Projeto Multimédia

Escola Superior de Educação de Coimbra

Licenciatura em Comunicação e Design Multimédia

2022/2023

Christopher Charles do Vale Cruz Lemos

2018039433

Indice

Introdução

Este relatório apresenta de forma detalhada o projeto desenvolvido na disciplina de Projeto Multimédia, abordando todas as etapas desde a sua concepção e planeamento até à execução, descrevendo e ilustrando os diversos componentes que constituem estas etapas.

Deste modo, esta introdução providencia uma visão geral do projeto: o seu contexto, objetivos e metodologia, servindo deste modo para contextualizar e facilitar a compreensão do conteúdo subsequente.

Contexto

Após deliberação com o professor orientador João Orvalho, foi decidido que o projeto a realizar seria o desenvolvimento de uma plataforma de realidade virtual, com um foco no uso do *A-Frame*[0], um framework que permite a criação de experiências de XR no browser.

Foi optado este caminho em específico pelo alinhamento de interesses e do seu potencial, possibilitando expandir o conhecimento de uma linguagem de programação familiar (Javascript) com tecnologias emergentes e com um grande potencial para valorização ao longo do tempo no que diz respeito a prospetivas profissionais.

Nas reuniões subsequentes, enquanto as tecnologias envolventes foram pesquisadas e praticadas, o âmbito do projeto foi sendo mais definido com o orientador, sendo direcionado para o campo da realidade virtual aplicado à medicina. Foram facultados scans tridimensionais de pacientes, a partir das quais se definiu o conceito final do projeto.

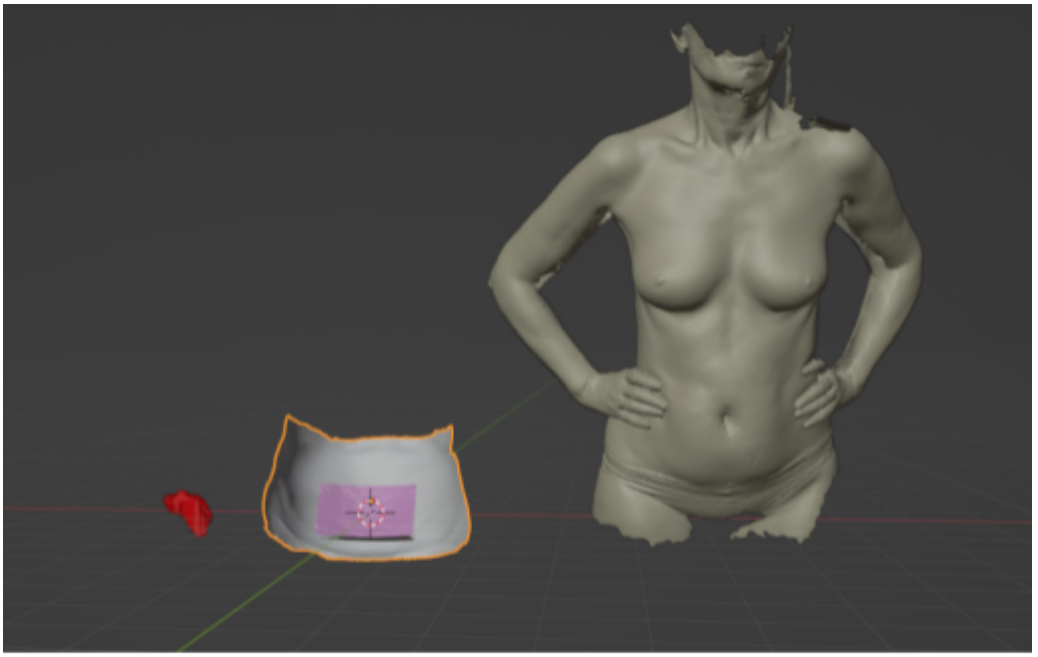

figura 1: scans paciente

Objetivos

O projeto evidencia uma multiplicidade e diversidade de modalidades e domínios, cada qual com os seus objetivos únicos.

Programação - esta modalidade é central ao projeto e consiste na àrea de maior foco e interesse, exibindo os seguintes objetivos:

- aprofundar o domínio do Javascript, consolidando os seus conceitos fundamentais e a sua compreensão a um nível lexical
- desenvolver capacidades de resolução de problemas a um nível abstrato, integrando abordagens e soluções mais elegantes, eficientes e estruturadas, bem como fortalecer a autonomia neste aspeto, procurando um pensamento mais flexível e criativo
- desenvolver capacidade de resolução de problemas a um nível técnico, melhorando os processos atuais de *testing* e *debugging,* incorporando as convenções e boas práticas modernas, assimilando ainda, melhores estratégias de pesquisa e análise
- construir um projeto de uma forma sistemática e organizada a um nível profissional tendo em conta convenções modernas e as melhores práticas de desenvolvimento de software
- aprender a criar código legível e sustentável, bem como uma documentação clara e concisa que descreve adequadamente o projeto e o código
- melhorar a capacidade de leitura de código, de modo a facilitar a compreensão de bases de código e frameworks existentes, permitindo deste modo uma maior adaptabilidade no curso de um projeto e na necessidade de colaborar com outros membros de uma equipa

- familiarização de tecnologias de XR, desde o equipamento utilizado à sua implementação e interação com as diversas frameworks/plataformas disponíveis
- familiarização de tecnologias 3D no browser, nomeadamente o A-Frame e por extensão o THREE.JS[1]
- praticar e aprofundar conhecimentos de desenvolvimento de web, traduzindo eficientemente e fielmente os designs da interface de utilizador através do uso de HTML e CSS

Design - esta modalidade terá um foco secundário no projeto, não sendo por isso negligenciável a sua importância para adequadamente transmitir a informação pretendida neste trabalho. Tendo por isso os seguintes objetivos:

- desenvolver uma experiência de utilizador (UX) intuitiva e harmoniosa com uma navegação clara e mecanismos de interação eficientes
- desenvolver uma experiência de realidade virtual imersiva e envolvente, tomando escolhas de design que tomam em consideração os limites da plataforma e os problemas associados com o uso de headsets virtuais, providenciado uma experiência confortável para o utilizador
- aprofundar conhecimentos de princípios de design com um foco na hierarquia visual e na organização de informação (cruciais em projeto de realidade virtual)
- definir uma estética consistente e adequada ao projeto, em prol de uma experiência coesiva, imersiva e memorável que alinha com os objetivos globais do projeto, a audiência e o tema
- implementar um design acessível e inclusivo de modo a garantir que o projeto é facilmente utilizável por grupo abrangente de utilizadores, incluindo indivíduos com deficiências

- aprender a implementar *usability testing* de modo a obter informações relevantes a potenciais problemas com o design
- aprofundar o conhecimento de programas de design como o figma

Animação/Modelação 3D - Dada a natureza do projeto, esta modalidade será habitualmente abordada, tendo por isso os seguintes objetivos:

- aprofundar conhecimentos mais técnicos de modelação 3D relevantes ao projeto, com a finalidade de conseguir interpretá-los programaticamente, aumentando o número de operações e mecanismos possíveis de criar no projeto
- aprender operações básicas relevantes ao projeto como o mapeamento e manipulação de texturas, exportação adequada de modelos em formatos compatíveis, limpar/arranjar modelos e técnicas de optimização
- aprender a usar o Blender em alternativa ao Cinema 4D

Educação - O projeto é fundamentalmente educacional, por isso há uma maior importância dada ao modo como a informação é transmitida, tem por isso os seguintes objetivos:

- transmitir com eficiência e rigor os conteúdos educacionais ao utilizador, priorizando veracidade, exatidão e a fiabilidade da informação apresentada
- organizar o conteúdo educacional de um modo coeso e linear para que os utilizadores conseguem facilmente acompanhar os passos
- dissecar o conteúdo para ser o mais simplificado e claro possível, tendo sempre em conta o primeiro objetivo
- elaborar formas imersivas e interativas de apresentar o conteúdo, tendo em conta o formato em que decorre

Gestão de Projeto - Fulcral ao projeto é a sua gestão, denotando por isso os seguintes objetivos:

- otimizar a alocação de recursos disponíveis, nomeadamente o tempo, para garantir uma execução eficiente do projeto
- melhorar aspetos comunicativos
- identificar riscos ao percurso do projeto e desenvolver estratégias para os mitigar
- implementar medidas para assegurar a qualidade do projeto
- criar e manter documentação compreensiva do projeto

Metodologia

Embora no decorrer do projeto não tenha sido propriamente estabelecido uma metodologia geral concreta, o desenvolvimento do projeto assemelha-se à abordagem *Agile*, que consiste num processo incremental e iterativo, na qual o projeto é repartido em componentes mais pequenas, possibilitando deste modo um feedback recorrente e uma maior flexibilidade consoante as necessidade do projeto.

Para além da *meta* metodologia, as seguintes modalidades refletiram uma metodologia subjacente:

Programação - Sendo a modalidade mais explorada neste projeto, é também a vertente que mais se aproximou do *Agile*, a sua complexidade levou naturalmente a um desenvolvimento incremental, iterativo e modular, estando em constante mudanças face aos problemas e soluções surgentes. Em termos de pesquisa, consistiu inicialmente de cursos online relevantes ao tópico, suplementado de outros websites, posts e comunidades. Ao longo do projeto, quando confrontado com problemas cada vez mais específicos, foi necessário recorrer a comunidades como o *Stackoverflow*[2], *GitHub*[3] e *Discord*[4], *Slack*[5] e ao auxílio de ferramentas como o *ChatGPT*[6].

Design - para esta modalidade recorreu-se a uma abordagem simplificada da metodologia "*Design Centrado no Usuário*"[7], que decorreu da seguinte forma:

- pesquisa inicial para entender qual seria o público alvo e as necessidades, objetivos, comportamentos e desafios destes.
- definição dos requisitos com base na pesquisa, identificando e definindo os requisitos do projeto, tomando nota das funcionalidades e características necessárias para atender às necessidades dos usuários.

- design e prototipagem, criando designs e protótipos iniciais com base nos requisitos identificados.
- testagem e interação, realizando testes e tomando nota de como interagia com a experiência de utilizador, com base nesse feedback, ajustava-se o design para melhorar a usabilidade e a experiência até obter o resultado final.

Em termos de pesquisa, consistiu também de cursos relevantes sendo suplementado com websites como o *UX Stack Exchange*[8] e o subreddit /r/web_dev[9] no caso de dúvidas específicas

Educação - Na elaboração desta vertente do projeto, tomou-se inspiração no modelo *SAM (Successive Approximation Model)*[10], com o objetivo de ter uma abordagem sistemática no design do conteúdo educacional. Em termos de pesquisa, esta consistiu de websites como o *Academia Stack Exchange*[11] e o Edutopia[12].

Projeto

Conceito

Foi facultado pelo orientador dois modelos, ambos scans tridimensionais de pacientes, um de uma mulher com um tumor mamário e o outro de uma zona abdominal com um *retalho*. Tomando estes modelos em conjunto, constituem uma reconstrução mamária designada de DIEAP (Deep Inferior Epigastric Artery Perforator)[11].

Esta operação serve de base para o projeto, que consiste numa experiência de aprendizagem guiada, concebido para fornecer um ambiente de aprendizagem estruturado e imersivo num espaço de realidade virtual. O utilizador segue uma sequência de eventos pré-programados e curados, via instruções que representam as várias fases da operação, abrangendo todos os componentes e procedimentos essenciais.

Guia de Utilização

Ao Inicializar a aplicação, o utilizador é transportado para um ambiente alienígena, direcionado para um *biodome*. Ao entrar no *biodome*, é revelado um espaço futurista projetado para simular cenários médicos realistas. O ambiente é imersivo e detalhado, com equipamentos médicos avançados e tecnologia de ponta. No centro de espaço há uma paciente em cima de uma mesa cirúrgica. O utilizador é recebido pelo guia de voz, uma interface de inteligência artificial que desempenha um papel fundamental durante toda a jornada na aplicação. O guia de voz é uma

assistente virtual projetada para instruir, informar e ajudar o utilizador em cada etapa do processo.

Dadas as boas vindas, o utilizador é instruído a dirigir-se ao paciente, ativando o evento seguinte, que consiste em revelar e explicar o equipamento e os instrumentos médicos disponibilizados. Cada ferramenta tem uma mecânica única, que permite ao utilizador um método de interagir com o paciente, estas mecânicas são ativadas, sob certas condições, pressionando o botão A.

É explicado o processo de agarrar os equipamentos, na qual o utilizador pode posicionar o raycaster sobre qualquer equipamento, ativando o seu estado de *hover,* e pressionar o botão de agarre (grip button) para fazê-la flutuar para as suas mãos.. Além disto, também é apresentada a opção de usar comandos de voz, simulando o ambiente de uma cirurgia real, na qual o cirurgião pode pedir a um enfermeiro um instrumento específico. Basta vocalizar o nome da ferramenta que esta dirige-se à mão do utilizador. Ao testar estas mecânicas, as condições necessárias são cumpridas, concluindo a explicação das mecânicas básicas, e procedendo aos passos da operação.

O primeiro passo consiste em demarcar as zonas de incisão com o marcador, marcando primeiro no abdómen com uma figura elíptica, e de seguida no peito afetado com a mesma figura. Com o marcador, basta apontar para o corpo e pressionar o botão A para começar a *pintar* sobre o paciente. O marcador, juntamente com o bisturi, tem um raycaster único para assistir na mira. Caso o objeto de interação, neste caso o paciente, esteja demasiado longe do utilizador, o raycaster apresenta uma cor branca translúcida, representando um estado de desativação, encurtando a distância ao paciente, esta muda para um branco mais sólido, indicando que pode ser ativado e projetado um alvo no ponto de interseção indicado onde o paciente será marcado, caso seja ativado, o raycaster passa a vermelho.

Ao concluir a demarcação no paciente, o utilizador é instruído para trocar de ferramenta para o bisturi, com o qual vai realizar o gesto prévio sobre as linhas marcadas, criando desta vez uma incisão na pele do paciente. Com os cortes efetuados, o utilizador tem agora que extrair o retalho cutâneo com a assistência de ambos o bisturi e a tenaz: com a tenaz o utilizador manipula o retalho para uma posição mais elevada, e com o bisturi realiza um corte nas artérias que o fixam. O retalho da mama é descartado, permitindo agora o acesso ao tumor, que é removido com a tenaz.

Com a remoção do tumor concluída, é necessário suturar os cortes. Começando com o peito, é utilizado o retalho previamente criado do abdômen, usando a tenaz para posicioná-lo no peito, quando este estiver suficientemente perto, indicado pela cor verde, basta largar o botão A que o retalho automaticamente adopta a forma do peito. Por fim, utiliza-se as suturas para fechar os cortes, apontando para as bordas da pele expostas, e usando os botões A e B para mudar o posicionamento da pele, os dois botões são usados para possibilitar direções diferentes. Repete-se o mesmo processo no peito, finalizando deste modo a operação.

Especificação Técnica

Programacão

Visao Geral

Esta aplicação foi desenvolvida utilizando Javascript e HTML, em conjunto com o A-Frame que providencia um *framework* para o desenvolvimento de aplicações de realidade mista (XR) na web. O A-Frame utiliza elementos de HTML personalizados de modo a criar uma experiência de desenvolvimento 3D que seja tão simples como configurar o DOM no desenvolvimento de web tradicional.

Este framework implementa uma arquitetura entidade-componente[13], este padrão de design facilita o desenvolvimento de lógica de jogos, sendo por isso ideal para o desenvolvimento de uma aplicação de simulação médica.

A biblioteca em si foi desenvolvida em cima do Three.JS, uma biblioteca popular usada para criar e visualizar elementos gráficos animados em 3D, que por sua vez foi desenvolvido em cima do WebGL[14], um API de baixo-nível usado para renderizar conteúdo em 3D. Esta relação significa que o A-Frame consegue usufruir do poder estendido do Three.JS e do WebGl, e em conjunto com javascript puro permite uma manipulação avançada do espaço tridimensional, algo de que o projeto usufrui repetidamente no seu desenvolvimento.

Além destas bibliotecas, o projeto utiliza outras bibliotecas no seu desenrolar, de modo a poder implementar funcionalidades mais complicadas e específicas como: construção geométrica sólida, físicas, uma framework de mãos virtuais e uma ferramenta de UI para o A-Frame.

Por fim, o projeto utiliza o Vite[14] como *build tool*, isto facilita e torna mais eficiente o desenvolvimento de projeto de web complicados, condensando os vários ficheiros js e módulos de node num único ficheiro *minified*, comprimindo os *assets* do projeto e contornando a necessidade de usar CDN's. Isto leva a uma aplicação mais performante, segura e consistente. O Vite possibilita isto e ainda providencia um ciclo de desenvolvimento mais eficiente graças à sua capacidade de *hot-module-replacement* e o seu alavancamento de módulos de ES nativos.

Estrutura do Projeto

De modo a conseguir uma base de código mais organizada e sustentável, o projeto está organizado por módulos. Este modo de organização permite que o código seja dividido em segmentos contidos, que podem ser importados e usados onde necessário. Esta abordagem tem os seguintes benefícios.

- **Reusabilidade de Código**: Os módulos promovem a reutilização de código, permitindo o uso de funções ou componentes em diferentes partes do aplicativo. Isso reduz a duplicação de código e facilita a manutenção e atualização da base de código.
- **Encapsulamento**: Os módulos permitem a encapsulação de funcionalidades e dados, evitando conflitos potenciais entre diferentes partes do código. Isso melhora a segurança do código e facilita a compreensão do comportamento e da lógica do aplicativo.
- **Escalabilidade**: Conforme o projeto cresce, gerir uma base de código torna-se mais complicado. O uso de módulos permite uma escalabilidade mais fácil do aplicativo, dividindo-o em partes gerenciáveis, tornando mais fácil digerir uma base de código complexa e em constante crescimento.
- **Legibilidade**: Com os módulos, a base de código torna-se mais legível e compreensível, já que cada módulo concentra-se num aspecto específico da funcionalidade do aplicativo. Isso facilita a compreensão da arquitetura geral do aplicativo.

O uso de módulos é implementado via a sintaxe de módulos ES6 do JavaScript, permitindo a definição de módulos separados dentro de ficheiros individuais e a importação/exportação de funções, variáveis e objetos para serem usados em outras partes da aplicação.

main.js

```
import * as AFRAME from 'aframe'
import 'super-hands'
import 'aframe-physics'
import 'aframe-environment'
import 'aframe-extras'
import './utils/index'
import { renderMaterials, storeInitState } from ‘./utils/helper-functions’

if (typeof AFRAME === 'undefined') {
  throw new Error('Component attempted to register before AFRAME was available.');
}

document.onload( () => {
  init()
})


function init() {
  renderMaterials()
  storeInitState()
}
```

O main.js organiza a lógica do projeto ao seu nível mais amplo e abstrato, garantindo que o código seja executado corretamente, evitando problemas de dependências. O ficheiro lida com o tratamento de erros de nível mais baixo, verificando a existência do módulo A-Frame e, em seguida, executa a função init(), que é usada principalmente para alterar os materiais iniciais de modelos específicos e para armazenar o estado inicial dos elementos que constituem a cena de A-Frame. A própria biblioteca trata da inicialização da sua cena e outros elementos constituintes.

index.html

No contexto de um projeto A-Frame, o arquivo HTML serve como o ponto de entrada principal da aplicação, é o arquivo HTML que define a estrutura e o conteúdo da cena A-Frame. A própria "a-scene" é o que lida com toda a biblioteca Three.js e a API WebVR, permitindo a criação de um ambiente de realidade virtual contínuo. Isto inclui:

- A configuração do canvas, renderizador e loop de renderização
- Câmera padrão e luzes
- A Adição do interface de utilizador para entrar na realidade virtual que chama a API WebVR
- A configuração de dispositivos WebXR através do sistema "webxr"
- Dentro da "a-scene", os *assets* são definidos através do *asset-management-system*, que permite o pré-carregamento e o armazenamento em cache dos *assets*.

```
<a-assets>
        <a-asset-item id="patient-model" src="/assets/patient-complex.glb">
        <a-asset-item id="bed" src="/assets/bed.glb"></a-asset-item>
        <a-asset-item id="cabinet-model" src="/assets/cabinet.glb"></a-asset-item>
         ...
```

Bem como os *mixins*, que permitem compor e reutilizar conjuntos de componentes

```
 <a-mixin
    id="tool"
    body="type: dynamic; shape: none"
    hoverable
    grabbable
    stretchable
    draggable
    droppable
    data-tool
    data-raycastable
    tool-hitbox
```

```
          rotation="0 0 0"
          raycaster="
            objects: #patient;
            showLine: false;
            direction: 0 -1 0;
            origin: 0 0 0;
            enabled: false"
        ></a-mixin>
```

O jogador é definido usando o *rig-container-model*, que agrupa a câmera e as mãos para facilitar o movimento e posicionamento. Neste ponto, são definidos atributos como rotação e posição, bem como vários componentes que serão explicados posteriormente.

```
  <a-entity
   id="rig"
   movement-controls="speed: 0.15"
   sound="src: #error-audio; volume: 0.4;">

   <a-entity
     id="leftHand"
     hand-controls=
       "hand: left;
       handModelStyle: highPoly;
       color: #00d1ce"
     mixin="hands">
   </a-entity>

   <a-entity
     id="rightHand"
     ...
   </a-entity>

   <a-camera
     id="camera"
     zoom-on-scroll
     ...
   </a-camera>
 </a-entity>
```

Após o jogador, todos os outros objetos, como o paciente, o equipamento médico e os elementos de UI, são inseridos na cena.

./utils

A pasta "utils" contém a maior parte da lógica do projeto em arquivos segmentados e modularizados. O arquivo index.js, como o nome indica, serve como um índice de todos os outros arquivos nesta pasta. Isto é usado para poder importar todo o código no main.js com apenas uma linha.

```
// index.js
export * from './aframe-components/look-at'
export * from './aframe-components/raycaster-listen'
export * from './aframe-components/gaze-interaction'
export * from './aframe-components/extend-arm'
export * from './aframe-components/object-description'
// ...
```

O ficheiro variables.js é usado para definir variáveis comumente usadas em todo o projeto, bem como variáveis globais, como objetos de estado.

```
// variables.js
// DOM
export const aScene = document.querySelector("a-scene");
export const threeScene = document.querySelector("a-scene").object3D
// ...
// STATES
export let userState = {
  tools: ['marker', 'scalpel', 'forceps', 'suture'],
  bodyParts: ['abdomin', 'breast', 'body'],
  leftHand: {
    holdingItem: false,
    heldItem: "",
  },
  rightHand: {
    holdingItem: false,
    heldItem: false,
  },
  controls: {},
};

export let gameState = {};
```

O ficheiro helper-functions.js é usado para funções personalizadas que são repetidas no projeto. Isto pode ser considerado como o framework deste projeto,

uma vez que contém as funções essenciais e comuns que são usadas extensivamente neste projeto.

```
// helper-function.js
export const enableRaycaster = (raycastingEntity, color, opacity) => {
  raycastingEntity.setAttribute("raycaster", "enabled", true);
  raycastingEntity.setAttribute("raycaster", "showLine", true);
  raycastingEntity.setAttribute("raycaster", "color", color);
  raycastingEntity.setAttribute("raycaster", "opacity", opacity);
};

export const disableRaycaster = () => {
  raycastingEntity.setAttribute("raycaster", "enabled", false);
  raycastingEntity.setAttribute("raycaster", "showLine", false);
};

export const changeMaterialbyName = (object3D, name, material) => {
  object3D.traverse((node) => {
    if (node.name == "name") node.material = material;
  });
};
// ...
```

O ficheiro controls.js configura todos os event listeners que são usados para controlar os estados dos botões. Isto atualiza a variável userState e permite uma validação condicional mais fácil.

```
// controls.js
import userState from "./variables";

addEventListener("triggerdown", () => {
  userState.controls.triggerdown = true;
  userState.controls.triggerup = false;
});

addEventListener("triggerup", () => {
  userState.controls.triggerdown = false;
  userState.controls.triggerup = true;
});
// ...
```

O ficheiro materials.js cria o objeto customMaterials, que contém todos os materiais personalizados que são acedidos e utilizados em todo o projeto. Isto funciona em conjunto com as funções auxiliares de material para permitir trocas de materiais e outras operações baseadas em materiais.

```
// materials.js
export const customMaterials = {
    fat: new THREE.MeshPhysicalMaterial({
      color: 0xca7f02
    }),

    appleGlass: new THREE.MeshPhysicalMaterial({
      metalness: 0,
      roughness: 1,
      envMapIntensity: 0.9,
      clearcoat: 1,
      transparent: true,
      transmission: 1,
      opacity: 0.8,
      reflectivity: 0.2,
      side: THREE.DoubleSide,
     })
}
```

voice-control.js implementa comandos de voz, como é o único código que funciona independentemente do A-Frame ou do three.js, é mantido no seu próprio ficheiro.

```
import { tools } from "./variables";

export const voiceControls = () => {
    // set up
    window.SpeechRecognition = window.SpeechRecognition || window.webkitSpeechRecognition;
    const recognition = new SpeechRecognition();
    recognition.interimResults = true;
    // ...
    recognition.start();
}
```

Componentes A-Frame

Na pasta "aframe-components" está o código de todos os componentes A-Frame. No padrão de Entity-Component-System (ECS), um componente é um pedaço reutilizável e modular de dados que conectamos a uma entidade para adicionar aparência, comportamento e/ou funcionalidade.

Desta forma, os componentes são aproximadamente análogos aos seletores CSS, pois ambos podem ser usados para alterar a aparência, funcionalidade e comportamento das entidades.

```
AFRAME.registerComponent('foo', {
  schema: {},
  init: function () {},
  update: function () {},
  tick: function () {},
  remove: function () {},
  pause: function () {},
  play: function () {}
});
```

Esta é a estrutura padrão de um componente no A-Frame. Este fornece um parâmetro "name" para nomear o componente e um *schema* para definir a forma dos dados a serem usados, ou seja, os parâmetros a serem utilizados ao atribuir o componente a uma entidade. Os métodos do ciclo de vida do componente (init, update, tick, etc.) são responsáveis pela manipulação real dos dados, cada um deles fornecendo um estado diferente no ciclo de vida do componente. Para os propósitos deste projeto, apenas os seguintes são relevantes:

- i**nit()** executado uma vez quando o componente é primeiro atribuido
- **tick()** executado uma vez por frame, sendo assim comparável ao requestAnimationFrame()

hands-interactions

Responsável por todas as interações de raycasting que não se relacionam com o paciente. Para garantir a legibilidade do código, o paciente recebe o seu próprio componente separado. Isto é feito para evitar que o componente já extenso fique muito grande e difícil de entender.

imports

```
import { userState, aScene } from "../variables";
import { enableRaycaster, disableRaycaster } from "../helper-functions";
import customMaterials from "../materials";
let hand;
```

As variáveis, materiais e funções auxiliares necessárias são importadas, e a variável "hand" é declarada para verificar em qual mão o código está a ser executado.

init()

```
  init: function () {
    this.el.addEventListener("raycaster-intersection", (event) => {
      this.raycaster = event.detail;
    });

    this.el.addEventListener("raycaster-intersection-cleared", (event) => {
      this.raycaster = null
      this.clearedIntersection = event.detail.clearedEls[0]
    });

    hand = userState[this.el.id];
  },
```

a função init() cria dois event listeners:

- **'raycaster-intersection'** é emitido na entidade que realiza o raycasting sempre que o raycaster está a intersectar com uma ou mais entidades. Retorna um objeto *event.detail* que contém todos os dados relevantes da interseção. Os resultados são armazenados na propriedade "raycaster" do componente para serem usados na função tick() do componente.

- **‘raycaster-intersection-cleared’** é emitido na entidade que realiza o raycasting sempre que o raycaster não está a intersectar com uma ou mais entidades. Quando esse evento é acionado, ele atribui a propriedade "raycaster" do componente como null e armazena as interseções desfeitas numanova propriedade: 'clearedIntersection'.

tick()

```
tick: function () {
  // return tool
  if (userState.controls.triggerdown && hand.holdingTool)
    this.returnTool();

  // validate intersection
  if (!this.raycaster) {
    // clear hover state
    let hitbox = this.clearedIntersection
    if (isTool(hitbox)) this.toolHoverOff(hitbox)
    return
  }

  let { intersection } = this.raycaster;
  // check if intersection is a tool
  let hitbox = intersection[0].object.el;
  if (isTool(hitbox)) {
    let tool = hitbox.parentElement

    // activate hover state
    this.toolHoverOn(tool, hitbox)
    this.hoverDescription(tool)
    if (userState.controls.triggerdown) {
      this.grabTool(tool)
    }
  }
},
```

A função tick() verifica a propriedade "raycaster" a cada frame e utiliza os dados de interseção para executar funções.

- Se a propriedade "raycaster" for nula, o componente verifica a última interseção desfeita ("clearedIntersection"), avalia se é uma ferramenta (tool), e caso seja, remove o *hover state* da entidade. Em seguida, a

função "tick()" continua as suas operações ou retorna as funções necessárias.

- Caso contrário, irá verificar se o objeto de intersecção é uma ferramenta utilizando a função isTool(). Nesse caso, irá adicionar o *hover-state* sobre a ferramenta) através da função toolHoverOn() e o elemento de descrição flutuante através da função hoverDescription(). Se o *trigger-button* for mantido pressionado, a ferramenta será agarrada através da função grabTool().
- Anterior a estas condições, é declarada uma condição para devolver a ferramenta, utilizando a função returnTool(). Isto é feito antes da lógica restante para permitir que o usuário alterne diretamente entre as ferramentas, se necessário.

`grabTool()`

```
grabTool(tool) {
   // set data
   hand.holdingTool = true;
   hand.heldTool = tool.id;
   disableRaycaster(this.el);

   // get world position of the hand
   let handWorldPosition = new THREE.Vector3();
   this.el.object3D.getWorldPosition(handWorldPosition);

   // animations
   // current position to hand
   tool.setAttribute("animation__grab", {
     property: "position",
     from: tool.object3D.position,
     to: handWorldPosition,
     dur: 500,
     easing: "easeInSine",
   });

   // rotation animation
   tool.setAttribute("animation__grab_rotate", {
     property: "rotation",
     to: "0 0 -180",
```

```
    dur: 500,
    easing: "easeInSine",
  });

  // on animation completion
  tool.addEventListener("animationcomplete__grab", () => {
    // add tool to hand
    this.el.object3D.add(tool.object3D);
    tumor.setAttribute("body", "type: static");

    // adjust tool positioning to fit in hand naturally
    tool.object3D.position.set(-0.01, -0.12, 0);
    tool.setAttribute("rotation", "5 5 5");
    enableRaycaster(tool, "white", 0.3);
  });
},
```

A função grabTool() permite que o utilizador agarre uma ferramenta

- primeiro, são definidos os dados cruciais para a lógica do componente.
- em seguida, é obtida a posição global da mão do usuário, que será usada nas animações que virão a seguir.
- São realizadas as animações utilizando a posição da mão para mover a ferramenta em direção a ela.
- Após a conclusão das animações, é acionado o event listener 'animationcomplete': a malha da ferramenta é adicionada à mão do utilizador , e sua posição e rotação são ajustadas para proporcionar uma aperto mais natural.

A função returnTool() opera com a mesma lógica, mas revertido

hoverDescription()

```
hoverDescription(tool) {
  if ( tool.getAttribute("hover") == null  ) {
    let line = document.createElement("a-entity");

    let boundingBox = new THREE.Box3().setFromObject(tool.object3D);
    let vector = new THREE.Vector3();
    let center = boundingBox.getCenter(vector);

    let lineStart = `${center.x} ${center.y + 0.05} ${center.z}`;
    let lineEnd = `${center.x + 0.08} ${center.y + 0.3} ${center.z}`;
    let lineParams = `start: ${lineStart}; end: ${lineEnd}; color: black`;
    line.setAttribute("line", lineParams);

    this.returntoolHTML(tool);

    let htmlPopOver = document.createElement("a-entity");
    htmlPopOver.setAttribute("html", `cursor:#cursor;html:#my-interface`);
    htmlPopOver.setAttribute("position", lineEnd);
    htmlPopOver.setAttribute("scale", "2 2 2");
    htmlPopOver.setAttribute("look-at", "#camera");
    aScene.insertAdjacenttool("afterbegin", line);
    aScene.insertAdjacenttool("afterbegin", htmlPopOver);
    tool.setAttribute("hover", "true");
  }
},
```

A função hoverDescription() utiliza a biblioteca 'aframe-html', que permite o uso de sintaxe HTML e CSS para criar elementos de interface do utilizador (UI) em A-Frame. Aqui é utilizada para criar as descrições das ferramentas que surgem quando o utilizador coloca o raycaster sobre uma ferramenta.

- cria uma linha utilizando a geometria do A-Frame e posiciona-a usando os dados de geometria da ferramenta.
- Em seguida, cria o elemento pop-up e utiliza o componente 'html' para inserir a sintaxe HTML e CSS.
- Anexa o componente 'look-at' para fazer com que o texto esteja sempre voltado para o utilizador.
- Por fim, chama a função returnToolHTML(), que altera o valor do texto do pop-up para o nome da ferramenta.

toolHoverOn() & toolHoverOff()

```
toolHoverOff(tool, hitbox) {
  // reset attributes
  tool.tool.setAttribute("scale", "1, 1, 1");
  tool.changeMaterialByMesh(hitbox, customMaterials.hitboxHoverOriginal)
  tool.changeMaterialByMesh(tool, customMaterials.toolHoverOriginal)
  tool.removeAttribute("animation");
  tool.removeAttribute("animation__scale");
  tool.setAttribute(
    "animation__scale2",
    "property: scale; dur 550;  to: 1 1 1; easing: linear"
  );
},

toolHoverOn(tool, hitbox) {
  // activate hover state materials
  changeMaterialByMesh(hitbox, customMaterials.hitboxHover);
  changeMaterialByMesh(tool, customMaterials.toolHover);

  // animation
  tool.setAttribute(
    "animation",
    "property: rotation; to: 0 360 0; loop: true; dur: 10000; easing: linear"
  );

  tool.setAttribute(
    "animation__scale",
    "property: scale; dur: 100; from: 1 1 1; to: 1.2 1.2 1.2; easing: linear"
  );
},
```

As funções toolHoverOn() e toolHoverOff() lidam com os efeitos de *hover* utilizando materiais personalizados e a função auxiliar changeMaterialByMesh() para alternar entre o material do estado *hover* e o material original.

patient-interaction

Esta componente lida com todas as interações que ocorrem com o paciente, isto inclui as mecânicas principais e mais complexas apresentadas nesta aplicação.

```
AFRAME.registerComponent("patient-parts", {
  schema: {},

  init: function () {
    let patient = this.el;
    this.painting = false
    this.el.addEventListener("model-loaded", () => {
      userState.bodyParts.forEach((meshName) => {
        patient.object3D.traverse((node) => {
          if (node.name === meshName) {
            this.setUpMesh(node);
            this.createCanvas(node);
          }
        });
      });

      this.el.addEventListener("raycaster-intersected", (evt) => {
        this.raycaster = evt.detail;
      });
      this.el.addEventListener("raycaster-intersected-cleared", () => {
        this.raycaster = null;
      });

      initComplete = true
    });
  },
```

A função init lida com a configuração inicial dos modelos de malha armazenando variáveis e dados para uso posterior no código.

- O *event listener* 'model-loaded' é acionado quando o modelo é totalmente carregado e garante que o código contido nele não apresente erros devido a problemas de carregamento.
- O modelo do paciente é um grupo de malhas (mesh), o que significa que ele é composto por várias malhas: a mama, o abdômen, o corpo e outras malhas que não são relevantes. A matriz bodyParts: ['breast', 'abdomen', 'body'] é usada para iterar facilmente pelos *child-elements* do modelo e identificar

corretamente cada parte do corpo, o que por sua vez é utilizado para executar as funções setUpMesh() e createCanvas().

- *event listeners* de raycasting são criados para acompanhar qualquer interseção do raycaster com o modelo do paciente.
- A variável initComplete é definida como *true* para permitir que a função tick seja executada somente após a conclusão da função init. Isto é feito para evitar problemas de dependência que podem surgir na função tick se o modelo ainda não tiver sido configurado no momento da sua execução.

createCanvas()

```
createCanvas(node) {
  let image = node.material.map.image;
  userState[node.name].canvas = document.createElement("canvas");
  userState[node.name].context = userState[node.name].canvas.getContext("2d");
  userState[node.name].canvas.id = node.name;
  userState[node.name].canvas.height = image.height;
  userState[node.name].canvas.width = image.width;
  userState[node.name].context.drawImage(image, 0, 0);
  userState[node.name].texture = new THREE.CanvasTexture(userState[node.name].canvas);
  node.material.map.dispose();
  node.material.map = texture;
  node.material.transparent = false;
  node.material.alphaTest = 0.1;
  node.material.map.flipY = false;
  node.material.needsUpdate = true;
  aScene.appendChild(canvas);
  this.createCastCanvas(node);
}
```

Para possibilitar o mecanismo de desenho no paciente, é usado um canvas como textura do modelo[15]. Isto permite a atualização dinâmica da textura e o uso dos métodos de desenho do canvas. Como o modelo é um grupo de malhas, um canvas tem de ser criado para cada uma das partes do corpo para que eles não compartilhem o mesmo canvas, o que resultaria em pinturas duplicadas em todas as partes do corpo.

tick()

```
tick: function () {
  if (initComplete) {
    // validate intersection
    if (!this.raycaster) return;
    let intersection = this.raycaster.getIntersection(this.el);

    // cast raycaster target
    if (intersection.distance < 5 && userState.rightHand.holdingItem)
      this.castTarget(intersection);

    // tool mechanics
    if (intersection.distance < 0.2 && userState.controls.aButtonDown) {
      switch (userState.rightHand.heldItem) {
        case "marker":
          this.draw(intersection);
          break;
        case "scalpel":
          this.incision(intersection);
          break;
        case "forceps":
          if (intersection.object.name !== "body") this.grab(intersection);
          break;
        case "suture": this.stitch()
      }
    }

    if (userState.controls.aButtonUp && heldBodyPart !== null)
      this.drop(userState[heldBodyPart]);

    // raycaster settings
    if (intersection.distance > 0.2)
      enableRaycaster(this.currentTool, "white", 0.3);
  }
},
```

A função tick() determina quais os mecanismos a executar com base na distância (em metros) da interseção do raycaster e na ferramenta utilizada.

draw()

```
draw(intersection) {
   let bodyPart = intersection.object;
   let uv = intersection.uv;

   // set up brush
   let markerPointX = Math.floor(uv.x * bodyPart.material.map.image.width);
   let markerPointY = Math.floor(uv.y * bodyPart.material.map.image.height);
   let markerPointHeight = 5;
   let markerPointWidth = 5;
   let offsetX = markerPointWidth / 2;
   let offsetY = markerPointHeight / 2;
   userState[bodyPart.name].context.globalCompositeOperation = "darken";
   userState[bodyPart.name].context.globalAlpha = 0.25;

  // draw
   userState[bodyPart.name].context.fillRect(
     markerPointX - offsetX,
     markerPointY - offsetY,
     markerPointWidth,
     markerPointHeight
   );

   // update
   bodyPart.material.map.flipY = false;
   bodyPart.material.map.needsUpdate = true;
 },
```

- utilizando o ponto UV dos dados de interseção e as dimensões do canvas, podemos definir a posição do pincel, juntamente com o tamanho e formato, usando o método drawRect(). O pincel do marcador, em particular, possui algumas configurações adicionais de material para simular a transparência e o efeito de sobreposição do marcador.
- O pincel é desenhado no canvas e posteriormente atualizado no renderizador.

castTarget()

```
castTarget(intersection) {
  // ... setup variables
  // refresh canvas
  userState[bodyPart.name].castContext.clearRect(
    0,
    0,
    userState[bodyPart.name].castCanvas.width,
    userState[bodyPart.name].castCanvas.height
  );

  // draw on the result canvas
  userState[bodyPart.name].resultContext.drawImage(userState[bodyPart.name].canvas, 0, 0);
  userState[bodyPart.name].resultContext.drawImage(userState[bodyPart.name].castCanvas, 0, 0);

  // ... render result canvas
},
```

Além do mecanismo de desenho do marcador, todas as ferramentas são capazes de projetar um alvo no paciente. Enquanto que o efeito pretendido do marcador é que permanece no corpo, o alvo tem que ser atualizado a cada frame, pelo que não é possível apenas desenhar o alvo usando o canvas criado anteriormente. Para isto, um castCanvas também tem que ser criado; este canvas não carrega inicialmente a textura do modelo e, em vez disso, é desenhado diretamente, o que significa que ele é essencialmente transparente onde não é desenhado. Além disso, o método clearRect() é chamado a cada quadro para permitir apenas um traço do pincel por frame. Em seguida, para exibir ambos os canvases, eles são desenhados num terceiro canvas, chamado resultCanvas, que é usado para renderizar a textura final do canvas.

- o canvas é primeiro atualizado usando clearRect()
- o pincel desenha sobre o canvas
- O resultCanvas, em seguida, desenha primeiro o canvas original, seguido pelo castCanvas e, finalmente, é renderizado como a textura do modelo.

incision()

```
incision(intersection) {
  let bodyPart = intersection.object;
  if (bodyPart.name == "body") return bodyPart.name = null;
  brush1 = new Brush(bodyPart.geometry.clone(), bodyPart.material);
  brush1.updateMatrixWorld();

  //setup brush2
  bodyPart.worldToLocal(intersection.point);
  brush2.position.copy(intersection.point);
  brush2.quaternion.copy(bodyPart.quaternion);
  brush2.updateMatrixWorld();

  csgEvaluator.evaluate(brush1, brush2, SUBTRACTION, bodyPart);
},
```

A função incision() utiliza a biblioteca 'three-bvh-csg'. Esta biblioteca é construída sobre o Three.JS e permite o uso de geometria sólida construtiva, o que possibilita operações booleanas entre duas geometrias do Three.JS. Nesta aplicação, ela é usada para simular o efeito de corte do bisturi.

- O brush1 é definido usando a geometria e o material da parte do corpo intersectada.
- O brush2 é uma geometria de caixa previamente definida e um material personalizado que simula gordura.
- A posição do brush2 é determinada usando o ponto de interseção e é traduzida para a posição local da parte do corpo.
- A função evaluate() recebe ambos os pincéis, o tipo de operação (SUBTRACTION, neste caso) e o objeto resultante, que neste caso é a própria parte do corpo.

grab() & drop()

As funções grab() e drop(), que são usadas pelas pinças, funcionam de maneira semelhante às funções grabTool() e returnTool() definidas no componente anterior, com algumas lógicas extras para lidar com os eventos "on-rails" da operação.

stitch()

```
 stitch(intersection) {
let face = intersection.face;

let vA = new THREE.Vector3();
let vB = new THREE.Vector3();
let vC = new THREE.Vector3();

let geometry = intersection.object.geometry;
let position = geometry.attributes.position;

vA.fromBufferAttribute(position, face.a);
vB.fromBufferAttribute(position, face.b);
vC.fromBufferAttribute(position, face.c);

// Calculate the distance between two Vector3 points
function distanceBetweenPoints(pointA, pointB) {
  return pointA.distanceTo(pointB);
}

// Calculate distances from the intersection point to each vertex
let distanceToA = distanceBetweenPoints(vA, intersection.point);
let distanceToB = distanceBetweenPoints(vB, intersection.point);
let distanceToC = distanceBetweenPoints(vC, intersection.point);

// Find the minimum distance among the three vertices
let minDistance = Math.min(distanceToA, distanceToB, distanceToC);

// Check which vertex is closest to the intersection point
let closestVertex;
let closestVertexIndex; // Variable to store the index of the closest vertex
if (minDistance === distanceToA) {
  closestVertex = vA;
  closestVertexIndex = face.a;
} else if (minDistance === distanceToB) {
  closestVertex = vB;
  closestVertexIndex = face.b;
} else {
  closestVertex = vC;
  closestVertexIndex = face.c;
}

// Modify the actual mesh's geometry
let mesh = intersection.object;
let g = mesh.geometry;
let p = g.attributes.position;

let moveAmount;
if (userState.controls.bButton) moveAmount = 0.001;
if (userState.controls.aButton) moveAmount = -0.001;
closestVertex.y += moveAmount;

// Update the vertex in the position attribute with the modified closestVertex
```

```
    p.setXYZ(
      closestVertexIndex,
      closestVertex.x,
      closestVertex.y,
      closestVertex.z
    );

    // Mark the attribute as needing an update
    p.needsUpdate = true;
  },
```

A função stitch() permite manipular os vértices[15] do plano de interseção, permitindo fechar a incisão do paciente.

- calcula as distâncias entre o ponto de interseção e cada um dos três vértices do plano de interseção.
- Em seguida, identifica o vértice mais próximo entre os três.
- Com base na *input* do utilizador, decide a direção do estiramento (para cima ou para baixo ao longo do eixo Y).
- O código modifica a coordenada Y do vértice mais próximo para esticar o objeto na direção escolhida.
- Por fim, a função atualiza a aparência do objeto para refletir as alterações feitas na malha

Outros componentes

Além destes dois componentes, o projeto inclui os seguintes componentes:

- **audio-play** permite uma manipulação mais fácil do áudio, tornando mais simples criar eventos sonoros, adicionar sons aos elementos e modificar as configurações sonoras atuais.
- **look-at** oferece uma maneira fácil de fazer com que qualquer entidade olhe para outra, é utilizado no *hover-state* das ferramentas, fazendo com que as descrições de texto que aparecem estejam sempre direcionadas para o utilizador, independentemente de onde estejam localizadas.
- **tool-hitbox** cria dinamicamente as áreas de colisão para as ferramentas.

Design

Visão Geral

Devido à escala da seção de programação deste projeto, os componentes de interface do utilizador e experiência do utilizador não foram totalmente desenvolvidos. No entanto, considerações em diferentes aspetos nestes campos foram levados em conta e incorporadas intermitentemente ao projeto.

Interface de Utilizador

figura 2: VisionOs UI

O design do UI foi inspirado pela recente demonstração da VisionOS da Apple, A linguagem visual do sistema operativo é uma clara transição da corrente estilística *glassmorphism* popularizado pela propria Apple para o espaço tridimensional. Destaca-se pelo uso do *glass material*, uma material translúcido e nebuloso, que segundo a Apple[], permite mais facilmente enquadrar designs de UI

num espaço tridimensional dado o modo como interage com a luz, conferindo um ambiente mais natural para o utilizador.

figura 3: recriação visionOS

Este efeito foi obtido através da criação de um custom material no Three.JS em conjunção com custom shaders que simulam vidro via o webGL

```
// Vertex Shader
const vertexShader = `
    varying vec3 vNormal;
    void main() {
        vNormal = normal;
        gl_Position = projectionMatrix * modelViewMatrix * vec4(position, 1.0);
    }
`;

// ...

appleGlass: new THREE.MeshPhysicalMaterial({
    vertexShader: vertexShader,
    fragmentShader: fragmentShader,
    metalness: 0,
    roughness: 1,
    envMapIntensity: 0.9,
    clearcoat: 1,
    transparent: true,
    transmission: 1,
    opacity: 0.8,
    reflectivity: 0.2,
    side: THREE.DoubleSide,  }
```

User Experience

Novamente, a UX foi desenvolvida tendo em consideração diretrizes estabelecidas pela Apple[14], sendo as mais facilmente identificáveis no projeto as seguintes:

- Evita encorajar as pessoas a se moverem demais enquanto estão em uma experiência totalmente imersiva.
- Evita a exibição de movimentos que sejam avassaladores, bruscos, muito rápidos ou que faltem um quadro de referência estacionário.
- Exibe o conteúdo dentro do campo de visão da pessoa, posicionando-o em relação à sua cabeça.
- Evita colocar o conteúdo em locais onde as pessoas precisam de virar a cabeça ou mudar de posição para interagir
- Prioriza o conforto do utilizador.

Para poder criar displays de UI em espaços tridimensionais que possam ser lidas a qualquer distância, ia ser implementado o sistema *Distance-Independent Millimeters* da Google[16] que introduz unidades angulares que permitem manter uma escala consistente em elementos de UI, independente da distância entre eles no eixo Z

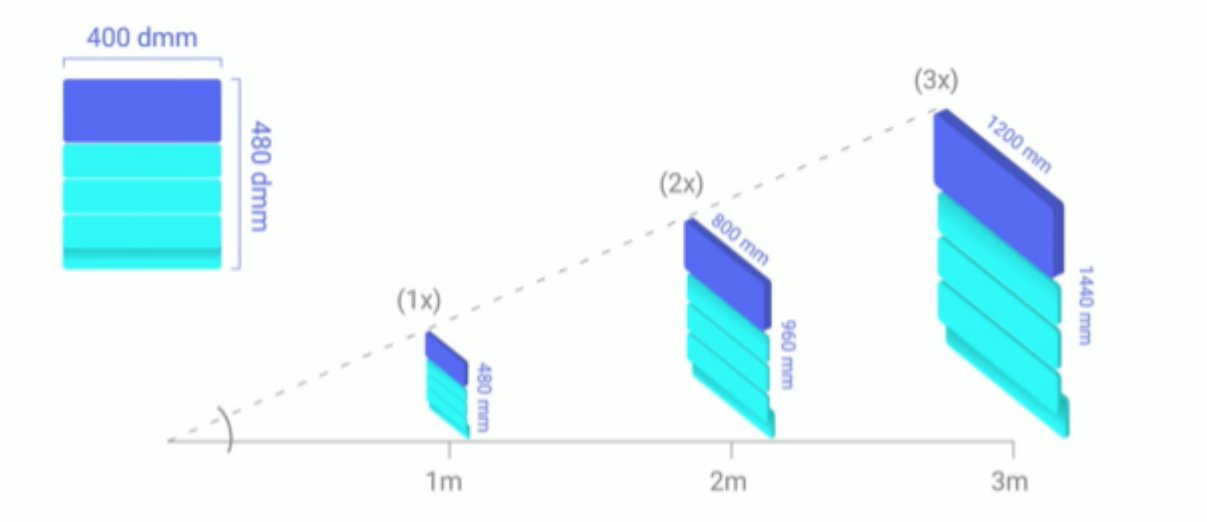

figura 4: dmm

Audio

Visão Geral

O elemento auditivo é criado de forma a complementar a experiência do utilizador. A assistente virtual guia o utilizador pelos procedimentos da operação, de uma forma didática, enquanto que os instrumentos médicos respondem à interação direta, e as *audio cues* reafirmam o sucesso das tarefas concluídas. Simultaneamente, o ambiente sonoro é aplicado de forma a assistir na imersão do utilizador, fazendo uso de efeitos sonoros naturais e envolventes, e de uma banda sonora subtil.

Processo Técnico

A manipulação do áudio foi feita pelo FL Studio, um programa de edição de som. Faz-se o uso de *field recordings* e instrumentos virtuais, aos quais são aplicados efeitos como compressão, equalização e reverberação.

O efeito de voz da assistente virtual foi obtido através da manipulação do tom que é alterado de forma semelhante ao efeito de *autotune* para um resultado robótico.

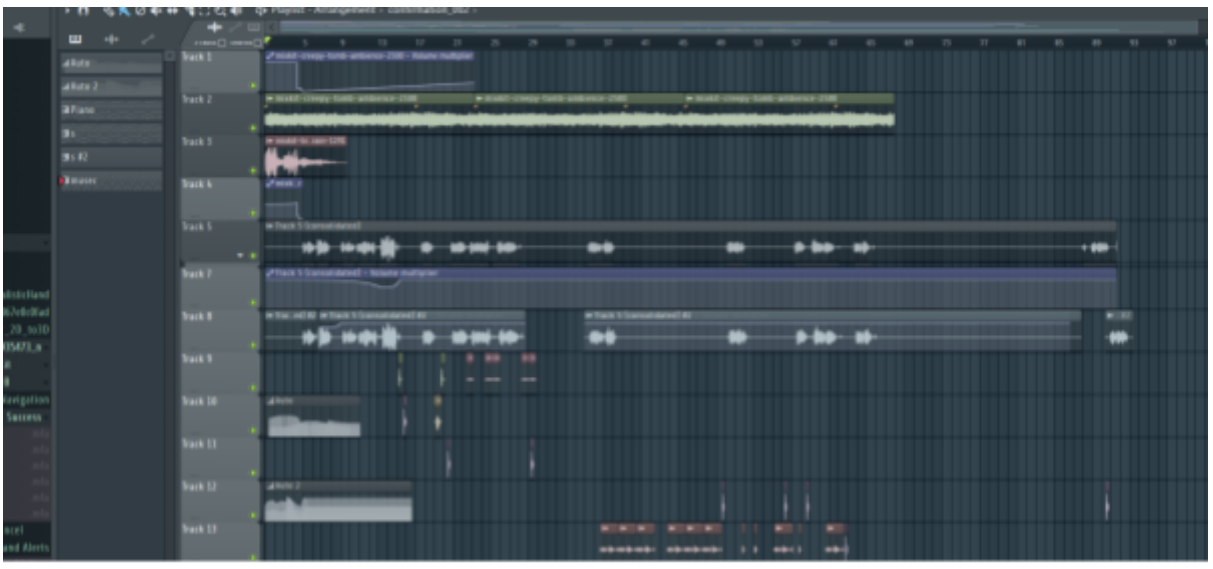

figura 5: FL Studio

Modelação 3D

Visão Geral

Há várias facetas de modelação 3D neste projeto, nomeadamente a adaptação de modelos pré-existentes para um formato compatível, a edição de modelos para uma aplicação mais específica; correções de elementos conflituosos e estandardização dos atributos geométricos dos modelos.

Para realizar uma das mecânicas principais da operação, foi usado o Blender para modificar a malha do modelo, criando um retalho na zona abdominal, e criando uma operação booleana com o modelo nesta zona e no peito, obtendo deste modo três malhas separadas: a do modelo original com os retalhos já subtraídos, o retalho abdominal e o retalho peitoral. Esta operação vai permitir, que no momento da operação o retalho abdominal caiba perfeitamente na cavidade do peito.

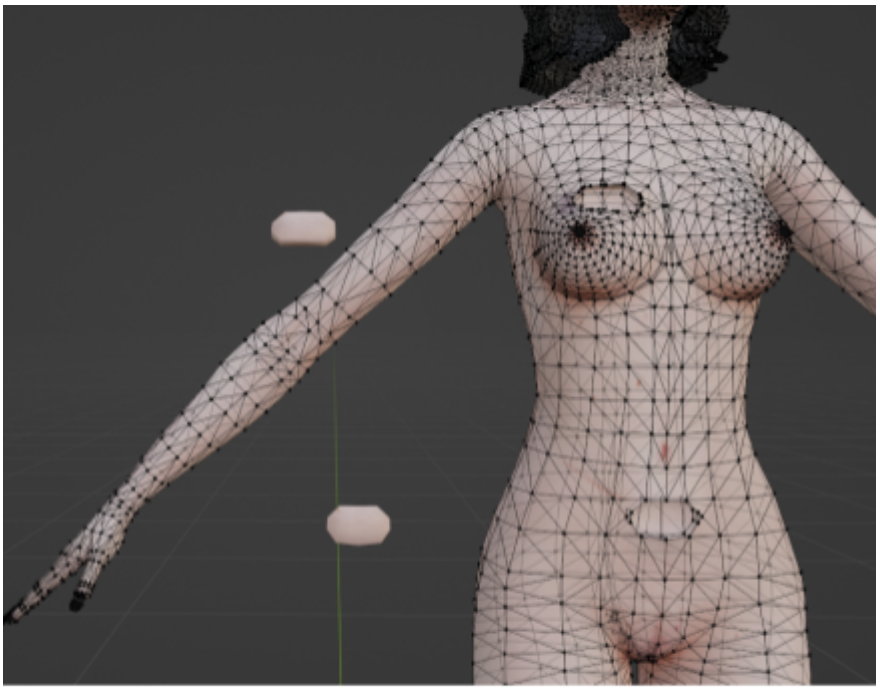

figura 6: modelo paciente

Bibliografia

[0] **A-Frame** - https://aframe.io/ - Biblioteca JavaScript de código aberto que facilita a criação de experiências de realidade virtual (VR) e realidade aumentada (AR) para a web, criado em cima da framework THREE.js

[1] **THREE.js** - https://threejs.org/ - biblioteca JavaScript de código aberto que facilita a criação e renderização de gráficos 3D interativos no navegador. Fornece uma abstração de alto nível para trabalhar com WebGL, que é uma API de baixo nível para renderização de gráficos 3D no navegador.

[2] **StackOverflow** - https://stackoverflow.com/ - website de perguntas e respostas para programadores e desenvolvedores de software

[3] **Github** - https://github.com/ - plataforma de hospedagem e colaboração de código-fonte para desenvolvimento de software

[4] **Discord** - https://discord.com/ - plataforma de comunicação projetada para criar comunidades online e facilitar a interação entre os membros por meio de bate-papo por texto, voz e vídeo

[5] **Slack** - https://slack.com/ - plataforma de comunicação e colaboração em equipe que permite que grupos de pessoas trabalhem juntas de forma eficiente

[6] **ChatGPT** - https://chat.openai.com/ - uma implementação específica do modelo de linguagem GPT (Transformador de Pré-Treinamento Gerado) desenvolvido pela OpenAI. É uma versão adaptada para conversação, permitindo que os usuários interajam e tenham diálogos com o modelo de linguagem.

[7] **Design Centrado no Utilizador** - User-Centered Design | Abras, C., Maloney-Krichmar, D., Preece, J. (2004) User-Centered Design. In Bainbridge, W. Encyclopedia of Human-Computer Interaction. Thousand Oaks: Sage Publications. (in press) - abordagem de design que coloca as necessidades, desejos e habilidades dos usuários no centro do processo de design

[8] **UX Stack Exchange** - https://ux.stackexchange.com/ - um fórum online dedicado a questões relacionadas à experiência do usuário

[9] **/r/webdev -** https://www.reddit.com/r/webdev/ - uma comunidade online dentro da plataforma Reddit, dedicada ao desenvolvimento web.

[10] **Modelo SAM** - https://dli.kennesaw.edu/resources/idmodels/sam.php - Um framework iterativo para desenvolver programas de treinamento envolventes rapidamente

[11] **Cirurgia DEIAP**
https://www.breastcancer.org/treatment/surgery/breast-reconstruction/types/autologous-flap/diep

[12] **Entity-Component Arquitecture**
https://aframe.io/docs/1.4.0/introduction/entity-component-system.html

[13] **Vite -** https://vitejs.dev/

[14] **Apple UI / UX Guidelines**-
https://developer.apple.com/design/human-interface-guidelines/designing-for-visionos - Diretrizes para developers em spatial design

[15] **Algoritmo vertice mais proximo**
https://stackoverflow.com/questions/35116060/how-do-i-calculate-which-vertice-is-the-closest-to-a-3d-point

[16] **Usar canvas como textura**
https://dustinpfister.github.io/2018/04/17/threejs-canvas-texture/

[17] Igarashi, Takeo. *Adaptive Unwrapping for Interactive Texture Painting*. 2001, http://www-ui.is.s.u-tokyo.ac.jp/~takeo/papers/i3dg2001.pdf.

Three.JS glass material rendering
https://tympanus.net/codrops/2021/10/27/creating-the-effect-of-transparent-glass-and-plastic-in-three-js/

mesh splitting
https://github.com/coolvision/three-bvh-csg/blob/iterative-dev/examples/iterative.js