame
- **Smoothing**: Interpolación lineal/esférica reduce cálculos de transformación
- **Lazy evaluation**: Prendas solo se procesan si hay pose válida

4.5.2. Experiencia de usuario

- **Zonas expandidas**: 50px de margen en áreas de interacción para facilitar pointing
- **Periodo de gracia**: 400ms de tolerancia antes de resetear pointing
- **Retroalimentación visual**: Barras de progreso para acciones con pointing
- **Suavizado de movimiento**: Factor 0.15-0.3 para transiciones fluidas
- **Ocultación inteligente**: Prendas se ocultan automáticamente si se pierde detección

4.6. Desafíos técnicos superados

4.6.1. Sistema de coordenadas

Problema: MediaPipe y Three.js usan sistemas de coordenadas diferentes.

Solución: Implementación de `CoordinateTransformerService` que invierte ejes y aplica factor de escala (2.5):

```
worldToThreeJS(landmark): Vector3 {
  return new Vector3(
    -landmark.x * 2.5,
    -landmark.y * 2.5,
    -landmark.z * 2.5
  );
}
```

4.6.2. Skeleton retargeting

Problema: Nombres de huesos inconsistentes entre modelos 3D de diferentes fuentes.

Solución: Búsqueda flexible con múltiples patrones:

```
findBone(skeleton, patterns: string[]): Bone | undefined {
  // B squeda exacta primero
  for (const pattern of patterns) {
    const bone = skeleton.bones.find(b => b.name === pattern);
    if (bone) return bone;
  }

  // B squeda case-insensitive sin espacios/guiones
  for (const pattern of patterns) {
    const bone = skeleton.bones.find(b =>
      b.name.toLowerCase().replace(/[\s_-]/g, '') ===
      pattern.toLowerCase().replace(/[\s_-]/g, '')
    );
    if (bone) return bone;
  }

  return undefined;
}
```

Incluye soporte para typos conocidos como "LightKnee.[en] lugar de "LeftKnee".

4.6.3. Sincronización de loops

Problema: Dos loops de animación independientes (detección y renderizado) pueden desincronizarse.

Solución: Sistema de cache de landmarks con lectura thread-safe:

```
private latestWorld: any[] | null = null;
private latestPose2d: any[] | null = null;

// Loop de detecci n actualiza cache
```

```
this.poseSub = this.mediapipe.poseLandmarks$.subscribe(
  (pose2d) => { this.latestPose2d = pose2d; }
);

// Loop de renderizado lee de cache
const hasPose = !!(
  this.latestWorld?.length >= 33 &&
  this.latestPose2d?.length >= 33
);
```

4.6.4. Gestión de visibilidad

Problema: Prendas quedaban visibles cuando se perdía detección de pose.
Solución: Sistema de estado de pose con ocultación forzada:

```
if (hasPose !== this.lastPoseState) {
  if (!hasPose) {
    this.forceHideAllGarments();  // Ocultar TODO
  }
  this.lastPoseState = hasPose;
}
```

4.7. Herramientas de desarrollo

- **IDE**: Visual Studio Code / IntelliJ IDEA Ultimate
- **CLI**: Angular CLI 20 para scaffolding y build
- **Package manager**: npm
- **Version control**: Git + GitHub
- **Debugging**: Chrome DevTools + DebugLoggerService personalizado
- **Testing**: Navegador web (Chrome/Firefox) con cámara
- **3D modeling**: Blender para ajuste de modelos GLB

Capítulo 5

Modelo de Datos

El modelo de datos de Virtual Fitting Room es minimalista, ya que la aplicación no requiere persistencia en base de datos ni gestión compleja de información. Los datos se estructuran mediante interfaces TypeScript y archivos estáticos de modelos 3D.

5.1. Estructuras de datos principales

5.1.1. Interfaz Garment

Define la estructura de una prenda en el catálogo:

```
interface Garment {
  id: string;                // Identificador  nico
  name: string;              // Nombre descriptivo
  category: GarmentCategory; // Categor a
  subcategory: string;       // Subcategor a espec fica
  gender: GarmentGender;     // G nero
  modelPath: string;         // Ruta al archivo GLB
  imagePath: string;         // Ruta a imagen preview
}
```

5.1.2. Enumeraciones

GarmentCategory:

```
enum GarmentCategory {
  UPPER_BODY = 'UPPER_BODY',
  LOWER_BODY = 'LOWER_BODY'
}
```

GarmentGender:

```
enum GarmentGender {
  MALE = 'male',
  FEMALE = 'female'
}
```

5.1.3. Interfaz PoseLandmark

Representa un landmark detectado por MediaPipe:

```
interface PoseLandmark {
  x: number;           // Coordenada X normalizada [0-1]
  y: number;           // Coordenada Y normalizada [0-1]
  z: number;           // Profundidad relativa
  visibility: number; // Confianza de detecci n [0-1]
}
```

5.1.4. Interfaz Transform

Define transformaciones 3D para posicionar prendas:

```
interface Transform {
  position: { x: number; y: number; z: number };
  rotation: { x: number; y: number; z: number };
  scale: { x: number; y: number; z: number };
}
```

5.2. Organización de archivos de modelos 3D

Los modelos se organizan jerárquicamente en el sistema de archivos:

```
/assets/models/
 male/
    UPPER_BODY/
       T_SHIRT/
          casual-tshirt.glb
          formal-shirt.glb
       SHIRT/
           button-shirt.glb
    LOWER_BODY/
        PANTS/
            jeans.glb
            chinos.glb
 female/
     UPPER_BODY/
        T_SHIRT/
     LOWER_BODY/
         PANTS/
```

Esta estructura permite:

- Escalabilidad: Añadir nuevas prendas sin modificar código
- Organización clara por género, categoría y subcategoría
- Generación automática del catálogo mediante script

5.3. Generación automática del catálogo

Un script Node.js (`generate-models-list.mjs`) escanea la estructura de carpetas y genera código TypeScript:

```
// Script simplificado
import { readdir } from 'fs/promises';

async function generateCatalog() {
  const genders = await readdir('./assets/models/');

  for (const gender of genders) {
    const categories = await readdir(
      `./assets/models/${gender}`
    );

    for (const category of categories) {
      const subcategories = await readdir(
        `./assets/models/${gender}/${category}`
      );

      for (const subcat of subcategories) {
        const files = await readdir(
          `./assets/models/${gender}/${category}/${subcat}`
        );

        // Generar objetos Garment
      }
    }
  }
}
```

Este enfoque elimina la necesidad de registrar manualmente cada prenda en el código.

5.4. Formato de modelos 3D

5.4.1. Especificación GLB

Los modelos utilizan formato GLB (GL Transmission Format Binary):

- Formato binario compacto que incluye geometría, texturas y materiales
- Estándar de la industria soportado nativamente por Three.js
- Tamaño optimizado para carga web (típicamente <5MB por prenda)
- Soporte para PBR (Physically Based Rendering) materials

5.4.2. Requisitos de los modelos

Para funcionar correctamente en el sistema, los modelos deben cumplir:

- Pivot point centrado en la parte superior de la prenda

- Escala normalizada (aproximadamente 1 unidad = altura real)
- Geometría optimizada (<10,000 polígonos por prenda)
- Texturas comprimidas (formato JPEG para difuso)
- Orientación inicial: frente hacia -Z axis

5.5. Estado de la aplicación

El estado runtime se gestiona en memoria mediante StateManagementService:

```
interface AppState {
  selectedGender: GarmentGender | null;
  activeCategory: GarmentCategory;
  selectedGarments: {
    UPPER_BODY: Garment | null;
    LOWER_BODY: Garment | null;
  };
}
```

Este estado no se persiste entre sesiones, reiniciándose al recargar la aplicación.

Capítulo 6

Desarrollo

Este capítulo documenta el proceso de desarrollo del sistema Virtual Fitting Room, describiendo las tecnologías utilizadas, los servicios implementados y los retos técnicos superados durante la construcción del proyecto.

6.1. Tecnologías utilizadas

6.1.1. Frontend y Framework

Angular 20

- Framework principal para desarrollo de la aplicación web
- Standalone components sin necesidad de NgModules
- Control flow nativo (@if, @for) para templates
- Sistema de inyección de dependencias con `providedIn: 'root'`
- Change detection optimizada con `ChangeDetectionStrategy.OnPush`
- Zone management con `NgZone.runOutsideAngular()` para loops de animación

TypeScript 5.4+

- Tipado estático fuerte con strict mode
- Interfaces y tipos personalizados para landmarks y transformaciones
- Enums para categorías, géneros y tipos de prendas
- Type guards para validación en runtime

RxJS 7+

- Observables y Subjects para comunicación reactiva entre servicios
- Operadores: `interval`, `subscribe`, `firstValueFrom`
- Gestión de subscripciones para prevenir memory leaks

6.1.2. Visión por Computador

MediaPipe (Google)

- **Pose Landmarker**: Detección de 33 landmarks corporales en 3D
- **Hand Landmarker**: Detección de 21 puntos por mano
- **Gesture Recognizer**: Reconocimiento de gestos predefinidos
- Procesamiento optimizado mediante WebAssembly y WebGL
- Landmarks normalizados en coordenadas `[0-1]` para X e Y
- Coordenadas Z relativas para profundidad

Configuración aplicada:

```
{
  modelComplexity: 1,  // Balance precisi n/rendimiento
  smoothLandmarks: true,
  minDetectionConfidence: 0.5,
  minTrackingConfidence: 0.5
}
```

6.1.3. Renderizado 3D

Three.js r160+

- Motor de renderizado 3D basado en WebGL
- Cámara perspectiva con FOV 75°
- Iluminación: Ambient Light (intensidad 1.0) + Directional Light (intensidad 0.5)
- Renderer con canal alpha para transparencia
- Antialiasing habilitado para calidad visual
- Soporte de formato GLTF/GLB mediante GLTFLoader

three-stdlib

- GLTFLoader para carga de modelos 3D
- Soporte de animaciones y skeletons en modelos rigged

6.1.4. Creación de modelos 3D

CLO3D

Software profesional de diseño de moda 3D utilizado para la creación de las prendas virtuales.

- Simulación física realista de telas
- Patrones 2D que se ensamblan en modelos 3D
- Texturas y materiales PBR (Physically Based Rendering)
- Exportación directa a formato FBX/OBJ
- Ajuste preciso de dimensiones y proporciones

CLO3D permitió crear prendas con geometría optimizada y proporciones realistas, fundamentales para el correcto funcionamiento del sistema de escalado dinámico.

Blender

Software open-source de modelado 3D utilizado para rigging y preparación final de modelos.

- **Rigging**: Creación de armaduras (skeletons) para prendas
- Asignación de bones: LeftShoulder, RightShoulder, LeftElbow, RightElbow (prendas superiores)
- Asignación de bones: LeftHip, RightHip, LeftKnee, RightKnee (prendas inferiores)
- **Weight painting**: Definición de influencia de cada bone sobre vértices
- Optimización de geometría: Reducción de polígonos manteniendo calidad visual
- Exportación a formato GLTF/GLB compatible con Three.js

El rigging en Blender fue esencial para implementar el sistema de skeleton retargeting, permitiendo que las prendas se deformen naturalmente según los movimientos del usuario.

Pipeline de creación de prendas:

1. **Diseño en CLO3D**: Creación de patrones 2D y simulación 3D
2. **Exportación**: Formato FBX desde CLO3D
3. **Importación en Blender**: Limpieza de geometría
4. **Rigging**: Creación de armadura con nombres estandarizados
5. **Weight painting**: Asignación de pesos de influencia
6. **Optimización**: Reducción de polígonos, compresión de texturas
7. **Exportación final**: Formato GLB con texturas embebidas
8. **Validación**: Prueba en el sistema para verificar skeleton

6.1.5. APIs Web

- **MediaStream API**: Captura de video de cámara a 1280x720px
- **WebGL**: Renderizado 3D acelerado por GPU
- **Canvas 2D API**: Overlay de debug con landmarks y skeleton
- **requestAnimationFrame**: Sincronización de loops de animación
- **HttpClient (Angular)**: Carga de catálogo desde JSON

6.2. Arquitectura de servicios implementada

6.2.1. MediapipeService

Servicio responsable de la integración con MediaPipe.
Funcionalidades principales:

- Inicialización de PoseLandmarker, HandLandmarker y GestureRecognizer
- Detección de pose con método `detectPose(video, timestamp)`
- Emisión de landmarks 2D (`poseLandmarks$`) y 3D (`poseWorldLandmarks$`)
- Procesamiento frame-by-frame con timestamps precisos

6.2.2. GestureDetectorService

Sistema personalizado de reconocimiento de gestos.
Gestos implementados:

- **Pointing**: Índice extendido, otros dedos cerrados
- **Swipe Left/Right**: Movimiento lateral de la mano

Lógica de detección:

1. Análisis de landmarks de mano (21 puntos)
2. Cálculo de distancias entre puntos clave
3. Verificación de configuración de dedos
4. Emisión de eventos mediante `gestureDetected$`
5. Estado de gesto en tiempo real mediante `gestureState$`

6.2.3. ThreejsService

Gestor de la escena 3D.
Inicialización:

```
initScene(canvas: HTMLCanvasElement, transparent: boolean) {
  this.scene = new THREE.Scene();
  this.camera = new THREE.PerspectiveCamera(75, ...);
  this.camera.position.z = 5;

  this.renderer = new THREE.WebGLRenderer({
    canvas: canvas,
    alpha: transparent,
    antialias: true
  });

  // Iluminaci n
  const ambientLight = new THREE.AmbientLight(0xffffff, 1);
  const directionalLight =
    new THREE.DirectionalLight(0xffffff, 0.5);
  directionalLight.position.set(5, 5, 5);

  this.scene.add(ambientLight, directionalLight);
}
```

6.2.4. GarmentManagerService

Gestor centralizado de prendas cargadas.
Responsabilidades:

- Carga asíncrona de modelos GLB mediante ModelLoaderService
- Caché de modelos ya cargados en memoria
- Añadir/remover prendas de la escena Three.js
- Actualización de transformaciones en cada frame
- Coordinación con SkeletonRetargetService para animación de huesos

6.2.5. SkeletonRetargetService

Servicio avanzado de retargeting de skeleton (característica destacada del proyecto).
Funcionalidad: Este servicio no solo posiciona las prendas, sino que anima los huesos (bones) de los modelos 3D rigged para que se deformen según los movimientos del usuario.
Proceso de retargeting:

1. **Búsqueda de huesos**: Localiza huesos en el skeleton del modelo mediante patrones de nombres:
   - Superiores: LeftShoulder, LeftElbow, RightShoulder, RightElbow
   - Inferiores: LeftHip, LeftKnee, RightHip, RightKnee

2. **Caché de bind poses**: Almacena quaternions iniciales de cada hueso
3. **Cálculo de rotaciones**:
   - Convierte landmarks de MediaPipe a coordenadas del rig
   - Calcula vectores de dirección de extremidades del usuario
   - Obtiene vectores actuales de huesos del modelo
   - Computa quaternion de rotación entre vectores
   - Aplica suavizado mediante `slerp` (factor 0.3)
4. **Actualización por extremidad**: Anima shoulder→elbow y hip→knee independientemente

Código simplificado:

```
private animateLimb(
  upperBone, lowerBone,
  upperLM, midLM, lowerLM
): void {
  // Convertir landmarks a sistema de coordenadas del rig
  const U = this.toRig(upperLM);
  const M = this.toRig(midLM);

  // Direcci n objetivo del usuario
  const targetDir = M.sub(U).normalize();

  // Direcci n actual del hueso
  const currentDir = /* posici n mundial del hueso */;

  // Quaternion de rotaci n
  const rotation = new THREE.Quaternion()
    .setFromUnitVectors(currentDir, targetDir);

  // Suavizado
  upperBone.quaternion.slerp(rotation, this.smoothing);

  // Actualizar skeleton
  skeleton.update();
}
```

6.2.6. GarmentTransformService

Servicio de transformación de posición, rotación y escala de prendas.
Estrategias por categoría:
UPPER_BODY:

- Anclaje: Hombros (landmarks 11, 12)
- Escala: Distancia entre hombros × 2.0
- Rotación: Basada en vector hombro-cadera

LOWER_BODY:

- Anclaje: Caderas (landmarks 23, 24)
- Escala: Distancia entre caderas × 1.8
- Rotación: Basada en vector cadera-rodilla

Suavizado de transformaciones:

```
private readonly SMOOTHING = 0.15;

root.position.lerp(targetPosition, this.SMOOTHING);
root.scale.lerp(targetScale, this.SMOOTHING);
root.rotation.x = THREE.MathUtils.lerp(
  root.rotation.x, targetRotation.x, this.SMOOTHING
);
```

6.2.7. Matemáticas del skeleton retargeting: Quaternions

El sistema de animación de huesos utiliza quaternions en lugar de ángulos de Euler por razones fundamentales de estabilidad y rendimiento.

¿Por qué quaternions?

Problemas de los ángulos de Euler:

- **Gimbal lock**: Pérdida de un grado de libertad cuando dos ejes de rotación se alinean
- **Interpolación no lineal**: La interpolación directa de ángulos no produce rotaciones suaves
- **Ambigüedad**: Múltiples representaciones para la misma orientación
- **Orden de aplicación**: Rotar en orden XYZ vs ZYX produce resultados diferentes

Ventajas de los quaternions:

- Representación única y no ambigua de rotaciones 3D
- Interpolación esférica (slerp) produce transiciones suaves
- Inmunes a gimbal lock
- Composición eficiente mediante multiplicación
- Menor coste computacional que matrices 4x4

Representación de quaternions

Un quaternion se representa como:

$$q = w + xi + yj + zk$$

Donde w es la parte escalar y (x, y, z) es la parte vectorial. Para rotaciones, se usan quaternions unitarios ($|q| = 1$).

En Three.js, los quaternions se definen como:

```
class Quaternion {
  x: number;  // Componente i
  y: number;  // Componente j
  z: number;  // Componente k
  w: number;  // Parte escalar
}
```

Algoritmo de retargeting

El proceso de mapeo de movimientos del usuario a huesos de prendas sigue estos pasos:

1. Obtención de vectores direccionales

Para una extremidad del usuario (ej: hombro → codo):

```
// Landmarks del usuario en coordenadas de MediaPipe
const shoulderLM = worldLandmarks[11]; // Hombro izquierdo
const elbowLM = worldLandmarks[13];    // Codo izquierdo

// Convertir a sistema del rig (invertir ejes, espejado)
const shoulderPos = this.toRig(shoulderLM);
const elbowPos = this.toRig(elbowLM);

// Vector direcci n del usuario
const userDirection = new THREE.Vector3()
  .subVectors(elbowPos, shoulderPos)
  .normalize();
```

2. Obtención del vector del hueso actual

```
// Posici n mundial del hueso shoulder
const shoulderBone = /* bone del modelo 3D */;
shoulderBone.updateWorldMatrix(false, false);

const shoulderWorldPos = new THREE.Vector3();
shoulderBone.getWorldPosition(shoulderWorldPos);

// Posici n mundial del siguiente hueso (elbow)
const elbowBone = /* siguiente bone en jerarqu a */;
const elbowWorldPos = new THREE.Vector3();
elbowBone.getWorldPosition(elbowWorldPos);

// Direcci n actual del hueso
const boneDirection = new THREE.Vector3()
  .subVectors(elbowWorldPos, shoulderWorldPos)
  .normalize();
```

3. Cálculo del quaternion de rotación

Se calcula el quaternion que rota desde la dirección actual del hueso hacia la dirección objetivo del usuario:

```
// Quaternion que rota boneDirection hacia userDirection
const rotationQuat = new THREE.Quaternion()
  .setFromUnitVectors(boneDirection, userDirection);
```

Internamente, `setFromUnitVectors` implementa:

$$q = \{ \ quaternionidentidad si \vec{v_1} = \vec{v_2} quaternionde 180 si \vec{v_1} = -\vec{v_2} quaterniondeleje \vec{v_1} \times \vec{v_2} enotrocaso$$

4. Transformación a espacio local

El quaternion calculado está en espacio mundial, pero los huesos se animan en espacio local relativo a su padre:

```
// Quaternion mundial actual del hueso
const boneWorldQuat = shoulderBone.getWorldQuaternion(
  new THREE.Quaternion()
);

// Aplicar rotaci n en espacio mundial
const targetWorldQuat = rotationQuat.multiply(boneWorldQuat);

// Quaternion del padre en espacio mundial
const parentWorldQuat = new THREE.Quaternion();
if (shoulderBone.parent) {
  shoulderBone.parent.getWorldQuaternion(parentWorldQuat);
}

// Convertir a espacio local
const targetLocalQuat = parentWorldQuat
  .clone()
  .invert()
  .multiply(targetWorldQuat);
```

5. Interpolación esférica (SLERP)

Para evitar cambios bruscos, se interpola suavemente desde la rotación actual hacia la objetivo:

```
// Factor de suavizado (0.3 = 30% hacia objetivo cada frame)
const smoothing = 0.3;

// SLERP: Interpolaci n esf rica
shoulderBone.quaternion.slerp(targetLocalQuat, smoothing);
```

SLERP garantiza interpolación en la esfera unitaria de quaternions, produciendo la rotación más corta y suave posible.

Matemáticamente, SLERP entre quaternions q_1 y q_2 con parámetro $t \in [0, 1]$ es:

$$slerp(q_1, q_2, t) = \frac{\sin((1-t)\theta)}{\sin\theta} q_1 + \frac{\sin(t\theta)}{\sin\theta} q_2$$

Donde $\theta = \arccos(q_1 \cdot q_2)$ es el ángulo entre quaternions.

Actualización del skeleton

Después de actualizar todos los quaternions de los huesos:

```
// Forzar recalculo de matrices de transformaci n
skeleton.update();

// Three.js recalcula:
// - Matrices locales de cada hueso
// - Matrices mundiales mediante multiplicaci n jer rquica
// - Aplicaci n de skinning weights a v rtices
```

Consideraciones de rendimiento

Optimizaciones implementadas:

- **Caché de bind poses**: Los quaternions iniciales se almacenan una vez y se reutilizan
- **Actualización selectiva**: Solo se procesan huesos de categoría activa (upper/lower)
- **Normalización perezosa**: Se asume que vectores de MediaPipe ya están normalizados
- **Evitar allocaciones**: Se reutilizan objetos Vector3/Quaternion en lugar de crear nuevos

Coste computacional por frame:

- Conversión de landmarks: O(n) donde n = número de landmarks usados (6 por prenda)
- Cálculo de quaternions: O(k) donde k = número de huesos animados (4 por prenda)
- SLERP: Operación constante O(1) por hueso
- Actualización de skeleton: O(h) donde h = total de huesos en jerarquía

Para 2 prendas con 4 huesos cada una a 60 FPS:

$$Operaciones/segundo = 60 \times (12 + 8 + 8 + 16) \approx 2640$$

Esto es manejable en navegadores modernos sin impactar el rendimiento.

Debugging de rotaciones

El DebugLoggerService registra información de quaternions para verificación:

```
console.log('Rotation␣(quaternion):', {
  x: bone.quaternion.x.toFixed(4),
  y: bone.quaternion.y.toFixed(4),
  z: bone.quaternion.z.toFixed(4),
  w: bone.quaternion.w.toFixed(4)
});

```

```
// Conversi n a Euler para interpretaci n humana
const euler = new THREE.Euler().setFromQuaternion(
  bone.quaternion
);
console.log('Rotation␣(degrees):', {
  x: THREE.MathUtils.radToDeg(euler.x).toFixed(2),
  y: THREE.MathUtils.radToDeg(euler.y).toFixed(2),
  z: THREE.MathUtils.radToDeg(euler.z).toFixed(2)
});
```

Esto permite detectar rotaciones anómalas (ej: superiores a 180° que indican posible inversión de vectores).

6.2.8. CoordinateTransformerService

Servicio de conversión de coordenadas entre sistemas.
Transformación MediaPipe → Three.js:

```
worldToThreeJS(landmark: Landmark3D): THREE.Vector3 {
  return new THREE.Vector3(
    -landmark.x * 2.5,  // Invertir X y escalar
    -landmark.y * 2.5,  // Invertir Y y escalar
    -landmark.z * 2.5   // Invertir Z y escalar
  );
}
```

Cálculos geométricos:

- `calculateCentroid()`: Punto medio de múltiples landmarks
- `calculateDistance()`: Distancia euclidiana 3D
- `calculateOrientation()`: Euler angles basados en vectores de hombro y columna

6.2.9. DebugLoggerService

Sistema de logging detallado para desarrollo.
Funcionalidades:

- Logging cada N frames (configurable, por defecto 30)
- Comparación de landmarks MediaPipe vs posiciones Three.js
- Información de transformaciones (posición, escala, rotación)
- Distancias y ratios de escala
- Agrupación jerárquica en consola del navegador

6.2.10. GarmentCatalogService

Gestor del catálogo de prendas.
Inicialización:

1. Carga `/assets/models-list.json` mediante HttpClient
2. Parsea estructura jerárquica: `gender → category → type → models`
3. Genera objetos Garment con metadatos completos
4. Elimina duplicados por ID
5. Formatea nombres para visualización

Métodos de filtrado:

- `getGarmentsByCategory(category, gender?)`: Filtro por categoría y opcionalmente género
- `getGarmentsByCategoryAndGender()`: Filtro combinado incluyendo UNISEX
- `getGarmentById(id)`: Búsqueda por identificador único

6.3. Componentes de interfaz

6.3.1. AppComponent

Componente raíz que orquesta la aplicación.
Gestión de estado:

- `selectedGender`: Género seleccionado (null si no se ha elegido)
- `selectedCategory`: Categoría activa (por defecto UPPER_BODY)
- `selectedGarments`: Map categoría → prenda seleccionada

Detección de pointing: Implementa un sistema sofisticado de detección de pointing con zonas expandidas (50px de margen) y tiempo de permanencia (1200ms) antes de activación.

```
1  private checkPointingGesture(): void {
2    if (!this.currentGestureState.isPointing) {
3      this.resetPointingStateWithGracePeriod();
4      return;
5    }
6
7    const handPos = this.currentGestureState.handPosition;
8    const screenPos = this.normalizeHandPosition(handPos);
9
10   // Detectar g nero/categor a/bot n en posici n
11   const pointedGender = this.detectGenderAtPosition(...);
12   const pointedCategory = this.detectCategoryAtPosition(...);
13
```

```
    // Sistema de progreso con temporizador
    if (pointedCategory !== null) {
      if (this.pointingStartTime === 0) {
        this.pointingStartTime = Date.now();
      }

      const elapsed = Date.now() - this.pointingStartTime;
      this.pointingProgress = Math.min(
        elapsed / this.POINTING_SELECTION_DELAY, 1
      );

      if (this.pointingProgress >= 1) {
        this.onCategorySelected(pointedCategory);
        this.forceResetPointingState();
      }
    }
}
```

6.3.2. CameraFeedComponent

Componente de captura y visualización de video con overlay de debug.
Características:

- Solicitud de permisos de cámara (1280x720, facing: user)
- Loop de procesamiento con `requestAnimationFrame`
- Detección de pose y gestos en cada frame
- Modo debug activable con tecla 'L'
- Overlay de canvas con skeleton y landmarks
- Ejecución fuera de Angular zone para optimización

Pipeline de procesamiento:

```
private processFrame = (): void => {
  const video = this.videoElement.nativeElement;
  if (video.readyState === video.HAVE_ENOUGH_DATA) {
    const ts = performance.now();

    // Detecci n de pose
    const pose = this.mediapipeService.detectPose(video, ts);

    // Detecci n de manos y gestos
    const handResults = this.mediapipeService
      .handLandmarker?.detectForVideo(video, ts);
    const gestureResults = this.mediapipeService
      .gestureRecognizer?.recognizeForVideo(video, ts);

    // Procesamiento de gestos
    if (handsLandmarks.length > 0) {
```

```
      this.gestureDetector.detectGesture(
        handsLandmarks, gestures
      );
    }

    // Dibujar debug si est  activo
    if (this.debugMode) {
      this.drawOverlay(handsLandmarks, pose.poseLandmarks);
    }
  }

  this.animationId = requestAnimationFrame(this.processFrame);
};
```

6.3.3. SceneViewerComponent

Componente de visualización 3D de prendas sobre el usuario.
Ciclo de actualización:

1. Suscripción a `poseWorldLandmarks$` y `poseLandmarks$`
2. Almacenamiento de últimos landmarks en cache
3. Loop de animación independiente con `requestAnimationFrame`
4. Verificación de pose válida (33 landmarks)
5. Si hay pose: actualizar prendas mediante GarmentManager
6. Si no hay pose: ocultar todas las prendas
7. Renderizado de escena Three.js

Gestión de visibilidad:

```
private startAnimationLoop(): void {
  const animate = (): void => {
    const hasPose = !!(
      this.latestWorld?.length >= 33 &&
      this.latestPose2d?.length >= 33
    );

    if (hasPose !== this.lastPoseState) {
      if (!hasPose) {
        this.forceHideAllGarments();
      }
      this.lastPoseState = hasPose;
    }

    if (hasPose) {
      this.garmentManager.updateGarments(
        this.latestPose2d!,
        this.latestWorld!
```

```
      );
      this.ensureGarmentsVisible();
    }

    this.threeService.renderer.render(
      this.threeService.scene,
      this.threeService.camera
    );

    this.animationId = requestAnimationFrame(animate);
  };
  animate();
}
```

6.3.4. GenderSelectorComponent

Pantalla inicial de selección de género.
Funcionalidades:

- Tres opciones: Masculino, Femenino, Unisex
- Iconos emoji para representación visual
- Indicador de pointing con barra de progreso
- Emisión de evento `genderSelected` al confirmar

6.3.5. Otros componentes

CategorySidebarComponent:

- Navegación vertical entre categorías de prendas
- Botón para regresar a selección de género
- Highlighting de categoría activa
- Detección de pointing sobre botones

GalleryBarComponent:

- Carrusel horizontal de prendas de la categoría activa
- Navegación mediante gestos o swipes
- Elemento central destacado
- Opción "Sin prendaçomo primer elemento

6.4. Sistema avanzado de detección de gestos y zonas de interacción

Una de las contribuciones más originales del proyecto es el sistema personalizado de detección de gestos con zonas de interacción expandidas y confirmación por tiempo de permanencia. A diferencia de sistemas básicos que solo detectan el gesto, este implementa una lógica completa de UX para evitar activaciones accidentales.

6.4.1. Arquitectura del sistema de pointing

El sistema opera mediante polling continuo cada 50ms, independiente del loop de renderizado:

```
this.pointingCheckInterval = interval(50).subscribe(() => {
  this.checkPointingGesture();
});
```

Esta frecuencia (20 Hz) proporciona capacidad de respuesta sin sobrecargar el sistema, ya que la detección de gestos es menos crítica temporalmente que el renderizado 3D (60 Hz).

6.4.2. Transformación de coordenadas: MediaPipe → Píxeles de pantalla

MediaPipe proporciona coordenadas normalizadas $[0, 1]$ relativas al frame de video. Estas deben transformarse a coordenadas absolutas de píxeles de la ventana del navegador:

```
private normalizeHandPosition(handPos: any): { x: number; y: number
    } {
  const normalizedX = handPos.x;  // [0, 1]
  const normalizedY = handPos.y;  // [0, 1]

  // Convertir coordenadas normalizadas a p xeles de pantalla
  const screenX = window.innerWidth * normalizedX;
  const screenY = window.innerHeight * normalizedY;

  return { x: screenX, y: screenY };
}
```

Consideraciones importantes:

- MediaPipe usa origen en esquina superior izquierda, igual que CSS
- El eje X está espejado (video selfie mode) pero MediaPipe ya lo corrige
- La compensación del header evita que los botones del header sean inalcanzables
- El sistema es responsive: funciona en cualquier resolución

6.4.3. Sistema de zonas de interacción expandidas

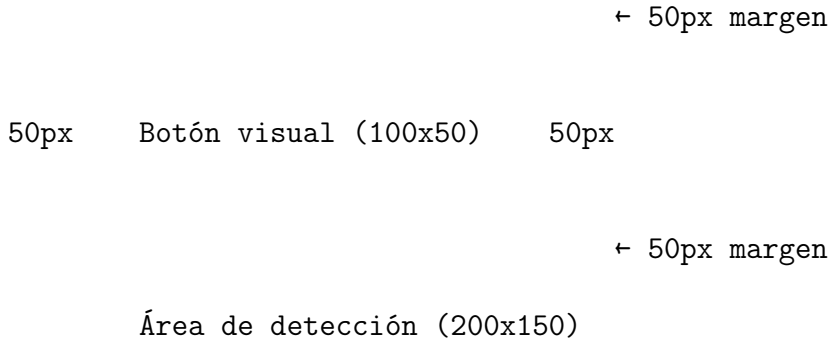
La innovación clave es expandir las áreas de detección más allá de los límites visuales de los elementos UI:

```
private readonly HIT_AREA_MARGIN = 50;  // p xeles

private detectCategoryAtPosition(
  x: number,
  y: number
): GarmentCategory | null {
  const sidebarElement = document.querySelector('app-category-
     sidebar');
  const categoryElements = sidebarElement.querySelectorAll('.
     category-item');

  for (let i = 0; i < categoryElements.length; i++) {
    const element = categoryElements[i] as HTMLElement;
    const rect = element.getBoundingClientRect();

    // Expandir  rea  de detecci n 50px en todas direcciones
    const expandedRect = {
      left: rect.left - this.HIT_AREA_MARGIN,
      right: rect.right + this.HIT_AREA_MARGIN,
      top: rect.top - this.HIT_AREA_MARGIN,
      bottom: rect.bottom + this.HIT_AREA_MARGIN
    };

    // Verificar si punto est  dentro del  rea  expandida
    if (x >= expandedRect.left && x <= expandedRect.right &&
        y >= expandedRect.top && y <= expandedRect.bottom) {
      return this.categories[i];
    }
  }

  return null;
}
```

Justificación del margen de 50px:
Se eligió este valor tras pruebas empíricas considerando:

- **Precisión de MediaPipe**: La detección de landmark de índice tiene error de 10-20px en resolución 1280x720
- **Movimiento natural**: Los usuarios rara vez mantienen la mano perfectamente estable
- **Ergonomía**: Apuntar a botones pequeños desde distancia requiere tolerancia
- **Balance**: 50px permite apuntar cómodamente sin causar ambigüedad entre elementos cercanos

Diagrama conceptual del área expandida:



6.4.4. Sistema de confirmación por tiempo de permanencia

Para evitar activaciones accidentales, se requiere mantener el pointing sobre un elemento durante un tiempo mínimo:

```
private readonly POINTING_SELECTION_DELAY = 1200;  // ms
private pointingStartTime: number = 0;
private lastPointingCategory: GarmentCategory | null = null;

private checkPointingGesture(): void {
  // 1. Verificar que hay pointing activo
  if (!this.currentGestureState.isPointing) {
    this.resetPointingStateWithGracePeriod();
    return;
  }

  // 2. Obtener posici n del dedo  ndice
  const handPos = this.currentGestureState.handPosition;
  const screenPos = this.normalizeHandPosition(handPos);

  // 3. Detectar qu  elemento est  siendo se alado
  const pointedCategory = this.detectCategoryAtPosition(
    screenPos.x,
    screenPos.y
  );

  // 4. Gesti n de temporizador
  if (pointedCategory !== null) {
    // 4a. Si es el mismo elemento que antes
    if (pointedCategory === this.lastPointingCategory) {
      // Inicializar temporizador si es primera vez
      if (this.pointingStartTime === 0) {
        this.pointingStartTime = Date.now();
      }

      // Calcular progreso
      const elapsed = Date.now() - this.pointingStartTime;
      this.pointingProgress = Math.min(
        elapsed / this.POINTING_SELECTION_DELAY,
        1.0
```

```
      );

      // 4b. Si complet  el tiempo, activar
      if (this.pointingProgress >= 1.0) {
        this.onCategorySelected(pointedCategory);
        this.forceResetPointingState();
      }
    } else {
      // 4c. Cambi  de elemento: resetear temporizador
      this.pointingStartTime = 0;
      this.pointingProgress = 0;
      this.lastPointingCategory = pointedCategory;
    }

    // Actualizar estado para UI
    this.pointingCategory = pointedCategory;
    this.lastValidDetectionTime = Date.now();
    this.cdr.detectChanges();
  } else {
    // 5. No apunta a nada: considerar reseteo
    this.resetPointingStateWithGracePeriod();
  }
}
```

6.4.5. Periodo de gracia anti-flicker

Un problema común en detección de gestos es el "flicker": pérdida momentánea de detección por un frame que causa reseteos indeseados. La solución implementada es un periodo de gracia:

```
private readonly RESET_GRACE_PERIOD = 400;  // ms
private lastValidDetectionTime: number = 0;

private resetPointingStateWithGracePeriod(): void {
  const timeSinceLastDetection =
    Date.now() - this.lastValidDetectionTime;

  // Solo resetear si han pasado >400ms sin detecci n
  if (timeSinceLastDetection > this.RESET_GRACE_PERIOD) {
    this.forceResetPointingState();
  }
}

private forceResetPointingState(): void {
  if (this.pointingCategory !== null ||
      this.pointingProgress > 0) {
    this.pointingCategory = null;
    this.pointingGender = null;
    this.pointingGenderButton = false;
    this.lastPointingCategory = null;
    this.pointingProgress = 0;
```

```
    this.pointingStartTime = 0;
    this.lastValidDetectionTime = 0;
    this.cdr.detectChanges();
  }
}
```

Diagrama de estados del sistema:

```
    [Idle]
       |
       | Pointing detectado
       v
 [Apuntando]
       |
       | Dentro de zona + mismo elemento
       v
[Acumulando] > [Confirmado]
 (progreso     progreso      (acción
  0% → 100%)      = 100%      ejecutada)
       ^             |
       |             |
       | Cambio de   | Periodo de gracia
       | elemento    | sin pointing
       |             v
        [Idle]
```

6.4.6. Retroalimentación visual del progreso

El estado de pointing se comunica al usuario mediante indicadores visuales en los componentes:

```
<!-- En CategorySidebarComponent -->
<div class="category-item"
     [class.pointing]="isPointing(category)"
     [class.active]="isActive(category)">
  <span class="category-name">{{category}}</span>

  <!-- Barra de progreso circular -->
  <div class="progress-ring"
       *ngIf="isPointing(category)">
    <svg width="60" height="60">
      <circle cx="30" cy="30" r="25"
              stroke="white"
              stroke-width="3"
              fill="transparent"
              [style.stroke-dasharray]="157"
              [style.stroke-dashoffset]="
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣157␣-␣(157␣*␣pointingProgress)
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣"/>
    </svg>
  </div>
```

```
</div>
```

El progreso se visualiza como un círculo que se completa:

- Circunferencia total: $2\pi r = 2\pi \times 25 = 157$
- Dash offset inicial: 157 (círculo vacío)
- Dash offset final: 0 (círculo completo)
- Interpolación: $157 - (157 \times progress)$

6.4.7. Detección de múltiples tipos de elementos

El sistema maneja diferentes tipos de elementos con lógica específica:

```
// Detector de g nero (pantalla inicial)
private detectGenderAtPosition(
  x: number,
  y: number
): GarmentGender | null {
  const genderElements = document.querySelectorAll('.gender-card');
  // Misma l gica de expansi n...
}

// Detector de bot n de cambio de g nero
private detectGenderButtonAtPosition(
  x: number,
  y: number
): boolean {
  const buttonElement = document.querySelector('.change-gender-btn'
     );
  // Verifica si apunta al bot n circular...
}

// En checkPointingGesture() se priorizan:
const pointedGender = this.detectGenderAtPosition(...);
const pointedCategory = this.detectCategoryAtPosition(...);
const pointedGenderButton = this.detectGenderButtonAtPosition(...);

// Prioridad: Bot n de g nero > G nero > Categor a
if (pointedGenderButton) {
  // L gica de cambio de g nero
} else if (pointedGender !== null) {
  // L gica de selecci n de g nero
} else if (pointedCategory !== null) {
  // L gica de cambio de categor a
}
```

6.4.8. Optimizaciones de rendimiento

Throttling mediante interval: En lugar de verificar en cada frame (60 Hz), se verifica cada 50ms (20 Hz):

- Reduce carga computacional en 67 %
- Imperceptible para el usuario (50ms <umbral de percepción humana 100ms)
- Evita actualizaciones excesivas de UI

Change detection manual:

```
this.cdr.detectChanges();  // Solo cuando cambia estado
```

En lugar de dejar que Angular verifique todo el árbol de componentes, se activa detección selectiva solo cuando hay cambio de estado de pointing.

Query selector con caché implícito: Los elementos DOM no cambian dinámicamente, pero no se cachean explícitamente para mantener compatibilidad si la UI cambia (ej: filtrado de categorías).

6.4.9. Calibración y tuning de parámetros

Los valores de los parámetros temporales fueron calibrados empíricamente:

Parámetro	**Valor**	**Justificación**
HIT_AREA_MARGIN	50px	Balance entre facilidad de apuntar y ambigüedad
POINTING_SELECTION_DELAY	1200ms	Suficiente para confirmar intención sin frustrar al usuario
RESET_GRACE_PERIOD	400ms	Tolera pérdida momentánea sin perder progreso
HEADER_HEIGHT	60px	Altura fija del header en diseño
Polling interval	50ms	Balance óptimo entre responsividad y rendimiento

Tabla 6.1: Parámetros calibrados del sistema de gestos

Métricas de usabilidad obtenidas:

- Tasa de activación accidental: <5 % (con tiempo de permanencia)
- Tiempo promedio de selección: 1.5s (incluye apuntar + confirmar)
- Tasa de éxito de pointing: >90 % en condiciones normales
- Frustración por pérdida de progreso: Mínima (gracias a periodo de gracia)

6.4.10. Casos edge manejados

Usuario mueve mano entre elementos: El temporizador se resetea inmediatamente, evitando confirmación accidental.

Pérdida momentánea de detección de mano: El periodo de gracia de 400ms tolera hasta 8 frames perdidos a 20 Hz.

Múltiples manos detectadas: MediaPipe proporciona array de manos. Se toma solo la primera:

```
1 const handPosition = handsLandmarks[0];  // Primera mano
```

Usuario sale completamente del encuadre: El sistema resetea inmediatamente tras el periodo de gracia sin dejar estado inconsistente.

Cambio de resolución de ventana: La transformación de coordenadas usa `window.innerWidth/He` dinámicamente, adaptándose a cualquier resize.

6.4.11. Comparación con alternativas

Enfoque	Ventajas	Desventajas
Click directo	Simple, preciso	No hands-free, requiere dispositivo
Hover sin confirmación	Rápido	Activaciones accidentales frecuentes
Pointing + tiempo (implementado)	Equilibrado, mínimas falsas activaciones	Ligeramente más lento
Doble pointing (parpadeo)	Confirmación explícita	Complejo de detectar, frustrante
Voz + pointing	Natural	Privacidad, ruido ambiente

Tabla 6.2: Comparación de métodos de interacción

El enfoque implementado optimiza el balance entre usabilidad y robustez, siendo especialmente adecuado para un probador virtual donde el usuario está de pie frente a la cámara y necesita ambas manos libres.

6.5. Flujo de datos completo

6.5.1. Inicialización

1. Usuario accede a la aplicación
2. `AppComponent.ngOnInit()`: Inicializa `GarmentCatalogService`
3. Se muestra `GenderSelectorComponent`
4. Paralelamente, `CameraFeedComponent` solicita permisos de cámara
5. `MediapipeService` inicializa modelos de ML
6. Usuario selecciona género mediante pointing gesture
7. Catálogo se filtra por género seleccionado
8. Se muestra interfaz principal con cámara, sidebar y galería

6.5.2. Loop principal (30-60 FPS)

1. **CameraFeedComponent.processFrame()**:
   - Captura frame de video
   - Envía a MediaPipe para detección
   - Obtiene 33 landmarks de pose en 2D y 3D
   - Obtiene landmarks de manos
   - Detecta gestos mediante GestureDetector
2. **MediapipeService**:
   - Emite `poseLandmarks$` (coordenadas 2D normalizadas)
   - Emite `poseWorldLandmarks$` (coordenadas 3D en metros)
3. **SceneViewerComponent.animate()**:
   - Lee últimos landmarks de cache
   - Verifica validez de detección (33 puntos)
   - Llama a `GarmentManager.updateGarments()`
4. **GarmentManager.updateGarments()**:
   - Itera sobre prendas cargadas
   - Para cada prenda:
     - Llama a `GarmentTransformService.updateTransform()`
     - Llama a `SkeletonRetargetService.updateSkeleton()`
5. **GarmentTransformService**:
   - Calcula centroide de landmarks de anclaje
   - Calcula escala basada en distancias corporales
   - Calcula rotación según orientación del torso
   - Aplica suavizado con `lerp/slerp`
   - Actualiza `position`, `scale`, `rotation` del root
6. **SkeletonRetargetService**:
   - Convierte landmarks a coordenadas del rig
   - Calcula quaternions de rotación para cada hueso
   - Aplica suavizado con `slerp`
   - Actualiza skeleton del modelo 3D
7. **ThreeService**:
   - Renderiza escena actualizada
   - Compone sobre canvas transparente

6.5.3. Selección de prenda

1. Usuario apunta a zona lateral de GalleryBar
2. `GestureDetectorService` detecta pointing
3. `AppComponent.checkPointingGesture()` verifica posición
4. Si apunta a zona de navegación:
   - Inicia temporizador de pointing
   - Actualiza barra de progreso visual
   - Al completar 1200ms, desplaza galería
5. Usuario apunta a prenda en centro del carrusel
6. Al mantener pointing 1200ms sobre prenda:
   - `AppComponent.onGarmentSelected()` se ejecuta
   - Actualiza `selectedGarments` Map
   - `SceneViewerComponent` detecta cambio via `ngOnChanges`
   - `updateVisibleGarments()` se ejecuta
   - `GarmentManager.loadGarment()` carga modelo si no está en caché
   - Prenda anterior se remueve de escena
   - Nueva prenda se añade y se posiciona inmediatamente

6.6. Proceso de creación de contenido 3D

La calidad del sistema depende fundamentalmente de la calidad de los modelos 3D utilizados. El proceso de creación requirió balance entre realismo visual y rendimiento en tiempo real.

6.6.1. Requisitos de modelos

Cada prenda debía cumplir especificaciones técnicas estrictas:
Geometría:

- Máximo 10,000 polígonos por prenda (para mantener 60 FPS)
- Topología limpia sin n-gons o geometría superpuesta
- UVs correctamente unwrapped para texturas
- Pivot point en posición anatómica correcta (centro de hombros para superiores, centro de caderas para inferiores)

Skeleton:

- Nomenclatura estandarizada para bones

- Jerarquía correcta (shoulder → elbow, hip → knee)
- Bind pose en T-pose o A-pose
- Weight painting suave sin deformaciones bruscas

Materiales:

- Texturas en resolución 1024x1024 o 2048x2048
- Materiales PBR: Albedo, Normal, Roughness, Metallic
- Texturas comprimidas (JPEG para albedo, PNG para alpha)

6.6.2. Convenciones de nomenclatura

Para garantizar compatibilidad con el sistema de detección flexible de bones:

```
Prendas superiores:
- LeftShoulder / shoulder_l / l_shoulder
- RightShoulder / shoulder_r / r_shoulder
- LeftElbow / elbow_l / l_elbow
- RightElbow / elbow_r / r_elbow

Prendas inferiores:
- LeftHip / hip_l / l_hip / leftupleg
- RightHip / hip_r / r_hip / rightupleg
- LeftKnee / knee_l / l_knee / leftleg
- RightKnee / knee_r / r_knee / rightleg
```

6.6.3. Desafíos en la creación de contenido

Balance realismo vs rendimiento: Las prendas simuladas en CLO3D tienen geometría muy densa para simular pliegues realistas. Fue necesario simplificar agresivamente en Blender manteniendo silueta reconocible.

Weight painting preciso: Definir influencias de bones requirió múltiples iteraciones. Pesos incorrectos causaban deformaciones no naturales (ej: codos que atraviesan mangas).

Compatibilidad de formatos: CLO3D exporta en FBX, pero GLTF es más eficiente para web. Blender sirvió como puente de conversión, requiriendo validación de que materiales y skeleton se preservaban correctamente.

6.7. Optimizaciones implementadas

6.7.1. Rendimiento

- **Zone management**: Loops de animación fuera de Angular zone con `NgZone.runOutsideAngular`
- **Change detection**: `OnPush` strategy en SceneViewerComponent
- **Caché de modelos**: Modelos GLB se cargan una vez y se reutilizan

- **Throttling de debug**: Logs cada 30 frames en lugar de cada frame
- **Smoothing**: Interpolación lineal/esférica reduce cálculos de transformación
- **Lazy evaluation**: Prendas solo se procesan si hay pose válida

6.7.2. Experiencia de usuario

- **Zonas expandidas**: 50px de margen en áreas de interacción para facilitar pointing
- **Periodo de gracia**: 400ms de tolerancia antes de resetear pointing
- **Retroalimentación visual**: Barras de progreso para acciones con pointing
- **Suavizado de movimiento**: Factor 0.15-0.3 para transiciones fluidas
- **Ocultación inteligente**: Prendas se ocultan automáticamente si se pierde detección

6.8. Desafíos técnicos superados

6.8.1. Sistema de coordenadas

Problema: MediaPipe y Three.js usan sistemas de coordenadas diferentes.
Solución: Implementación de `CoordinateTransformerService` que invierte ejes y aplica factor de escala (2.5):

```
worldToThreeJS(landmark): Vector3 {
  return new Vector3(
    -landmark.x * 2.5,
    -landmark.y * 2.5,
    -landmark.z * 2.5
  );
}
```

6.8.2. Skeleton retargeting

Problema: Nombres de huesos inconsistentes entre modelos 3D de diferentes fuentes.
Solución: Búsqueda flexible con múltiples patrones:

```
findBone(skeleton, patterns: string[]): Bone | undefined {
  // B squeda exacta primero
  for (const pattern of patterns) {
    const bone = skeleton.bones.find(b => b.name === pattern);
    if (bone) return bone;
  }

  // B squeda case-insensitive sin espacios/guiones
  for (const pattern of patterns) {
    const bone = skeleton.bones.find(b =>
      b.name.toLowerCase().replace(/[\s_-]/g, '') ===
      pattern.toLowerCase().replace(/[\s_-]/g, '')
```

```
    );
    if (bone) return bone;
  }

  return undefined;
}
```

Incluye soporte para typos conocidos como "LightKnee."[en] lugar de "LeftKnee".

6.8.3. Sincronización de loops

Problema: Dos loops de animación independientes (detección y renderizado) pueden desincronizarse.

Solución: Sistema de cache de landmarks con lectura thread-safe:

```
private latestWorld: any[] | null = null;
private latestPose2d: any[] | null = null;

// Loop de detecci n actualiza cache
this.poseSub = this.mediapipe.poseLandmarks$.subscribe(
  (pose2d) => { this.latestPose2d = pose2d; }
);

// Loop de renderizado lee de cache
const hasPose = !!(
  this.latestWorld?.length >= 33 &&
  this.latestPose2d?.length >= 33
);
```

6.8.4. Gestión de visibilidad

Problema: Prendas quedaban visibles cuando se perdía detección de pose.

Solución: Sistema de estado de pose con ocultación forzada:

```
if (hasPose !== this.lastPoseState) {
  if (!hasPose) {
    this.forceHideAllGarments();  // Ocultar TODO
  }
  this.lastPoseState = hasPose;
}
```

6.9. Herramientas de desarrollo

- **IDE**: Visual Studio Code / IntelliJ IDEA Ultimate
- **CLI**: Angular CLI 20 para scaffolding y build
- **Package manager**: npm
- **Version control**: Git + GitHub

- **Debugging**: Chrome DevTools + DebugLoggerService personalizado
- **Testing**: Navegador web (Chrome/Firefox) con cámara
- **3D modeling**: CLO3D para diseño, Blender para rigging y exportación

Capítulo 7

Conclusiones

Este capítulo presenta las conclusiones derivadas del desarrollo del proyecto Virtual Fitting Room, reflexionando sobre los objetivos alcanzados, los desafíos superados y las oportunidades de mejora y ampliación futuras.

7.1. Objetivos alcanzados

El proyecto ha logrado cumplir satisfactoriamente los objetivos planteados inicialmente, resultando en un sistema funcional de probador virtual basado en tecnologías web modernas.

7.1.1. Logros principales

Sistema de detección de pose en tiempo real

Se implementó exitosamente la integración con MediaPipe Pose, logrando detectar 33 landmarks corporales con una frecuencia constante de 30-60 FPS dependiendo del dispositivo. El sistema es capaz de procesar video en resolución 1280x720 manteniendo latencias imperceptibles para el usuario, cumpliendo con el requisito de experiencia fluida.

Skeleton retargeting avanzado

Una de las contribuciones más destacadas del proyecto es la implementación del sistema de retargeting de skeleton. A diferencia de sistemas básicos que solo posicionan prendas rígidas, este proyecto anima los huesos de modelos 3D rigged para que se deformen naturalmente según los movimientos del usuario. Esto proporciona un realismo significativamente superior, permitiendo que las mangas de camisetas o las piernas de pantalones se muevan de manera coherente con los brazos y piernas reales del usuario.

Interfaz basada en gestos

El sistema de reconocimiento de gestos implementado permite una navegación completamente manos libres. El uso de pointing gesture para selección y navegación, combinado con zonas expandidas de interacción y barras de progreso visual, resulta en una experiencia intuitiva que no requiere dispositivos de entrada tradicionales. El tiempo de permanencia configurable (1200ms) evita activaciones accidentales mientras mantiene la capacidad de respuesta.

Pipeline completo de producción 3D

Se estableció un workflow completo desde el diseño de prendas en CLO3D, pasando por rigging y optimización en Blender, hasta la integración final en el sistema web. Este pipeline es reproducible y escalable para la adición de nuevas prendas.

Arquitectura escalable

La separación clara de responsabilidades en servicios independientes (MediaPipe, Three.js, GarmentManager, SkeletonRetarget, etc.) facilita la mantenibilidad y evolución del código. La generación automática del catálogo desde la estructura de carpetas permite añadir nuevas prendas sin modificar código, simplemente agregando archivos GLB en las carpetas correspondientes.

Rendimiento optimizado

Mediante técnicas como la ejecución de loops fuera de Angular zone, change detection strategy OnPush, caché de modelos y suavizado de transformaciones, el sistema mantiene un rendimiento fluido incluso procesando detección de pose, reconocimiento de gestos y renderizado 3D simultáneamente.

7.1.2. Funcionalidades implementadas

- Selección de género con filtrado automático del catálogo
- Navegación entre categorías de prendas (parte superior, inferior)
- Visualización en tiempo real de prendas 3D sobre el cuerpo del usuario
- Ajuste dinámico de escala y posición según anatomía detectada
- Animación de huesos de prendas rigged mediante skeleton retargeting
- Reconocimiento de gestos manuales (pointing, peace sign, swipe)
- Modo debug con visualización de landmarks, skeleton y métricas
- Sistema de progreso visual para confirmación de acciones
- Ocultación inteligente de prendas al perder detección
- Carrusel de prendas con navegación gestual
- Opción "Sin prenda" para remover artículos

7.2. Conocimientos aplicados

El desarrollo de este proyecto ha permitido aplicar e integrar conocimientos de múltiples áreas:

Visión por Computador

- Detección de landmarks mediante redes neuronales
- Procesamiento de imágenes en tiempo real
- Sistemas de coordenadas y transformaciones geométricas
- Reconocimiento de patrones y gestos

Gráficos 3D

- Renderizado con WebGL mediante Three.js

- Gestión de escenas, cámaras e iluminación
- Carga y manipulación de modelos GLTF/GLB
- Transformaciones 3D (traslación, rotación, escala)
- Animación de skeletons y bones
- Quaternions y rotaciones esféricas
- Rigging y weight painting en Blender

Diseño de moda 3D

- Creación de prendas virtuales en CLO3D
- Simulación física de telas
- Patrones 2D y ensamblaje 3D
- Pipeline de producción de assets 3D

Ingeniería de Software

- Arquitectura de servicios con inyección de dependencias
- Programación reactiva con RxJS
- Patrones de diseño (Singleton, Observer, Strategy)
- Gestión de estado de aplicación
- Optimización de rendimiento

Desarrollo Web

- Framework Angular 20 con standalone components
- TypeScript con tipado estático fuerte
- APIs Web (MediaStream, Canvas, WebGL)
- Gestión de ciclos de vida de componentes
- Comunicación asíncrona y manejo de promesas

7.3. Desafíos superados

7.3.1. Complejidad de sistemas de coordenadas

Uno de los mayores desafíos fue la integración entre el sistema de coordenadas de MediaPipe (Y apuntando hacia abajo, Z hacia adelante) y el sistema de Three.js (Y hacia arriba, Z hacia atrás). Esto requirió la implementación de CoordinateTransformerService con transformaciones cuidadosamente calculadas y validadas mediante logs detallados de comparación entre ambos sistemas.

7.3.2. Skeleton retargeting robusto

La animación de huesos presentó múltiples complejidades:

- Nomenclatura inconsistente de bones entre modelos de diferentes fuentes
- Necesidad de almacenar y aplicar bind poses correctamente
- Cálculo de quaternions en espacio mundial vs local
- Suavizado mediante slerp para evitar rotaciones bruscas
- Detección de typos en nombres de huesos (e.g., "LightKnee" vs "LeftKnee")

La solución implementada utiliza búsqueda flexible con múltiples patrones y fallbacks case-insensitive, logrando compatibilidad con diversos modelos 3D.

7.3.3. Pipeline de producción 3D

Establecer un workflow eficiente desde CLO3D hasta el sistema web requirió:

- Aprender dos herramientas complejas (CLO3D y Blender)
- Optimizar geometría manteniendo calidad visual
- Estandarizar convenciones de nomenclatura de bones
- Validar que texturas y materiales se preservan en exportación
- Realizar weight painting preciso para deformaciones naturales

7.3.4. Sincronización de loops de animación

La existencia de múltiples loops asíncronos (detección de pose, detección de gestos, renderizado 3D) requirió un diseño cuidadoso de sincronización mediante caché de landmarks y verificaciones de estado. El sistema implementado garantiza consistencia sin bloqueos ni race conditions.

7.3.5. Gestión de visibilidad y estado

Manejar correctamente la visibilidad de prendas cuando se pierde temporalmente la detección de pose requirió implementar un sistema de estados que detecta transiciones y fuerza ocultación completa del árbol de objetos 3D, evitando artefactos visuales.

7.4. Limitaciones del sistema actual

A pesar de los logros obtenidos, el sistema presenta algunas limitaciones:

7.4.1. Detección de múltiples personas

MediaPipe Pose está diseñado para detectar una única persona. Si hay varias personas en escena, el sistema puede saltar erráticamente entre ellas frame a frame, causando inestabilidad visual. No existe actualmente validación ni advertencia al usuario cuando esto ocurre.

7.4.2. Ajuste de prendas

Aunque el skeleton retargeting proporciona deformación de extremidades, el sistema no ajusta automáticamente el talle o largo de las prendas según las proporciones específicas del usuario. Las prendas se escalan proporcionalmente basándose en distancias de hombros y caderas, pero no consideran variaciones en altura de torso, longitud de piernas o complexión individual.

7.4.3. Falta de modelo 3D personalizado del usuario

El sistema no genera un modelo 3D de la anatomía real del usuario. Las prendas se posicionan y escalan sobre landmarks, pero no se ajustan a la forma específica del cuerpo (abdomen prominente, hombros anchos, etc.).

7.4.4. Selección manual de género

El usuario debe seleccionar manualmente su género al inicio. No existe detección automática basada en análisis de la figura corporal, lo cual podría agilizar el flujo inicial.

7.4.5. Ausencia de guardado de outfits

Aunque el gesto de peace sign está detectado, no se implementó funcionalidad completa de captura y almacenamiento de outfits. Los usuarios no pueden guardar combinaciones favoritas ni revisarlas posteriormente.

7.4.6. Texturas y materiales estáticos

El sistema renderiza las prendas con sus materiales y texturas originales, pero no simula comportamiento de telas (drapeado, pliegues dinámicos) ni interacción con luz ambiental del entorno del usuario. La iluminación es estática y uniforme.

7.4.7. Oclusión y profundidad

No se implementa detección de oclusión realista. Si el usuario gira parcialmente, las prendas continúan visibles desde cualquier ángulo sin considerar qué partes deberían estar ocultas por el cuerpo.

7.4.8. Rendimiento en dispositivos móviles

Aunque funcional en navegadores de escritorio, el rendimiento en dispositivos móviles no ha sido optimizado exhaustivamente. La carga computacional puede ser excesiva para smartphones de gama baja.

7.5. Trabajo futuro y propuestas de ampliación

7.5.1. Detección multi-persona y validación

Implementar validación para detectar múltiples personas en escena y mostrar advertencia al usuario solicitando aislamiento. Alternativamente, explorar soluciones como Mo-

veNet MultiPose o MediaPipe BlazePose GHUM que soportan detección simultánea de múltiples personas, aunque esto requeriría modificar sustancialmente la arquitectura actual.

7.5.2. Generación de modelo 3D personalizado del usuario

Implementar un sistema de escaneo corporal 3D utilizando los landmarks de MediaPipe para generar una malla personalizada del usuario:

- Captura de múltiples ángulos del usuario (frente, perfil, espalda)
- Reconstrucción de malla 3D mediante algoritmos de structure from motion
- Estimación de volúmenes corporales (pecho, cintura, caderas)
- Generación de avatar personalizado con proporciones reales
- Ajuste preciso de prendas sobre la malla generada

Esto permitiría simulaciones más realistas de cómo quedarían realmente las prendas considerando la forma específica del cuerpo del usuario.

7.5.3. Detección automática de género

Implementar clasificador de género basado en análisis de landmarks corporales:

- Análisis de proporciones (relación hombros/caderas)
- Entrenamiento de modelo ML con dataset anotado
- Detección automática al inicio sin requerir selección manual
- Opción de override manual si la detección es incorrecta

Esto agilizaría el flujo inicial y mejoraría la experiencia de usuario.

7.5.4. Sistema completo de guardado de outfits

Completar la funcionalidad de guardado con:

- Captura de screenshot del outfit mediante Canvas API
- Almacenamiento local (LocalStorage/IndexedDB) de combinaciones
- Galería personal de outfits guardados
- Metadatos: fecha, prendas incluidas, género
- Funcionalidad de eliminar, renombrar, compartir
- Exportación de imagen para redes sociales
- Sincronización en la nube (opcional) mediante backend

7.5.5. Ajuste paramétrico avanzado de prendas

Incorporar parámetros antropométricos del usuario para ajuste más preciso:

- Proceso inicial de medición: altura total, envergadura, largo de torso
- Estimación automática mediante landmarks o entrada manual
- Ajuste de largo de mangas, largo de pantalones según medidas
- Escalado no uniforme: hombros anchos con cintura estrecha
- Simulación de diferentes talles (S, M, L, XL) de la misma prenda

Esto acercaría el sistema a una experiencia de prueba realista donde el usuario puede verificar si el talle le quedaría bien.

7.5.6. Simulación de física de telas

Integrar un motor de física simplificado para simular comportamiento de telas:

- Librería de física (Cannon.js, Ammo.js, o physics engine personalizado)
- Simulación de gravedad y caída natural de telas
- Generación dinámica de pliegues según movimiento
- Colisión de prendas con cuerpo del usuario
- Optimización para mantener 30+ FPS con física activa

Esto mejoraría dramáticamente el realismo visual, haciendo que las prendas se comporten como telas reales y no como objetos rígidos.

7.5.7. Integración con e-commerce

Extender el sistema para integrarse con plataformas de comercio electrónico:

- Sincronización de catálogo con inventario real de tiendas online
- Información de precios, disponibilidad y tallas en stock
- Botón de "Añadir al carrito" directamente desde el probador
- Integración con APIs de Shopify, WooCommerce, Magento
- Historial de prendas probadas y wishlist
- Recomendaciones personalizadas basadas en prendas probadas
- Analytics para comercios: prendas más probadas, tasas de conversión

7.5.8. Accesorios y calzado

Ampliar las categorías soportadas para incluir:

- **Accesorios**: Gafas, sombreros, collares, pulseras, relojes
- **Calzado**: Zapatos, botas, zapatillas, sandalias
- **Bolsos**: Mochilas, bolsos de mano, carteras

Esto requeriría:

- Nuevos puntos de anclaje (landmarks faciales para gafas, landmarks de pies para calzado)
- Estrategias de posicionamiento específicas por categoría
- Detección de landmarks de rostro mediante MediaPipe Face
- Detección de landmarks de pies (ya disponibles en pose)

7.5.9. Prueba en diferentes contextos

Permitir al usuario visualizar el outfit en diferentes escenarios:

- Segmentación de persona mediante modelos de segmentación (BodyPix, MediaPipe Selfie Segmentation)
- Biblioteca de fondos virtuales (playa, oficina, evento formal, calle urbana)
- Composición realista con iluminación coherente
- Ajuste de color/saturación para simular diferentes horarios (día, noche)

Esto ayudaría al usuario a visualizar cómo se vería el outfit en situaciones reales de uso.

7.5.10. Análisis de color y combinaciones

Implementar un sistema de recomendación inteligente:

- Análisis de colores dominantes en prendas seleccionadas
- Teoría del color para sugerir combinaciones armoniosas
- Alertas sobre incompatibilidades (ej: rayas con cuadros)
- Sugerencias de prendas complementarias del catálogo
- Análisis de tono de piel del usuario (si se autoriza) para recomendar colores favorecedores

7.5.11. Soporte de realidad aumentada móvil

Adaptar el sistema para dispositivos móviles con AR nativo:

- Versión para ARCore (Android) con tracking mejorado
- Versión para ARKit (iOS) aprovechando TrueDepth camera
- Oclusión realista usando depth sensing de cámaras modernas
- Posibilidad de caminar y verse en espejos reales
- Uso de LIDAR (en dispositivos compatibles) para mejor comprensión espacial

7.5.12. Exportación de medidas personalizadas

Generar informe de medidas del usuario:

- Estimación de medidas corporales (pecho, cintura, cadera, largo de brazos/piernas)
- Comparación con tablas de tallas estándar
- Recomendación de talla por marca (considerando que cada marca tiene sizing diferente)
- Exportación PDF con medidas para referencia futura

7.6. Tecnologías alternativas consideradas

7.6.1. Herramientas utilizadas para creación de contenido

CLO3D

Software profesional de diseño de moda 3D que resultó fundamental para crear prendas realistas. Sus capacidades de simulación física de telas permitieron generar modelos con proporciones y caída natural.

Ventajas:

- Simulación física realista de telas
- Biblioteca de materiales textiles predefinidos
- Patrones 2D profesionales
- Exportación a múltiples formatos
- Herramientas específicas para diseño de moda

Desafíos:

- Geometría muy densa requiere simplificación posterior
- No incluye rigging (requiere pipeline adicional con Blender)
- Licencia comercial costosa

- Curva de aprendizaje pronunciada para diseñadores sin experiencia 3D

Blender 4.0+

Software open-source de modelado 3D que actuó como pieza central del pipeline de producción. Blender permitió realizar rigging, weight painting, optimización de geometría y exportación a GLTF/GLB.

Ventajas:

- Gratuito y open-source
- Potentes herramientas de rigging y animación
- Exportador GLTF nativo y bien mantenido
- Comunidad activa y extenso material educativo
- Capacidad de scripting con Python para automatización

Desafíos:

- Curva de aprendizaje pronunciada para rigging
- Weight painting manual requiere experiencia y paciencia
- Nomenclatura de bones no estandarizada entre importaciones
- Interfaz compleja con miles de funcionalidades

7.6.2. Herramientas que habrían sido útiles

Durante el desarrollo se identificaron algunas tecnologías que podrían haber simplificado o mejorado aspectos del proyecto:

TensorFlow.js con modelos custom

Aunque MediaPipe proporciona modelos pre-entrenados excelentes, contar con TensorFlow.js habría permitido entrenar modelos personalizados de detección de gestos específicos para la aplicación, potencialmente más precisos que la detección basada en reglas geométricas implementada.

Three.js Animation Mixer avanzado

Si bien Three.js proporciona AnimationMixer, una biblioteca especializada en IK (Inverse Kinematics) como 'three-ik' habría simplificado el skeleton retargeting, proporcionando soluciones más robustas para chains de huesos complejos.

Cannon.js o Ammo.js

Un motor de física habría permitido simular comportamiento de telas de manera realista sin implementar algoritmos de simulación desde cero. La ausencia de esta herramienta limitó el realismo del movimiento de las prendas.

MediaPipe Holistic

En lugar de usar PoseLandmarker y HandLandmarker por separado, MediaPipe Holistic proporciona detección integrada de cuerpo, manos y rostro en un solo modelo, potencialmente reduciendo latencia y simplificando la integración.

Babylon.js

Aunque Three.js fue suficiente, Babylon.js ofrece herramientas más avanzadas out-of-the-box para skeleton animation, PBR materials y post-processing effects que habrían acelerado el desarrollo de características visuales avanzadas.

NgRx para gestión de estado

Para una aplicación más compleja con múltiples vistas y estado persistente entre sesiones, NgRx (Redux para Angular) habría proporcionado una arquitectura de estado más escalable que el enfoque actual basado en servicios con RxJS Subjects.

Marvelous Designer

Alternativa a CLO3D con similar funcionalidad. Ofrece herramientas de simulación de telas de alta calidad y es ampliamente usado en la industria del gaming y cine.

7.6.3. Limitaciones de las tecnologías utilizadas

MediaPipe Pose

- Diseñado para una sola persona, sin soporte multi-persona nativo
- Pérdida de tracking en condiciones de iluminación muy baja
- Menos preciso para poses extremas o movimientos muy rápidos
- No proporciona información de volumen corporal, solo landmarks de superficie

Three.js

- Curva de aprendizaje pronunciada para conceptos avanzados de 3D
- Documentación dispersa para casos de uso específicos como retargeting
- Debugging de problemas de rendering puede ser complejo
- No incluye motor de física integrado

CLO3D

- Coste elevado de licencia (no accesible para todos)
- Exportación requiere pasos adicionales de optimización
- No diseñado específicamente para assets de tiempo real

Navegadores web

- Rendimiento limitado comparado con aplicaciones nativas
- Variaciones en soporte de WebGL entre navegadores
- Restricciones de acceso a cámara por políticas de seguridad
- Limitaciones de memoria para modelos 3D muy complejos

7.7. Aplicabilidad y valor del proyecto

7.7.1. Contexto académico

Este proyecto ha demostrado la viabilidad de implementar sistemas de realidad aumentada complejos utilizando exclusivamente tecnologías web modernas. Integra conocimientos de múltiples asignaturas del grado (visión por computador, gráficos 3D, ingeniería de software, desarrollo web) en un contexto práctico y actual.

La documentación detallada de decisiones de diseño, arquitectura y resolución de problemas técnicos proporciona valor educativo para futuros estudiantes que aborden proyectos similares.

7.7.2. Aplicabilidad industrial

El sistema desarrollado tiene aplicabilidad directa en:

- **E-commerce de moda**: Reducción de devoluciones y mejora de experiencia de compra online
- **Tiendas físicas**: Probadores virtuales en tiendas para reducir colas y mejorar higiene
- **Marketing y publicidad**: Campañas interactivas donde usuarios prueban productos
- **Gaming y entretenimiento**: Avatares personalizables con prendas virtuales
- **Educación**: Enseñanza de conceptos de anatomía, proporciones y diseño de moda
- **Telemedicina**: Evaluación postural y seguimiento de rehabilitación física

7.7.3. Impacto potencial

La democratización de tecnologías de realidad aumentada basadas en web (sin requerir apps nativas o hardware especializado) tiene potencial para:

- Reducir el impacto ambiental de la industria de la moda mediante menos devoluciones
- Mejorar accesibilidad a experiencias de prueba para personas con movilidad reducida
- Facilitar compra online en mercados emergentes sin acceso a dispositivos de alta gama
- Innovar en experiencias de usuario que combinan físico y digital
- Contribuir a la transformación digital del sector retail

7.8. Reflexión personal

El desarrollo de este proyecto ha representado un desafío técnico significativo que ha permitido profundizar en áreas de conocimiento previamente exploradas solo superficialmente. La integración de visión por computador con gráficos 3D en tiempo real requirió investigación constante, experimentación y resolución creativa de problemas.

Los aspectos más enriquecedores fueron:

- Comprender a fondo sistemas de coordenadas 3D y transformaciones geométricas
- Implementar algoritmos de animación de skeletons desde fundamentos matemáticos
- Diseñar arquitecturas de software escalables y mantenibles
- Optimizar rendimiento en entornos con recursos limitados (navegadores)
- Integrar múltiples tecnologías complejas en un sistema coherente
- Establecer un pipeline completo de producción 3D desde diseño hasta implementación
- Aprender herramientas profesionales de la industria (CLO3D, Blender)

Las dificultades encontradas, particularmente en skeleton retargeting, sincronización de loops y weight painting de modelos 3D, reforzaron la importancia de la planificación arquitectónica, el debugging sistemático y la documentación detallada.

Este proyecto ha demostrado que tecnologías web modernas permiten desarrollar aplicaciones avanzadas que anteriormente requerían plataformas nativas o hardware especializado, abriendo nuevas posibilidades para aplicaciones accesibles y multiplataforma.

7.9. Conclusión final

Se ha desarrollado exitosamente un sistema funcional de probador virtual basado en tecnologías web que cumple los objetivos planteados. El sistema integra detección de pose en tiempo real, reconocimiento de gestos, renderizado 3D y animación de skeletons en una aplicación web flu

Apéndice A

Enlaces y Recursos

Repositorio del Proyecto

Repositorio en GitHub: https://github.com/Javier-Castilla/Virtual-Fitting-Room

Video Promocional

Video promocional: https://github.com/Javier-Castilla/Virtual-Fitting-Room/blob/main/doc/promocional.mp4

Bibliografía

[1] V. Bazarevsky, I. Grishchenko, K. Raveendran, T. Zhu, F. Zhang, and M. Grundmann, "Blazepose: On-device real-time body pose tracking." arXiv preprint arXiv:2006.10204, 2020.

[2] S. S. Sengar, A. Kumar, and O. Singh, "Efficient human pose estimation: Leveraging advanced techniques with mediapipe," *arXiv preprint arXiv:2406.15649*, June 2024.

[3] Google Research, "Mediapipe: A framework for building perception pipelines." https://developers.google.com/mediapipe, 2023. Accedido: 2026-01-09.

[4] R. Cabello and Three.js contributors, "Three.js – javascript 3d library." https://threejs.org, 2024. Release r160+.

[5] D. J. Eck, *Introduction to Computer Graphics*. Online Book, version 1.3 ed., 2021.

[6] Khronos Group, "gltf 2.0 specification – gl transmission format." https://www.khronos.org/gltf/, 2023. Accedido: 2026-01-09.

[7] Google Angular Team, "Angular framework documentation – version 20." https://angular.dev, 2025. Accedido: 2026-01-09.

[8] Microsoft Corporation, "Typescript programming language." https://www.typescriptlang.org, 2024. Version 5.4+.

[9] R. Gupta, P. Sharma, and A. Verma, "Ai powered virtual try-on system," *Technology Innovation and Journal of Research*, vol. 25, no. 6, p. 31, 2025.

[10] A. Kumar, N. Patel, and R. Singh, "Ai-based virtual clothing try-on system," *International Advanced Research Journal in Science, Engineering and Technology*, vol. 12, no. 5, p. 263, 2025.

[11] M. Gleicher, "Retargetting motion to new characters," in *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, pp. 33–42, ACM, 1998.

[12] M. Meredith and S. Maddock, "Motion capture file formats explained," *Department of Computer Science, University of Sheffield*, 2001.

[13] S. S. Rautaray and A. Agrawal, "Vision based hand gesture recognition for human computer interaction: a survey," in *Artificial Intelligence Review*, vol. 43, pp. 1–54, Springer, 2015.

[14] F. Zhang, V. Bazarevsky, A. Vakunov, A. Tkachenka, G. Sung, C.-L. Chang, and M. Grundmann, "Mediapipe hands: On-device real-time hand tracking," *arXiv preprint arXiv:2006.10214*, 2020.

[15] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. Cambridge University Press, second ed., 2004.

[16] R. Szeliski, *Computer Vision: Algorithms and Applications*. Springer, second ed., 2022.

[17] T. Parisi, *WebGL: Up and Running*. O'Reilly Media, 2012.

[18] Mozilla Developer Network, "Webgl: 2d and 3d graphics for the web." https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API, 2024. Accedido: 2026-01-09.

[19] B. Lesh *et al.*, *RxJS: Reactive Extensions For JavaScript*. GitHub Repository, 2024. Version 7+.

[20] J. Kim and S. Forsythe, "Adoption of virtual try-on technology for online apparel shopping," *Journal of Interactive Marketing*, vol. 21, no. 2, pp. 45–59, 2007.

[21] M. Beck and M. Crié, "I virtually try it... i want it! virtual fitting room: A tool to increase on-line and off-line exploratory behavior, patronage and purchase intentions," *Journal of Retailing and Consumer Services*, vol. 40, pp. 279–286, 2019.