UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

ESCOLA TÉCNICA SUPERIOR DE ENXEÑARÍA

Interacción natural mediante Leap Motion

Edición de modelos en 3D

Autor:
Pablo Dobarro Peña

Tutor:
Julián Flores González

Grao en Enxeñaría Informática

Febreiro 2018

Traballo de Fin de Grao presentado na Escola Técnica Superior de Enxeñaría da Universidade de Santiago de Compostela para a obtención do Grao en Enxeñaría Informática

D. Julián Flores González, Profesor do Departamento de Electrónica e Computación da Universidade de Santiago de Compostela,

INFORMA:

Que a presente memoria, titulada *Interacción natural mediante Leap Motion: Edición de modelos en 3D*, presentada por **D. Pablo Dobarro Peña** para superar os créditos correspondentes ao Traballo de Fin de Grao da titulación de Grao en Enxeñaría Informática, realizouse baixo miña dirección no Departamento de Electrónica e Computación da Universidade de Santiago de Compostela.

E para que así conste aos efectos oportunos, expiden o presente informe en Santiago de Compostela, a 6/2/2018:

O director, O alumno,

Índice general

Índice de figuras

Índice de cuadros

Glosario

- **Bases**: este documento se referirá a bases como el conjunto de tres vectores, todos ortogonales entre sí.
- **Escena**: Conjunto de objetos en un espacio 3D cuya situación en el mismo se define mediante un origen y unas bases ortonormales.
- **Espacio global**: Espacio definido mediante las bases de la escena a la que pertenece el objeto.
- **Espacio local**: Espacio definido mediante las bases del objeto.
- **Gizmo**: Representación visual en una interfaz gráfica como ayuda a la hora de realizar una edición. Suelen tener forma de flecha y permiten su manipulación con el ratón, arrastrándolos o moviéndolos.
- **HMD**: Siglas de Head Mounted Display. Casco de realidad virtual.
- **Malla**: Conjunto de vértices y caras que definen un modelo en 3D
- **Motor**: Conjunto de herramientas de desarrollo que generalmente incluye un renderizador, un sistema de físicas y elementos de interfaz gráfica para facilitar la programación de aplicaciones interactivas.
- **Nodo**: En Godot, objeto con las propiedades y métodos necesarios para pertenecer al árbol del programa y poder ejecutarse dentro del motor.
- **Objeto**: Todo elemento que puede pertenecer a una escena en 3D.
- **Proyección**: Obtención de las coordenadas 2D de un punto situado en un espacio 3D.
- **Rig**: Conjunto de huesos con diferentes limitaciones en su movimiento que se aplican a una malla 3D para poder deformarla y así animar un personaje.
- **Renderizador**: Software que permite visualizar elementos por pantalla.
- **Viewport**: Parte de la interfaz de un programa de edición 3D en la que se visualizan los objetos que se están editando.

Capítulo 1

Introdución

La interacción natural entre hombre y máquina ha tenido un especial interés en los últimos años debido a la aparición de nuevos dispositivos tales como Kinect, Wiimote o el mismo Leap Motion. El Leap Motion es un dispositivo que permite la interacción con el ordenador mediante las manos sin ningún tipo de marcador. El propio sistema realiza el seguimiento de las manos del usuario generando un modelo de las mismas y proporcionando información de las diferentes articulaciones que la componen.

En este trabajo se pretende explorar las posibilidades de uso de este dispositivo en un espacio 3D para la edición y visualización de modelos complejos mediante interacción natural.

1.1. Objetivos

El objetivo principal de este trabajo es realizar una aplicación que permita la visualización y edición de modelos en 3D mediante interacción natural, para lo cual se utilizará el dispositivo Leap Motion. Será necesario:

- Estudiar el funcionamiento del sensor Leap Motion, las posibilidades de su SDK y su integración en aplicaciones informáticas.
- Estudiar diferentes motores gráficos atendiendo a su rendimiento con mallas dinámicas y compatibilidad con Leap Motion. Implementar un renderizador 3D si fuera necesario.
- Implementar una capa de abstracción sobre el SDK de Leap Motion para facilitar el desarrollo de una herramienta de modelado 3D.
- Diseñar de los paradigmas e interfaz de usuario atendiendo a los requisitos de la aplicación y a las formas de interacción que requiera.

- Implementar un sistema de importación y exportación para poder cargar archivos en el programa.

- Implementar diferentes algoritmos de edición de una malla 3D.

1.2. Estado del arte

La interacción persona-ordenador es un campo de investigación que se centra en estudiar la relación entre los seres humanos y la tecnología. Para ello, intenta entender las reglas que un humano podría usar para comunicarse con un ordenador y en base a estas diseña nuevos paradigmas de interacción. Estos nuevos modelos de interacción pueden tener un gran rango de aplicaciones, ya sean industriales, educativas o como entretenimiento.

En los últimos años la industria del videojuego, buscando experiencias más interactivas, ha desarrollado nuevas tecnologías que permiten una interacción más natural que un mando o un teclado y ratón. Estos dispositivos, como el Kinet o el WiiMote permiten reconocer los movimientos del cuerpo y gestos del jugador integrándolos en el juego, lo que aumenta la sensación de inmersión.

Todas estas tecnologías e ideas se han llevado más allá del campo de los videojuegos, como por ejemplo a la educación o a la industria creativa, dando lugar a la aparición de diversos dispositivos que permiten una interacción natural con el software.

Un ejemplo es el programa TiltBrush de Google [7] para HTC Vive [6]. Este software es capaz de mostrar en realidad virtual los trazos de pintura que se realizan con los mandos del HMD (dispositivo que se coloca en la cabeza con una pantalla en frente de cada ojo). Esto permite una visualización en 3D real del modelo que se está realizando, proporcionando al artista la sensación de tener un pincel real en su mano y dándole mayor control y expresividad (Figura 1.1)

Otro tipo de dispositivo diseñado para una interacción natural es el ratón 3D fabricado por 3DConnexion. Este hardware permite el control del viewport de muchas aplicaciones 3D tradicionales con 6 ejes de movimientos simultáneos. Esto posibilita el movimiento de la cámara por la escena de una forma más precisa y directa que mediante un mecanismo de entrada de 2 ejes como un ratón un una tableta gráfica.

El sensor Leap Motion, capaz de detectar el movimiento de las manos sin necesidad de tocarlo, ha conseguido bastante notoriedad dentro del campo de la interacción natural debido a su facilidad de desarrollo y a la gran cantidad de software disponible compatible con él. El más reciente es el project Orion[9], un intento de acoplar el Leap Motion a un HMD para integrar las manos del usuario en el espacio de realidad virtual y así poder interactuar con los objetos que contiene. También existe software disponible para este sensor de carácter más tradicional, como la demo que incluye en su proceso de configuración. Esta

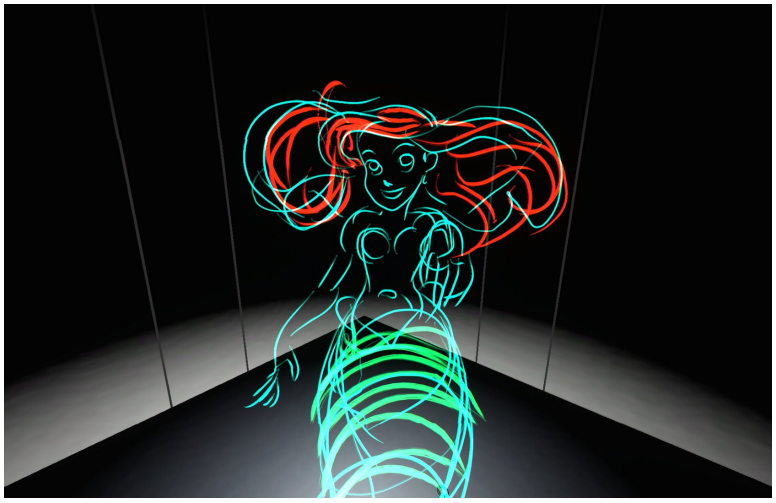

Figura 1.1: Software Tiltbrush para HTC Vive

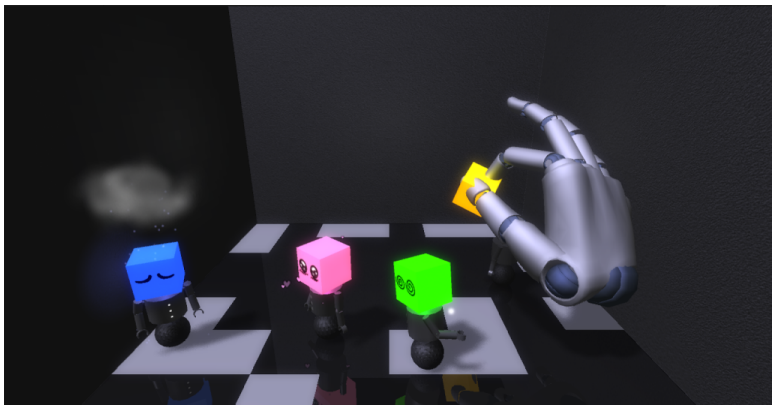

Figura 1.2: Demo de interacción con un espacio 3D con Leap Motion

Figura 1.3: Dog of Wisdom, animado en tiempo real

permite interactuar con unos muñecos mediante una representación de la mano del usuario como se muestra en la figura 1.2. La principal finalidad de este software es familiarizar al usuario con el funcionamiento del sensor de forma que pueda coordinar mejor sus movimientos para interactuar con un entorno 3D.

Otro tipo de interacción con la que se ha experimentado usando el Leap Motion se basa en controlar directamente los huesos de un rig de un personaje animado para generar animaciones en tiempo real (figura1.3). Los movimientos de los personajes del corto Dog of Wisdom están controlados mediante el Leap Motion como si fueran marionetas, acelerando de forma considerable el proceso de animación. Esta técnica también puede aplicarse para animar las manos de personajes 3D copiando directamente los movimientos de la manos del usuario [10].

A pesar de todas estas alternativas, muchas de ellas son muy poco útiles en un entorno de producción en 3D real, donde no se prioriza una interacción natural con el software, a pesar de que este modo de interacción podría integrarse para acelerar procesos como la animación o el prototipado. Salvo el uso de un ratón 3D en ocasiones puntuales, el resto de hardware es poco funcional, contiene demasiados errores y una precisión insuficiente como para su uso en proyectos serios, por lo que se limita a su uso como juguetes o demos técnicas.

1.3. El sensor Leap Motion

Leap Motion, Inc. es una compañía americana que manufactura y comercializa el sensor Leap Motion [8] (figura 1.4), un dispositivo que permite usar las manos como método de entrada de un ordenador, sin necesidad de tocarlo. En 2016, esta empresa publicó un software para integrar este dispositivo con aplicaciones de realidad virtual.

Figura 1.4: Sensor Leap Motion

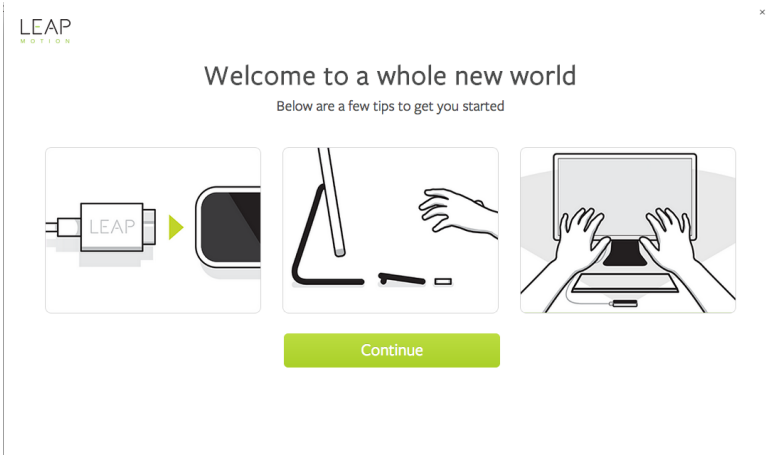

Figura 1.5: Esquema del proceso de puesta a punto de sensor Leap Motion

El sensor Leap Motion es un pequeño dispositivo USB que fue originalmente diseñado para colocarse en la mesa del escritorio, delante del monitor, orientado hacia arriba. Este dispositivo permite la interacción con el ordenador mediante la utilización de las manos como entrada sin la necesidad de interacción directa (figura 1.5). Su funcionamiento está basado en dos cámaras infrarrojas junto a unos LEDs para capturar 200 imágenes por segundo de las manos del usuario. Estas imágenes se mandan al ordenador por el cable USB y este ejecuta una serie de algoritmos matemáticos no liberados al público para determinar la posición de las manos. El software que ejecuta estos algoritmos cuenta con un visualizador de diagnóstico para comprobar que el sensor está funcionando correctamente.

1.4. Aplicaciones

Un software de edición y visualización de modelos en 3D puede beneficiarse del uso del Leap Motion y un paradigma de interacción natural obteniendo información espacial en tiempo real del usuario, imposible de conseguir mediante el uso del teclado y ratón. En base a estas características el software que se desarrollará en este proyecto podrá tener aplicaciones en las siguientes áreas:

- Visualización de modelos en 3D en entornos en los que el usuario no pueda estar en contacto directo con un ordenador, como en un entorno médico o en salas esterilizadas de laboratorios [12].
- Prototipado de modelos en 3D más rápido, permitiendo manipular objetos en todos sus ejes al mismo tiempo
- Uso adecuado para personas con movilidad reducida, permitiendo realizar más acciones con una sola mano respecto a un software de edición 3D tradicional [13].
- Uso en la industria del entretenimiento para crear juegos o simuladores
- Educación, ya que proporciona un sistema de interacción mucho más sencilla e intuitiva para un niño que el uso de un teclado y ratón [11].

Capítulo 2

Análisis

En este capítulo se establecerán las características del software a desarrollar identificando sus requisitos funcionales y no funcionales. También se tratarán aspectos relacionados con la gestión del proyecto y su planificación, como sus riesgos o su alcance. Por último, se hará un estudio y comparativa de las tecnologías disponibles para empezar a diseñar e implementar el proyecto, procurando escoger la opción que mejor se adapte a las características de este.

Contents

2.1. Requisitos funcionales

A continuación se recogerán los requisitos funcionales del producto software que generará el proyecto. Dichos requisitos definen la funcionalidad de la aplicación final, por lo que el objeto es documentarlos de la forma más precisa posible para que el producto satisfaga las necesidades del cliente.

En software de este proyecto está pensado para que un único usuario lo use al mismo tiempo, siendo irrelevante el rol de dicho usuario. Por lo tanto, siempre que en el documento se refiera a 'usuario', se está refiriendo al único actor que interactuará con el software. Sus casos de uso también vendrán dados por los requisitos funcionales listados a continuación.

Para definir los requisitos funcionales usarán los siguientes campos:

RF## - Requisito funcional

- Nombre: Nombre del requisitos funcional
- Descripción: Descripción del requisito funcional
- Prioridad: Prioridad para organizar la implementación y diseño de los requisitos funcionales.
- Precondiciones: Descripción del estado del sistema antes de la ejecución de la funcionalidad.
- Postcondiciones: Descripción del estado del sistema una vez se ejecuta la funcionalidad.

La prioridad vendrá definida por:

- Alta: Es esencial que el requisito funcional esté implementado cuanto antes. El proyecto fracasará en caso de no conseguir implementarlo
- Media: El requisito ha de implementarse, pero no pone en peligro la viabilidad del proyecto
- Baja: El requisito es deseable pero no esencial para el proyecto.

Los requisitos funcionales del software a desarrollar son los siguiente:

RF1 - Requisito funcional

- Nombre: Interpretar posición y rotación de la mano
- Descripción: El programa deberá ser capaz de interpretar la posición y rotación de la mano cuando se encuentra sobre el sensor para poder responder adecuadamente.
- Prioridad: Alta
- Precondiciones: El sensor Leap Motion deberá estar conectado al ordenador con el driver ejecutándose y una mano del usuario deberá colocarse sobre el sensor.
- Postcondiciones: El programa deberá reconocer la posición y rotación de la mano del usuario respecto al sensor.

RF2 - Requisito funcional

- Nombre: Interpretar estado de la mano
- Descripción: El programa deberá ser capaz de diferenciar una mano abierta de una cerrada cuando están colocadas sobre el sensor.
- Prioridad: Alta
- Precondiciones: El sensor Leap Motion deberá estar conectado al ordenador con el driver ejecutándose y una mano del usuario deberá colocarse sobre el sensor.
- Postcondiciones: El programa deberá interpretar la mano como abierta o cerrada, respondiendo adecuadamente según el contexto en el que se encuentre.

RF3 - Requisito funcional

- Nombre: Renderizar objeto
- Descripción: Una malla almacenada en la memoria del programa deberá ser visible por pantalla.
- Prioridad: Alta
- Precondiciones: Una malla formada por triángulos deberá estar almacenada en la estructura de datos interna del programa.
- Postcondiciones: La malla deberá ser mostrada por pantalla, en perspectiva y con un shading básico con una única luz.

RF4 - Requisito funcional

- Nombre: Mover vista de la escena
- Descripción: El usuario deberá poder manipular la vista de la escena, de forma que los objetos que contiene sean visibles desde diferentes ángulo, posiciones y distancias.
- Prioridad: Alta
- Precondiciones: La escena deberá tener al menos un objeto.
- Postcondiciones: El objeto se mostrará en la pantalla desde una nueva vista.

RF5 - Requisito funcional

- Nombre: Reiniciar vista
- Descripción: El usuario deberá poder restablecer en cualquier momento la vista de la escena, devolviendo el origen de esta al centro de la pantalla.
- Prioridad: Baja
- Precondiciones: La escena deberá tener al menos un objeto.
- Postcondiciones: El origen de la escena se establecerá en el centro del viewport.

RF6 - Requisito funcional

- Nombre: Configurar iluminación
- Descripción: El usuario deberá poder rotar la iluminación direccional de la escena con el fin de facilitar la visualización de los objetos que contiene.
- Prioridad: Baja
- Precondiciones: La escena deberá tener al menos un objeto.
- Postcondiciones: La iluminación rotará siguiendo la entrada del usuario.

RF7 - Requisito funcional

- Nombre: Importar archivo
- Descripción: El usuario deberá poder cargar una malla de un archivo .obj en el programa de forma que esta sea añadida a la escena como un objeto.
- Prioridad: Alta
- Precondiciones: Al menos deberá existir un archivo en la carpeta /models desde la cual se ejecuta la aplicación que contenga una malla compuesta por triángulos en formato obj.
- Postcondiciones: Existirá un nuevo objeto en la escena que contendrá la malla almacenada en el archivo .obj.

RF8 - Requisito funcional

- Nombre: Exportar escena
- Descripción: El usuario deberá poder exportar la escena a un archivo .obj como una única malla, conservando todas las transformaciones realizadas en los objetos.
- Prioridad: Media
- Precondiciones: La escena deberá tener al menos un objeto.
- Postcondiciones: Se creará un nuevo archivo llamado export.obj en la carpeta /models conteniendo las mallas unidas de todos los objetos de la escena. En caso de que este archivo ya exista, se sobreescribirá.

RF9 - Requisito funcional

- Nombre: Mover objecto
- Descripción: El usuario deberá poder desplazar el objeto desde su posición actual a una nueva posición en el espacio.
- Prioridad: Alta
- Precondiciones: La escena deberá tener al menos un objeto y este deberá estar seleccionado.
- Postcondiciones: El objeto seleccionado tendrá unas nuevas coordenadas en el espacio. En el caso de una selección múltiple, todos los objetos seleccionados tendrán unas nuevas coordenadas.

RF10 - Requisito funcional

- Nombre: Rotar objecto
- Descripción: El usuario deberá poder rotar el objeto a una nueva rotación, teniendo en cuenta su origen.
- Prioridad: Alta
- Precondiciones: La escena deberá tener al menos un objeto y este deberá estar seleccionado.
- Postcondiciones: Los objetos seleccionados se dibujarán según unas nuevas bases rotadas desde el origen de la escena.

RF11 - Requisito funcional

- Nombre: Escalar objecto
- Descripción: El usuario deberá poder escalar el objeto en todos sus ejes teniendo en cuenta su origen.
- Prioridad: Alta
- Precondiciones: La escena deberá tener al menos un objeto y este deberá estar seleccionado.
- Postcondiciones: Los objetos seleccionados se dibujarán siguiendo unas nuevas bases escaladas desde el origen cada uno de los objetos.

RF12 - Requisito funcional

- Nombre: Seleccionar objecto
- Descripción: El usuario deberá poder seleccionar un objeto de la escena de forma que sea sujeto de las transformaciones que quiera aplicarle.
- Prioridad: Alta
- Precondiciones: La escena deberá tener al menos un objeto.
- Postcondiciones: Todos los objetos menos el objeto seleccionado pasarán a estar no seleccionados, por lo que no podrán verse afectados por transformaciones.

RF13 - Requisito funcional
- Nombre: Seleccionar todos los objetos de la escena
- Descripción: El usuario deberá poder seleccionar todos los objetos de la escena de forma que sean sujetos de las transformaciones que quiera aplicar.
- Prioridad: Alta
- Precondiciones: La escena deberá tener al menos un objeto.
- Postcondiciones: Todos los objetos de la escena pasarán a estar seleccionados.

RF14 - Requisito funcional
- Nombre: Borrar objeto
- Descripción: El usuario deberá poder borrar un único objeto de la escena, conservando los restantes.
- Prioridad: Alta
- Precondiciones: La escena deberá tener al menos un objeto.
- Postcondiciones: El objeto seleccionado se borrará de la escena.

RF15 - Requisito funcional
- Nombre: Borrar todo
- Descripción: El usuario deberá poder borrar todos los objetos de la escena y devolver su origen al centro de la pantalla.
- Prioridad: Baja
- Precondiciones: La escena deberá tener al menos un objeto.
- Postcondiciones: Se eliminarán todos los objetos contenidos en la escena y su origen se establecerá en el centro del viewport.

RF16 - Requisito funcional
- Nombre: Unir mallas de objetos
- Descripción: El usuario deberá poder unir las mallas de un grupo de objetos, generando un nuevo objeto con las mallas de los objetos anteriores combinadas.
- Prioridad: Media
- Precondiciones: La escena deberá tener al menos dos objetos cuyas mallas no tengan vértices con coordenadas coincidentes.
- Postcondiciones: Se creará un nuevo objeto con las mallas unidas de los objetos seleccionados. Estos objetos se eliminarán.

RF17 - Requisito funcional

- Nombre: Separar mallas de objetos
- Descripción: El usuario deberá poder separar una malla de un objeto en sus elementos independientes. Esto es, grupos de vértices que no tienen conexión entre sí.
- Prioridad: Alta
- Precondiciones: La escena deberá tener al menos un objeto con una malla triangularizada válida.
- Postcondiciones: Se crearán tantos objetos como mallas independientes contenga la malla del objeto original.

RF18 - Requisito funcional

- Nombre: Deshacer
- Descripción: El usuario deberá poder devolver el estado de la escena a un estado anterior.
- Prioridad: Baja
- Precondiciones: La escena deberá tener al menos un objeto.
- Postcondiciones: La escena volverá al estado anterior a ejecutar esta acción.

2.2. Requisitos no funcionales

Los requisitos no funcionales listados a continuación definen aspectos del proyecto que no están directamente relacionados con su funcionalidad. Los requisitos no funcionales del proyecto se organizarán en la siguiente categorías:

- Requisitos de diseño.
- Requisitos de interfaz.
- Requisitos de rendimiento.
- Requisitos del proyecto.

Para clasificar la prioridad de los requisitos no funcionales se usará el siguiente criterio:

- Esencial: Si el requisito no se cumple el proyecto fracasa.
- Deseable: Es recomendable que el proyecto cumpla con el requisito.
- Opcional: El proyecto podría cumplir con el requisitos, pero el hecho de no hacerlo no supondría un mayor impacto en el mismo.

A continuación se listan los requisitos no funcionales del proyecto, organizados por categorías. Para documentar cada uno de los requisitos se usará la siguiente tabla:

RN## - Requisito No Funcional

- Nombre: Nombre del requisito no funcional.
- Descripción: Descripción del requisito no funcional.
- Prioridad: Esencial, deseable u opcional.
- Criterio de aceptación : Condición que ha de cumplirse para que el requisito sea aceptado.

2.2.1. Requisitos de diseño

Los siguientes requisitos establecen restricciones que se aplican a la hora de diseñar y desarrollar el producto software.

RN1 - Requisito No Funcional
- Nombre: Ejecutable en un PC con Linux
- Descripción: La aplicación deberá poder ejecutarse en un PC con una distribución de Linux instalada.
- Prioridad: Esencial
- Criterio de aceptación : El software se ejecuta en un PC con Manjaro 17.

RN2 - Requisito No Funcional
- Nombre: OpenGL 3.0 o posterior
- Descripción: El programa deberá ejecutarse bajo OpenGL 3 para aumentar su compatibilidad y tener un rendimiento adecuado.
- Prioridad: Deseable
- Criterio de aceptación : El software se ejecuta sin errores de renderizado en una GTS650 compatible con OpenGL 3.0.

RN3 - Requisito No Funcional
- Nombre: SDK V2 de Leap Motion
- Descripción: La aplicación deberá ser compatible con el SDK de la versión 2 del driver de Leap Motion para aumentar su compatiblidad, ya que no usa funcionalidades de realidad virtual implementadas en el diver Orion.
- Prioridad: Esencial
- Criterio de aceptación : El software se ejecuta correctamente y detecta el sensor Leap Motion en un ordenador con el driver 2.0 instalado, sin estar actualizado a Orion.

RN4 - Requisito No Funcional
- Nombre: Resolución mínima de 600 x 1024
- Descripción: Se establece una resolución mínima de 600 x 1024 para aumentar la compatibilidad de la aplicación y su rendimiento.
- Prioridad: Deseable
- Criterio de aceptación : Toda la interfaz y menús del programa se dibujan de forma completa en una ventana de tamaño 600x1024.

RN5 - Requisito No Funcional
- Nombre: Único usuario
- Descripción: La aplicación deberá se diseñará de forma que sea usable por un único usuario simultaneo
- Prioridad: Esencial
- Criterio de aceptación : La aplicación no tendrá ningún modo que soporte la interacción de varios usaurios con la misma. Tampoco tendrá ningún tipo de control de acceso, roles o identificación de usuarios.

2.2.2. Requisitos de interfaz

Los siguientes requisitos establecen aspectos relacionadas con la interfaz del programa y la forma en la que el usuario interactuará con el mismo.

RN6 - Requisito No Funcional
- Nombre: Usable con Leap Motion
- Descripción: La aplicación deberá ser usable mediante un sensor Leap Motion.
- Prioridad: Esencial
- Criterio de aceptación : Se pueden comprobar todos los requisitos funcionales usando solamente el sensor Leap Motion, sin usar ratón o teclado.

RN7 - Requisito No Funcional
- Nombre: Iconos para las opciones del menú
- Descripción: Se diseñarán iconos para los menús de la aplicación de forma que su usabilidad aumente.
- Prioridad: Opcional
- Criterio de aceptación : Todas las opciones del menú tienen una representación en forma de icono.

RN8 - Requisito No Funcional
- Nombre: Inglés
- Descripción: La aplicación tendrá su interfaz en inglés.
- Prioridad: Deseable
- Criterio de aceptación : Todas las opciones del menú tienen sus descripciones en inglés.

RN9 - Requisito No Funcional

- Nombre: Interfaz no dependiente del color
- Descripción: La aplicación no dependerá del color para distinguir funcionalidades o transmitir información al usuario. Esto hará el software más accesible para personas daltónicas que tenga problemas para distinguir ciertos colores.
- Prioridad: Deseable
- Criterio de aceptación : El programa debe ser usable y no presentar opciones ambiguas tras aplicar un filtro de blanco y negro al monitor en el que se está usando.

2.2.3. Requisitos de rendimiento

Los siguientes requisitos afectan al rendimiento del proyecto, buscando el el producto final se pueda ejecutar con un rendimiento deseable para la mayoría de usuarios.

RN10 - Requisito No Funcional

- Nombre: Carga de mallas
- Descripción: La aplicación deberá ser capaz de visualizar mallas de alta densidad.
- Prioridad: Esencial
- Criterio de aceptación : El programa tarda como máximo 20 segundos en cargar una malla en .OBJ de con un millón de vércites y mostrarla en pantalla en un ordenador con un i7 6700, una GTX 1080 y un SSD Samsung EVO Pro.

RN11 - Requisito No Funcional

- Nombre: 60 frames por segundo en el viewport
- Descripción: La aplicación deberá tener una tasa de frames estable para facilitar su uso
- Prioridad: Esencial.
- Criterio de aceptación : El programa es capaz de mantener 60 frames por segundo con una malla de 50k vértices en el viewport con una única luz direccional sin sombras en un ordenador con un i7 6700, una GTX 1080 y un SSD Samsung EVO Pro.

RN12 - Requisito No Funcional

- Nombre: Separación de mallas
- Descripción: El rendimiento a la hora de separar una malla en sus componentes deberá ser el adecuado.
- Prioridad: Deseable
- Criterio de aceptación : El programa es capaz de partir en sus componentes la malla de Suzane por defecto que incorpora Blender exportada como malla triangular y 2 niveles de subdivisión en un ordenador con un i7 6700, una GTX 1080 y un SSD Samsung EVO Pro en menos de 2 segundos.

2.2.4. Requisitos de proyecto

Estos requisitos afectan a los aspectos relacionados con el proyecto y a las tecnologías que usaran en el desarrollo del mismo:

RN13 - Requisito No Funcional

- Nombre: Software libre
- Descripción: Se usará software libre en la medida de lo posible para realizar el proyecto.
- Prioridad: Opcional
- Criterio de aceptación : Todos los componentes usados en el desarrollo del producto serán compatibles con la licencia MIT.

2.3. Gestión de la configuración

En esta sección se identificarán los elementos que forman la configuración del proyecto con el fin de mantener la integridad y evitar cambios no controlados en los mismos.

En el proyecto se pueden identificar los siguientes elementos de la configuración:

- Código fuente asociado al proyecto.
- Documentación.
- Motor de desarrollo y plantillas de exportación.
- SVG de iconos.

2.3.1. Control de versiones

El control de versiones del código de la aplicación se hará mediante un repositorio de git local. Cada vez que se añada un cambio significativo en el código o se añada una funcionalidad nueva se hará un nuevo commit explicando los cambios. Al finalizar la jornada de trabajo se hará un push al servidor local, de forma que siempre haya una copia del código en dos ordenadores distintos.

La documentación está escrita en Latex. Ya que no tiene una relación fuerte con la versión del código que se está manejando se almacenará en Google Drive. Dicho servicio almacena los cambios en la nube y guarda diferentes versiones de los archivos para poder volver a un estado de la documentación anterior si fuera necesario.

El proyecto usará una versión en desarrollo de un motor de juegos, por lo que es importante almacenar la versión en la que se encuentra su desarrollo para evitar incompatibilidades. Dicha versión del ejecutable del motor se sincronizará con el repositorio de git local.

Los iconos se almacenarán en un SVG, también almacenado en Google Drive. Los iconos exportados una vez incluidos en el proyecto estarán incluidos también en el repositorio de git junto con el código de la aplicación.

2.4. Alcance

En esta sección se detallará el alcance del proyecto, necesario para realizar una planificación correcta y centrar los esfuerzos en las áreas necesarias en cada momento para generar el producto software deseado.

2.4.1. Definición del alcance del proyecto

El proyecto generará un software para PC compatible con Linux que permitirá la visualización y manipulación de objetos 3D. Dicho software deberá poder cargar e importar objetos desde un archivo, moverlos, rotarlos, escalarlos y modificar su malla para poder cortarlos y separarlos en sus componentes.

Este software deberá ser controlado enteramente mediante el sensor Leap Motion, por lo tanto es fundamental diseñar unos paradigmas de interacción y una interfaz adaptada al software que se va a realizar.

2.4.2. Criterios de aceptación

Para ser aceptado, el producto final deberá cumplir con los requisitos marcados con prioridad alta en la especificación de requisitos del proyecto. El tiempo sobrante se dedicará a implementar los requisitos restantes.

2.4.3. Entregables

El proyecto generará los siguientes entregables:

- Código fuente del proyecto producido a lo largo del mismo.
- Ejecutable del proyecto para la plataforma Linux.
- Memoria del proyecto en la que se detallarán los detalles de la gestión del proyecto, sus características de diseño e implementación.
- Manual de usuario del software destinada al usuario final del mismo.

2.4.4. Exclusiones

A lo largo del proyecto será necesario usar modelos 3D y mallas para desarrollar y probar las diferentes funcionalidades. La creación de estos modelos no forma parte del proyecto, por lo que se usarán modelos ya disponibles. Estos archivos tampoco se distribuirán con el producto final.

Dado que el proyecto será publicado con una licencia MIT la biblioteca para el acceso al Leap Motion ni su driver podrán ser distribuidas con el mismo. Estos archivos son gratuitos y se podrán descargar desde la página de Leap Motion para que el proyecto se ejecute.

El proyecto será desarrollado para Linux. En caso de que sea necesario unos archivos binarios compilados para una plataforma diferente no se establecerán los pasos para su compilación en dichas plataformas.

Leap Motion	80€
Ordenador de desarrollo	4500€
TOTAL:	**4580€**

Cuadro 2.1: Costes asociados al hardware del proyecto

2.4.5. Restricciones

Para realizar el proyecto se tendrán en cuenta las siguientes restricciones:

- El tiempo máximo para desarrollar el proyecto es de 410 horas.
- El proyecto se realizará en Linux, concretamente en su distribución Manjaro 17. No se atenderán a problemas de compatibilidad con otras plataformas.
- Toda la funcionalidad especificada en los requisitos deberá ser accesible mediante el uso del sistema Leap Motion.

2.5. Costes

En esta sección se recogerán los costes del proyecto.

2.5.1. Hardware

En la tabla 2.1 se muestra la lista del hardware usado en la realización del proyecto, indicando su coste.

2.5.2. Recursos humanos

El proyecto ha sido realizado por una sola persona, la cual ha cumplido diferentes roles en diferentes partes del ciclo de vida del proyecto. Para calcular el coste asociado a los recursos humanos se asumirá que una persona ejerciendo los siguientes roles cobra las siguientes cantidades de euros a la hora:

- Gestor de proyecto: 35€/hora.
- Diseñador gráfico: 30€/hora.
- Analista/Programador: 40€/hora.

Una jornada de trabajo de este proyecto equivale a 5 horas. Trabajando de lunes a viernes durante 17 semanas se obtienen los resultados mostrados en la tabla 2.2.

Por lo tanto, sumando los costes del hardware y los recursos humanos el proyecto tendrá un coste total de 20705€.

	Semanas	Horas	Precio/Hora	Precio
Gestor de proyecto	5	125	35€	4375€
Diseñador gráfico	1	25	30€	750€
Analista/programador	11	275	40€	11000€
			Total:	**16125€**

Cuadro 2.2: Costes asociados a los recursos humanos del proyecto

2.6. Metodología

Una vez de definido los requisitos y el alcance del proyecto se definirá la metodología a seguir para realizarlo. La metodología elegida deberá ser compatible con las características del proyecto, de forma que aumente su probabilidad de éxito.

Dadas las características de este proyecto se descarta el uso de una metodología predictiva a favor de una metodología ágil por las siguientes razones:

- El cliente está disponible, por lo que puede ser consultado en caso de dudas o para la evaluación de los requisitos que están siendo implementados en cada momento.
- No se cuenta con la suficiente experiencia en determinadas áreas de la tecnología a utilizar, por lo que no es posible hacer una predicción exacta del desarrollo de proyecto de forma global.
- Los requisitos ya especificados están sujetos a cambio, especialmente los requisitos no funcionales relativos al rendimiento del proyecto.

Se ha elegido scrum como metodología para ejecutar el proyecto, ya que es compatible con los puntos mencionados anteriormente.

La metodología scrum define tres fases diferenciadas en el desarrollo del proyecto:

1. La primera fase se dedica al análisis del proyecto. En esta fase se identifican los requisitos, se toman decisiones generales de diseño y se crea un boceto de la arquitectura del software que se va a desarrollar.
2. La segunda fase consisten en la realización consecutiva de sprints para desarrollar el producto software. Cada sprint incremental tendrá una duración de 1 a 2 semanas. Estos periodos cuentan con una fase de planificación, en la que se identifican las tareas a realizar, una de ejecución y una revisión por parte del cliente.
3. En la última fase se crean la documentación y manuales, finalizando así el proyecto.

2.7. Planificación

La figura 2.2 muestra el diagrama de Gantt de la planificación del proyecto.

La figura 2.1 muestra la descomposición del trabajo del proyecto.

En este proyecto se ha llevado a cabo una fase de planificación inicial en la que se ha determinado el contenido de cada sprint.

A continuación se detalla el trabajo realizado en cada sprint:

Sprint 1: Análisis de tecnologías

Descripción: En el primer sprint del proyecto se hará un análisis y valoración de las tecnologías disponibles para implementarlo, así como un primer boceto de su interfaz y funcionalidad general.

Requisitos relacionados: RN1, RN2, RN3, RN4, RN5, RN10, RN12

Sprint 2: Lectura de datos desde el sensor Leap Motion

Descripción: En este sprint, el primero dedicado a desarrollo, se trabajará con el SDK de Leap Motion para poder acceder a los datos del sensor desde la tecnología usada para desarrollar el proyecto. Una vez finalizado este sprint se debe poder acceder a la posición y estado de cada una de las manos desde el entorno de desarrollo elegido para el proyecto, estado el Leap Motion ya integrado en el mismo.

Requisitos relacionados: RF1, RF2, RN1

Sprint 3: Viewport

Descripción: En este sprint se pretende mostrar una malla sencilla en pantalla y desarrollar el movimiento del viewport. También se analizarán diferentes alternativas a los algoritmos para el movimiento de la cámara del viewport y se calibrará para un funcionamiento óptimo con el sensor.

Requisitos relacionados: RF3, RF4, RF9, RF10

Sprint 4: Menú básico

Descripción: Este sprint consite en codificar la estructura básica para realizar el menú de la aplicación, así como los algoritmos que permitan usar el Leap Motion para navegar de forma cómoda por el menú.

Requisitos relacionados: RN1

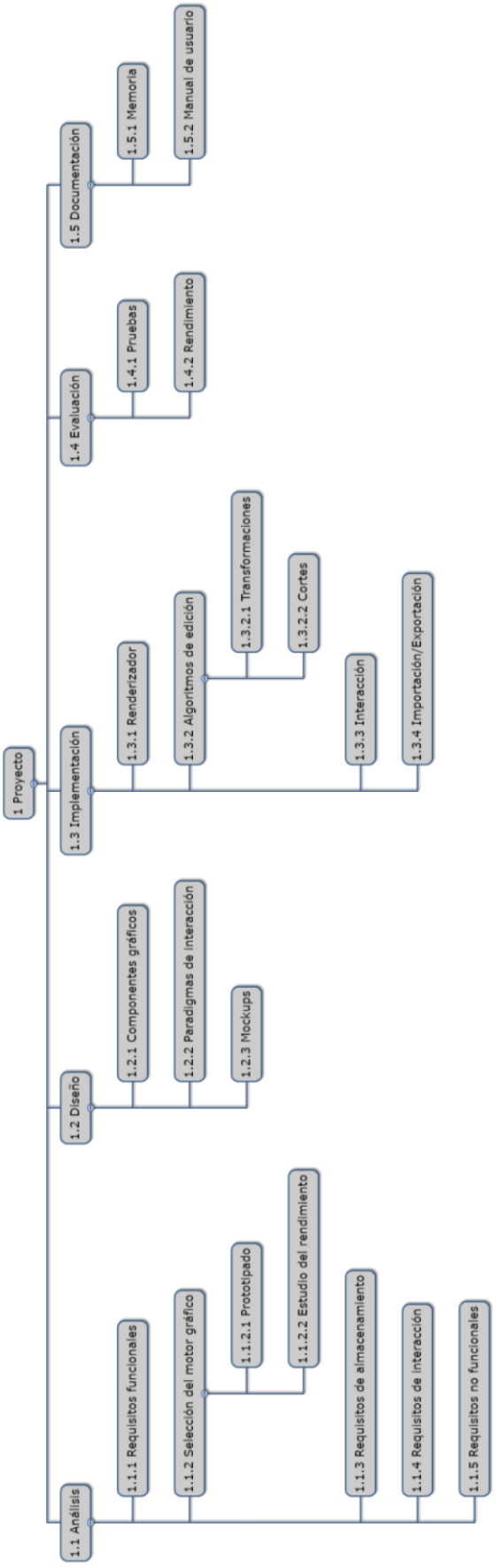

Figura 2.1: EDT del proyecto

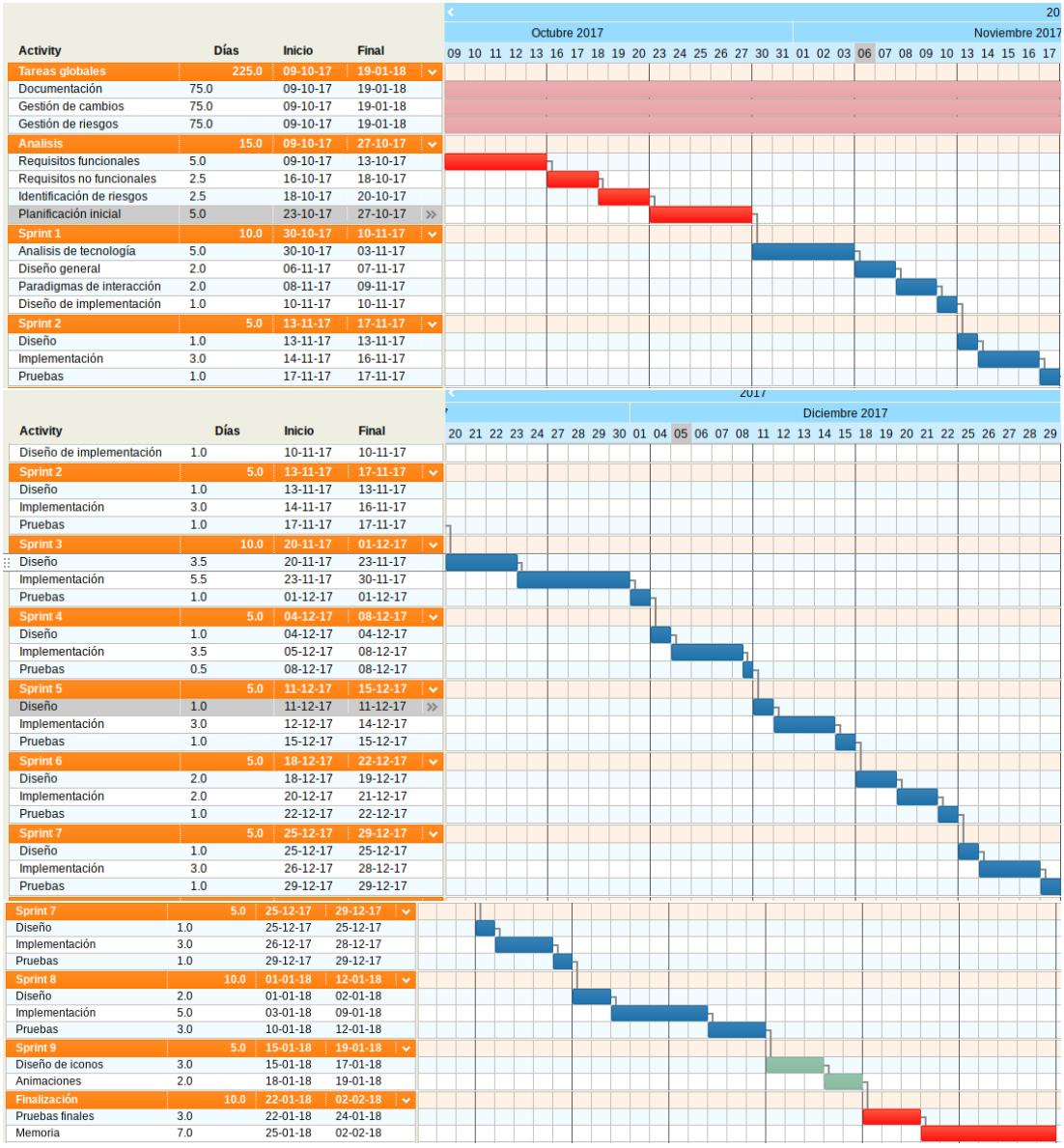

Figura 2.2: Diagrama de Gantt del proyecto

Sprint 5: Menús secundarios y operadores

Descripción: En este sprint se finaliza el menú y se codificará el sistema de operadores, de forma que se puede lanzar un operador sencillo. Esto completa la estructura para seguir ampliando el programa añadiendo más operadores los siguientes sprints.

Requisitos relacionados: RF5, RF6, RF7, RF8, RF11, RF12, RF13, RF14, RF15, RF16, RF17, RF18

Sprint 6: Listas y estado de los objetos

Descripción: En este sprint se desarrolla los menús especiales que listen objetos en la escena, así como funcionalidades que modifiquen el estado de los objetos del viewport, su creación o eliminación.

Requisitos relacionados: RF9, RF10, RF12, RF13, RF14, RF15

Sprint 7: Operadores de transformación y control de escena

Descripción: En este sprint se añaden los operadores relacionados con el escalado de objetos y el control general del viewport, por lo que es necesario experimentar con formas de interacción que permitan un uso satisfactorio de los mismos.

Requisitos relacionados: RF5, RF6, RF11, RN1

Sprint 8: Operadores de modificación de la malla y acceso a archivos

Descripción: Este sprint consiste en desarrollar los algoritmos relacionados con la edición directa de la malla. Para ello es necesario experimentar con la API que proporciona la tecnología elegida para el proyecto y evaluar su rendimiento. También se implementan los operadores de carga y guardado de archivos.

Requisitos relacionados: RF7, RF8, RF16, RF17, RN10, RN12

Sprint 9: Diseño gráfico y animaciones

Descripción: En el último sprint se diseñan los iconos para los menús y se añaden animaciones a los mismos para hacer la interfaz más agradable. También se añade la barra de estado para facilitar el uso de la aplicación por parte del usuario.

Requisitos relacionados: RF18, RN7, RN8, RN9

	S1	S2	S3	S4	S5	S6	S7	S8	S9
RF1		X							
RF2		X							
RF3			X						
RF4			X						
RF5					X		X		
RF6					X		X		
RF7					X			X	
RF8					X			X	
RF9			X			X			
RF10			X			X			
RF11					X		X		
RF12					X	X			
RF13					X	X			
RF14					X	X			
RF15					X	X			
RF16					X			X	
RF17					X			X	
RF18					X				X

Cuadro 2.3: Matriz de trazabilidad: Requisitos funcionales y sprints

	S1	S2	S3	S4	S5	S6	S7	S8	S9
RN1	X	X		X			X		
RN2	X								
RN3	X								
RN4	X								
RN5	X								
RN6	X								
RN7									X
RN8									X
RN9									X
RN10								X	
RN11	X								
RN12								X	
RN13	X								

Cuadro 2.4: Matriz de trazabilidad: Requisitos no funcionales y sprints

	Muy bajo	Bajo	Medio	Alto	Muy Alto
Insignificante	Muy bajo	Muy bajo	Bajo	Bajo	Medio
Moderado	Muy bajo	Bajo	Bajo	Medio	Alto
Serio	Bajo	Bajo	Medio	Alto	Muy alto
Catastrófico	Bajo	Medio	Alto	Muy alto	Muy alto

Cuadro 2.5: Especificación de la exposición a un riesgo dado su impacto y probabilidad de aparición

2.8. Riesgos

En esta sección se llevará a cabo un análisis de los riesgos que afectan a la realización del proyecto con el fin de identificarlos para poder anticiparse a ellos y buscar estrategias para evitar que el proyecto fracase.

Los riesgos se clasificarán en las siguiente categorías:

- Riesgos de gestión del proyecto
- Riesgos de disponibilidad
- Riesgos de desarrollo
- Riesgos de recursos humanos

La probabilidad de un riesgo será definida por una escala con los valores muy bajo, bajo, medio, alto y muy alto.

Se usará el siguiente criterio para clasificar el impacto de cada uno de los riesgos:

- Insignificante: el impacto sería inapreciable en el desarrollo normal del proyecto.
- Moderado: el impacto afectaría al desarrollo del proyecto, pero este podría continuar con normalidad.
- Serio: el impacto pondría seriamente en peligro la viabilidad del proyecto.
- Catastrófico: el impacto implicaría la terminación del proyecto.

Para calcular la exposición del proyecto a determinado riesgo en función de su probabilidad e impacto se usará el cuadro 2.5.

Los riesgos serán especificados siguiendo la siguiente plantilla:

Riesgo ##

- Nombre: Nombre del riesgo
- Descripción: Descripción del riesgo
- Probabilidad: Probabilidad de que el riesgo se materialice
- Impacto: Magnitud del impacto del riesgo

- Medidas de prevención: Acciones para evitar la aparición del riesgo
- Medidas de corrección: Acciones para minimizar el impacto en caso de que el riesgo llegara a materializarse

A continuación se listarán los riesgos del proyecto, clasificados en diferentes categorías:

2.8.1. Riesgos de gestión del proyecto

Los riesgos especificados a continuación afectan a los aspectos relacionados con la gestión del proyecto.

Riesgo 1

- Nombre: Alcance del proyecto incorrecto
- Descripción: El alcance del proyecto no se ha limitado correctamente, por lo que se está realizando trabajo innecesario.
- Probabilidad: Baja
- Impacto: Medio
- Medidas de prevención: Comprobar que las funcionalidades que se implementarán cumplen con los requisitos del proyecto, revisando su alcance las veces que sea necesario.
- Medidas de corrección: Eliminar cualquier funcionalidad extra que no esté especificada en el proyecto, dedicando tiempo al desarrollo de funcionalidades deseadas.

Riesgo 2

- Nombre: Requisito no especificado correctamente
- Descripción: Un requisito del proyecto no ha sido documentado correctamente, por lo que el producto final no corresponde con el deseado.
- Probabilidad: Medio
- Impacto: Serio
- Medidas de prevención: Repasar los requisitos en caso de encontrar información poco clara.
- Medidas de corrección: Corregir la especificación de requisitos y cambiar su implementación si fuera necesario.

Riesgo 3

- Nombre: Presupuesto insuficiente
- Descripción: El presupuesto destinado al proyecto no es suficiente para cubrir sus costes.
- Probabilidad: Bajo
- Impacto: Moderado
- Medidas de prevención: Repasar el presupuesto del proyecto y comprobar que este se corresponde con la planificación.
- Medidas de corrección: Pedir más presupuesto para conseguir que el proyecto cumpla con las funcionalidades mínimas.

2.8.2. Riesgos de disponibilidad

Los riesgos descritos a continuación afectan a la disponibilidad de material hardware necesario para realizar el proyecto

Riesgo 4
- Nombre: Rotura o pérdida del sensor Leap Motion
- Descripción: El sensor Leap Motion se pierde o estropea de forma que el desarrollo y pruebas de la aplicación no pueden continuar.
- Probabilidad: Bajo
- Impacto: Serio
- Medidas de prevención: Conseguir más de un sensor a la hora de iniciar el proyecto.
- Medidas de corrección: Comprar un sensor nuevo.

Riesgo 5
- Nombre: Fallo en el ordeandor de desarrollo
- Descripción: El ordenador principal en el que se desarrollará el proyecto falla.
- Probabilidad: Medio
- Impacto: Alto
- Medidas de prevención: Tener acceso a un ordenador de prestaciones similares en el que se pueda seguir desarrollando el proyecto.
- Medidas de corrección: Comprar un ordenador nuevo con las prestaciones mínimas para desarrollar el proyecto.

2.8.3. Riesgos de desarrollo

Los riesgos citados a continuación afectan a la implementación del proyecto. Estos están relacionados con las tecnologías que se pretenden usar para llevarlo a cabo.

Riesgo 6
- Nombre: Fallo o pérdida del repositorio
- Descripción: Se pierde el repositorio en el que se almacenan los archivos relacionados con el proyecto.
- Probabilidad: Bajo
- Impacto: Catastrófico
- Medidas de prevención: Realizar copias de seguridad en una segunda ubicación física.
- Medidas de corrección: Reimplementar el proyecto en caso de que sea posible. Si el proyecto ya se encontraba en un estado avanzado, cancelarlo.

Riesgo 7

- Nombre: Errores en el motor gráfico
- Descripción: El motor gráfico contiene errores que impiden que el proyecto se desarrolle con normalidad.
- Probabilidad: Medio
- Impacto: Moderado
- Medidas de prevención: Probar con anterioridad las funcionalidades que se van a usar para el proyecto asegurándose de que están libres de errores.
- Medidas de corrección: Modificar el motor gráfico y corregir el error.

Riesgo 8

- Nombre: Rendimiento insuficiente
- Descripción: El rendimiento que proporciona la tecnología en la que se realiza la implementación no es suficiente para cumplir lo especificado en los requisitos no funcionales.
- Probabilidad: Medio
- Impacto: Serio
- Medidas de prevención: Comprobar previamente el rendimiento de dicha tecnología en un caso de uso similar.
- Medidas de corrección: Reimplementar la parte crítica para el rendimiento en otra tecnología.

2.8.4. Riesgos de recursos humanos

Los siguientes riesgos afectan a las personas que realizarán el proyecto.

Riesgo 9

- Nombre: Perdida de personal
- Descripción: El personal dedicado a la realización del proyecto no está disponible.
- Probabilidad: Bajo
- Impacto: Serio
- Medidas de prevención: Asegurarse de que otra persona pueda realizar ciertas actividades relacionadas con el proyecto para seguir con la planificación si fuera necesario.
- Medidas de corrección: Contactar con personas dispuestas a participar en el proyecto para continuar con la planificación.

Riesgo 10

- Nombre: Desconocimiento de las tecnologías
- Descripción: El personal que implementará el proyecto no conoce las tecnologías de desarrollo, impidiendo su implementación correcta.
- Probabilidad: Muy bajo
- Impacto: Serio
- Medidas de prevención: Asegurarse de contar con personal formado en las tecnologías que se usarán en el proyecto.
- Medidas de corrección: Formar al personal en las tecnologías del proyecto.

2.9. Análisis de tecnologías

En este apartado se analizarán diferentes soluciones tecnológicas a la hora de implementar el proyecto. Dichas opciones deberán ser compatibles con los requisitos del proyecto y facilitar su desarrollo e implementación en la medida de lo posible.

Al ser un proyecto basado en su mayor parte en su parte gráfica, esta será la que mayor tiempo de desarrollo consumiría. Por lo tanto, se valorará tanto la opción de programar un renderizador de cero adaptado al proyecto como la de usar un renderizador ya existente disponible en un motor de juegos.

2.9.1. Criterios de selección

Se decidirá una tecnología para desarrollar el proyecto. Para ello se tendrán en cuenta las siguientes cuestiones:

- El proyecto cuenta con una parte 3D (viewport) y una 2D (interfaz gráfica). La tecnología elegida deberá soportar ambos modos de funcionamiento y facilitar en la medida de lo posible el desarrollo.
- La tecnología elegida deberá ser compatible con el SDK de Leap Motion de forma que cumpla con los requisitos RF1, RF2, RN3 y RN6
- De cara a la posible expansión del programa o posibles problemas de rendimiento, la tecnología elegida deberá soportar tecnologías de desarrollo que no supongan problemas adicionales en este aspecto. Esto afectaría a los requisitos RF14 y RF15 relacionados con la manipulación de mallas, así como a los requisitos RN10, RN11 y RN12.

2.9.2. Implementación propia

Esta opción consistirá en programar un renderizador en OpenGL 3.0 adaptado a las necesidades del proyecto. Para cumplir con los requisitos del proyecto, este renderizador debería contar con las siguientes funcionalidades:

- Dibujado de mallas en 3D
- Dibujado de sprites 2D
- Dibujado de texto
- Soporte para materiales, al menos para modificar su color con un valor fijo.

Ventajas

- El renderizador estaría completamente adaptado a las funcionalidades del proyecto, por lo que tendría un rendimiento optimo
- Leap Motion cuenta con un SDK compatible con gran cantidad de lenguajes de programación, entre ellos C++, Java o Python

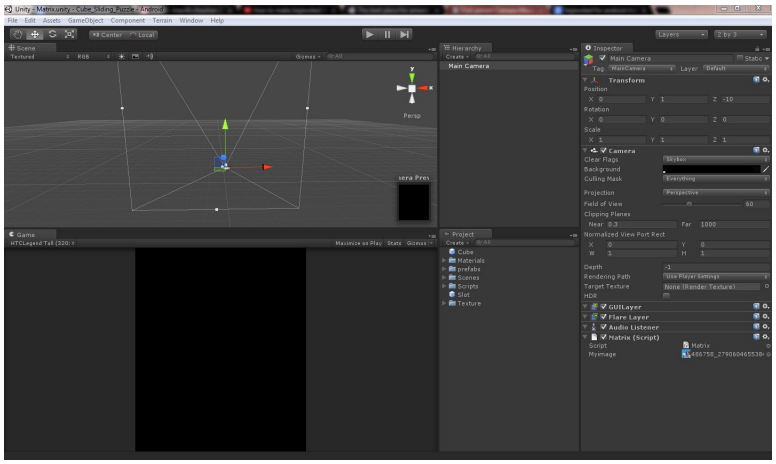

Figura 2.3: Editor del motor Unity 5

- Mayor flexibilidad a la hora de estructurar y programar el resto del proyecto

Inconvenientes

- Programar un renderizador de cero aumentará el coste y duración del proyecto
- No se dispone de ningún framework base para la creación de la interfaz. Las opciones restantes serían programar uno o genar una interfaz suficientemente sencilla como para no necesitar framework.
- No se cuenta con un editor gráfico para visualizar el proyecto mientras se desarrolla
- Ninguna ayuda a la hora de tratar otros aspectos del proyecto, como la lectura de archivos o la gestión de las ventanas de la aplicación.

2.9.3. Unity 2017

Unity es un motor gráfico multiplataforma orientado a videojuegos y aplicaciones interactivas. Cuenta con las herramientas básicas para realizar juegos en 2D y 3D, un potente editor y una gran comunidad de usuarios. El lenguaje principal para programar proyectos en el motor es C#, y este está implementado en C y C++ de forma cerrada. Su funcionamiento se basa en el modelo entidad-componente.

A pesar de su fama, muchos estudios no lo consideran como una opción seria a la hora de realizar sus proyectos porque su estructura interna no está preparada para soportar juegos de gran complejidad, por lo que se considera un motor gráfico de juguete para realizar pequeños proyectos independientes o iniciarse en el mundo del desarrollo de videojuegos.

Ventajas

- Es un motor gráfico muy usado, por lo que cuenta con una buena documentación y un gran soporte de la comunidad

- Cuenta con un buen soporte con el SDK de Leap Motion, ya que se puede descargar un proyecto ya preparado para empezar a usar el sensor sin necesidad de modificar el motor.
- A pesar de contar únicamente con un renderizador 3D que usa en modo ortográfico para generar una imagen 2D, cuenta con unas herramientas suficientemente potentes para poder generar una interfaz de usuario compleja.

Inconvenientes

- Es de código cerrado, por lo que no habría posibilidad de modificar el motor en caso de que generara algún problema durante el desarrollo del programa.
- No permite el control del bucle principal del motor, por lo que no es posible determina el orden de ejecución de los componentes de los objetos. Esto puede dificultar el desarrollo del proyecto. Además es conocido que muchos estudios que trabajan con Unity necesitaron pedir el código del motor para solucionar este problema.
- El rendimiento del motor es terrible incluso en escenas muy sencillas, supuestamente por la alta cantidad de plataformas que soporta
- Solamente cuenta con C# como lenguaje de programación. A pesar de que esto no es un factor limitante para programar la interfaz gráfica del programa, puede serlo para crear los algoritmos de edición de la malla debido al recolector de basura.
- El mal diseño de su estructura interna impide usar correctamente un control de versiones con sus archivos, ya que no soporta que dos usuarios modifiquen una escena al mismo tiempo. En caso de necesitar esta funcionalidad es necesario el uso de plugins.

2.9.4. Unreal Engine 4

Unreal Engine 4 es un motor de juegos multiplataformas orientado a títulos que requieren un alto rendimiento y calidad. Cuenta con uno de los mejores renderizadores 3D en tiempo real disponibles actualmente en el mercado, así como otras herramientas como editor gráfico de materiales, físicas o sistema de AI.

El código del motor está disponible, pero es demasiado complejo como para poder hacer modificaciones en el desarrollo de este proyecto.

Su lenguaje de programación más integrado con el editor es Blueprints, un sistema de scripting visual. Además, también soporta C++.

Ventajas

- Tiene una buena documentación y una gran comunidad de usuarios
- Cuenta con uno de los mejores motores gráficos disponibles al público actualmente, tanto en calidad como en rendimiento

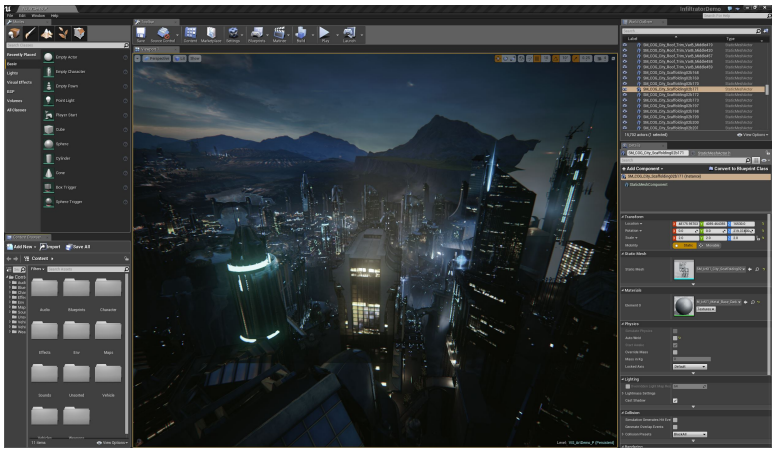

Figura 2.4: Editor de Unreal Engine 4

- Su lenguaje de programación principal es C++, por lo que no habría ningún tipo de problema con el rendimiento del programa
- Cuenta con un soporte directo del SDK de Leap Motion mediante un proyecto ya creado

Inconvenientes

- No es la mejor opción para desarrollar en 2D y esto podría suponer un problema dado que gran parte del desarrollo del proyecto consistirá en la creación de los menús adaptados para el Leap Motion.
- A lo largo del proyecto no será necesario usar la gran mayoría de opciones que ofrece el renderizador del motor
- C++ no es el mejor lenguaje para prototipar interfaces gráficas

2.9.5. Godot Engine 3

Godot Engine 3 es un motor de juegos multiplataforma de código abierto. La versión 1.0 se ha liberado al público en el año 2015 tras 10 años de desarrollo.

La versión estable disponible a la hora de empezar el proyecto es la 2.1. Esta opción no se contempla ya que tiene muy pocas funcionalidades a la hora de desarrollar en 3D y no sería suficiente para cumplir con los requisitos de la aplicación que se desarrollará. Para la nueva versión 3.0 el renderizador y editor 3D se ha vuelvo a programar y ya cuenta con unas funcionalidades y calidad similares a las otras opciones disponibles en el mercado. Esta versión se encuentra en alpha y está planificado que la versión final ya esté disponible en las últimas etapas de este proyecto. La versión alpha es perfectamente usable y sus desarrolladores han dicho que no generará ninguna incompatiblidad seria con la versión final.

Su lenguaje principal es GDScript, un lenguaje especialmente diseñado para el motor y altamente integrado con su editor que se ejecuta en una máquina de bytecodes.

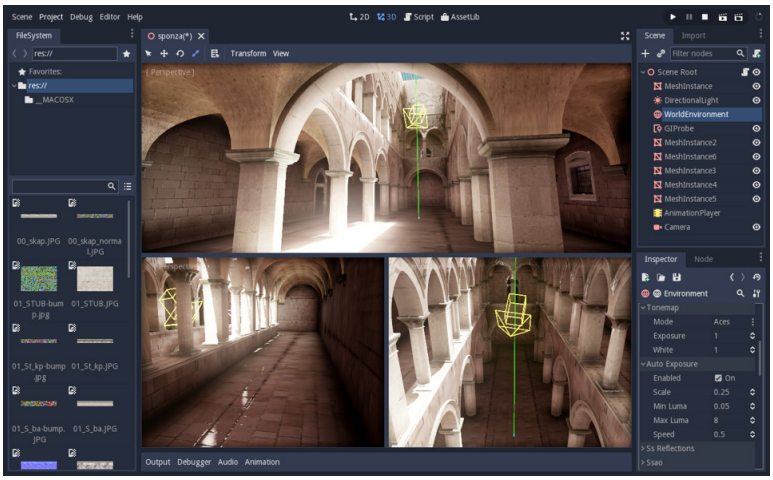

Figura 2.5: Editor de Godot Engine 3

Además en su versión 3.0 soporta C++, un lenguaje de VisualScript y C# mediante Mono.

Ventajas

- Es de código abierto y es fácilmente modificable. Se podrían añadir nuevas funcionalidades al motor en caso de que fuera necesario para el proyecto
- Cuenta con un renderizador 2D y 3D por separado, por lo que facilitará tanto la creación de la interfaz gráfica como del viewport del programa
- El renderizador 3D que incorpora cuenta con una calidad y rendimiento suficientes para el proyecto
- Soporta varios lenguajes de programación simultáneos en un mismo proyecto, pudiendo elegir entre rendimiento o facilidad de uso según convenga.

Inconvenientes

- La versión más reciente (3.0) del motor se encuentra en alpha a la hora de iniciar el proyecto, por lo que puede hacer que ciertas partes del programa a desarrollar sean incompatibles con la versión final del motor.
- No tiene ningún tipo de soporte directo con el Leap Motion por lo que habría que desarrollar esta parte. A pesar de esto, cuenta con las funcionalidades necesarias para poder incorporar fácilmente al motor cualquier biblioteca ya disponibe.
- Muchas partes del motor no tienen documentación.

2.9.6. Conclusión

Después de evaluar todas las opciones detenidamente y teniendo en cuenta la experiencia desarrollando proyectos con estas plataformas, considero que la mejor opción para realizar el proyecto es Godot Engine 3 por las siguientes razones:

- Cuenta con herramientas independientes y optimizadas para el desarrollo 3D y 2D por separado.
- A pesar de que no dispone integración directa con Leap Motion, cuenta con una funcionalidad para usar bibliotecas de C++ de forma sencilla con el motor, por lo que el desarrollo de esta parte del proyecto no sería demasiado compleja. En caso de que esta opción no sea suficiente, se podría añadir el módulo al motor en un tiempo razonable.
- Soporta diversas opciones a la hora de programar proyectos en el motor. GDScrit está orientada a sencillez y velocidad de desarrollo, mientras que se podría usar C++ en cualquier momento para partes del proyecto en las que el rendimiento no sea suficiente-

Capítulo 3

Diseño

Este capítulo se centrará en el diseño de la aplicación, tanto a nivel de interacción, interfaz y estructura. En primer lugar se analizarán otras aplicaciones de edición 3D para detectar los conceptos y paradigmas con los que un usuario habitual de las misma estará familiarizado. Posteriormente estos conceptos se adaptarán a su uso con un Leap Motion para usarlos en esta aplicación. Por último se diseñará una estructura para el proyecto siguiendo los patrones de diseño que se pueden aplicar en Godot Engine de forma que la aplicación pueda ser fácilmente ampliable con nuevos requisitos funcionales en un futuro.

Contents

3.1. Sprint 1: Diseño general

En este Sprint se definirán las características generales del programa, su funcionamiento y estructura de datos, así como en los conceptos que estará basada su interacción. Para ello se analizarán otras aplicaciones de diseño 3D y posteriormente se establecerá una serie de comportamientos generales que deberán cumplir las acciones del Leap Motion dentro de la aplicación para poder interactuar con la interfaz.

3.1.1. Análisis de otras aplicaciones

En este apartado se analizarán las características que comparten la mayoría de programas de edición 3D para poder hacer una adaptación lo más correcta posible en el producto software que se está desarrollando. Este análisis se lleva a cabo durante el primer sprint del ciclo de vida del proyecto, ya que sus conclusiones afectan directamente a la estructura interna del programa.

Para ello se usarán dos grandes grupos de aplicaciones. Por una parte, las orientadas a una edición 3D tradicional basada en la modificación directa de los componentes de una malla (Blender, Maya, 3DSMax y Cinema4D). Por otra, las optimizadas para escultura y trabajo con mallas de alta densidad, que no requieren acceso a los componentes de la malla (Zbrush o Mudbox). Una vez detectadas sus similitudes y diferencias se procederá a seleccionar las características que más se adecuan a este proyecto.

Las aplicaciones de edición 3D se basan en el uso de objetos como principal entidad para abstraer los datos de una malla situados en una posición en el espacio. Estos objetos son contenedores de datos que, además de contener los datos de la malla, son responsables de almacenar posiciones, estado de visualización, características para el render o flags y máscaras para las simulaciones físicas (figura 3.1). Para ilustrar el concepto en diferentes softwares se muestran los siguientes ejemplos:

- En Blender, el modo por defecto de la aplicación es el modo objeto. En este modo se pueden acceder a las características del objeto mencionadas anteriormente mediante el panel de propiedades del objeto en el inspector. Al seleccionar un objeto se puede entrar en su modo edición para acceder a los componentes de su malla individualmente y poder modificarlos. Es importante notar que las transformaciones que se realizan en la malla y en el objeto son independientes, ya que el origen del espacio en el que se representarán las coordenadas de los vértices de la malla se almacenan a nivel de objeto.

- En 3DSMax tampoco se puede acceder a los componentes de la malla por defecto. Estos datos son visibles tras usar un modificador EditPoly(figura 3.2) , conservando también los dos niveles en la estructura de datos. Este funcionamiento también aporta la posibilidad de almacenar diferentes estados de la edición de la malla en varios modificadores EditPoly.

Además, lo común de este tipo de programas es permitir las jerarquías de objetos. Un objeto hijo de otro hereda todas sus transformaciones y su espacio. Esto es útil para rigging o para la organización de objetos.

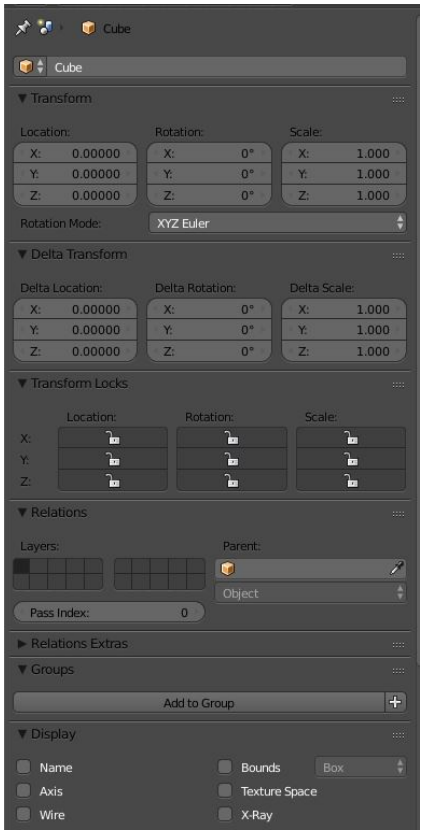

Figura 3.1: Inspector de las propiedades de un objeto de Blender

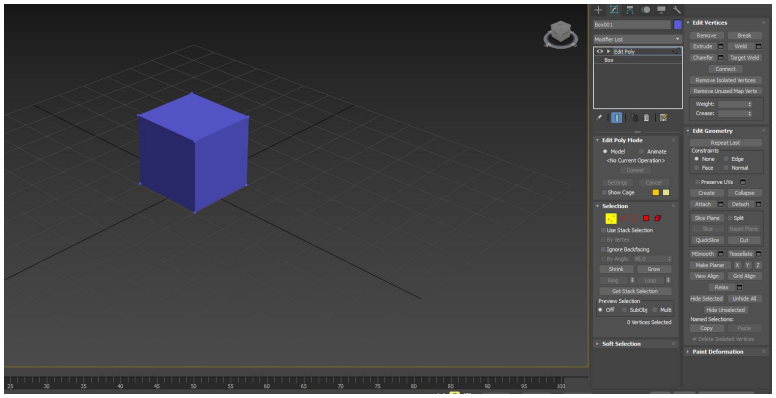

Figura 3.2: Modificador EditPoly de 3DS Max

Figura 3.3: Lista de subtools en ZBrush

También comparten el concepto de escena, la cual es el conjunto de objetos que se visualizan en un momento determinado, con los que se está trabajando.

Por otra parte tenemos el funcionamiento de Zbrush. En este programa, el concepto de objeto se substituye por el de herramienta (Tool). Una herramienta contiene muchos más datos que un objeto en los programas tradicionales, como los estados de todas las subdivisiones de la malla, su historial o sus modos de edición activos. Estas herramientas contienen a su vez subherramientas (subTools) (figura 3.3). Estas subTools tienen las siguientes características:

- A cualquier Tool se le puede añadir una subTool como un componente interno, pero no se permiten las jerarquías. No se puede añadir una subTool como hija de otra subTool.

- Solo una subTool puede estar seleccionada al mismo tiempo. Cuando se selecciona una nueva subTool como activa el resto se bloquea automáticamente. Para hacer transformaciones globales es necesario unir todas las subTools.

- Las subTools se comportan de forma diferente en función de su estado de selección. Por ejemplo, alterar la visibilidad de una subTool seleccionada ocultará todas las demás subTools menos esta, mientras que alterar la visibilidad de una subTool no seleccionada afectará solamente a dicha subTool.

- Los datos de las subTools pueden unirse o separarse en una nueva subTool con un solo comando para poder hacer ediciones conjuntas.

Por lo tanto, la representación de la aplicación podría basarse en las siguientes premisas:

- Objetos y malla son dos conceptos separados. La malla son los datos sobre los vértices, aristas y caras que se almacenan dentro del objeto.
- El origen para las coordenadas de los vértices de la malla está contenido en el objeto.
- Las transformaciones sobre los modelos se podrán ejecutar tanto a nivel de objeto (modificando sus propiedades) o a nivel de malla (modificando la estructura interna de los vértices).

Basándose en las limitaciones por la precisión del Leap Motion, también se usarán otros conceptos procedentes de ZBrush, ya que no requieren acceso directo a componentes que necesiten una edición de precisión para poder cumplir los requisitos funcionales:

- Solo se permitirá un objeto seleccionado a la vez, o toda la escena seleccionada.
- Se deberá poder unir todos los objetos de la escena en uno solo para una edición conjunta y posteriormente volver a separarlos.
- Mientras un objeto esté seleccionado, los demás deberán estar bloqueados.

3.1.2. Paradigmas de interacción

El diseño de la aplicación deberá estar adaptado a su uso con Leap Motion, por lo tanto a las características especiales que este método de interacción supone. Tras usar diferentes aplicaciones con el Leap Motion y probar diferentes paradigmas de interacción dentro de las mismas se ha llegado a las siguientes conclusiones:

- No usar cursores: tanto el uso de un cursor 2D como 3D es una mala práctica para el diseño de aplicaciones con un Leap Motion. La principal razón es que la herramienta no cuenta con la suficiente precisión como para poder situar un cursor sobre un elemento de forma estable y poder hacer un gesto de selección sin desplazar el cursor de su posición. Los movimientos de selección deberían ser tan sutiles que el usuario acabaría activando sin querer esta acción demasiadas veces. Se han publicado diferentes aplicaciones para el control del cursor del sistema operativo mediante el Leap Motion, pero su uso produce una experiencia muy poco satisfactoria.

  En el caso de un cursor en 3D sería necesaria una representación espacial del mismo de forma que el usuario fuera consciente en todo momento de su posición en los tres ejes. A pesar de que esto es posible, es una forma confusa y poco directa de interactuar con un espacio en 3D y por lo mencionado anteriormente sobre la precisión del sensor tampoco aporta ningún beneficio real.

  Por lo tanto en la aplicación no se usará ningún tipo de cursor ni en el viewport ni en los menús de selección, haciendo que sea necesaria la creación de otros métodos de interacción.

- Optimizar para movimientos poco precisos: En las herramientas de interacción con un ordenador tradicionales (ratón, trackball, tabletas gráficas, pantallas táctiles) el usuario tiene siempre la mano apoyada sobre una superficie estática. Esto le facilita poder detener el movimiento de forma precisa y controlada. Dado que en el caso del Leap Motion el usuario deberá tener el brazo en el aire para que el sensor lo detecte, este no puede mantener el brazo completamente estático por mucho tiempo. La aplicación no deberá hacer que el usuario tenga que detener en ningún momento el movimiento de la mano para realizar una acción, así como no deberá hacer que el usuario coloque la mano en una posición precisa en el espacio para seleccionar un elemento.

  Para ello se tomará la siguiente decisión en el diseño de la aplicación: las manos se ignorarán a no ser que estén parcialmente cerradas. Esto dará al usuario más control, ya que puede detener la acción siempre que quiera abriendo la mano y a partir de ese momento seguir moviendo la mano libremente sin afectar al estado del programa.

- Usar ambas manos: El sensor Leap Motion proporciona información sobre la mano que el usuario está usando, por lo que se pueden asignar diferentes comportamientos a cada una de las manos. Esto hace que el acceso a diferentes funciones del programa sea mucho más rápido, así como ayuda que la consistencia del software aumente.

  Además, dicho comportamiento se puede alterar en diferentes estados del programa. Con determinados operadores activos se podrá sobrescribir el comportamiento por defecto de las manos para permitir una entrada de 6 ejes simultáneos o computar la distancia entre las mismas.

  En este caso, se hará una clara diferenciación del propósito que tiene cada una de las manos en el software. La mano derecha se usará para controlar la vista del viewport, mientras que la izquierda se usará para controlar el menú y los operadores. En caso de ser necesario el funcionamiento de las manos se puede intercambiar, pero para facilitar el desarrollo del tema se usará siempre la mano derecha como mano asignada a las funciones del viewport y la izquierda como mano asignada a las funciones de los operadores.

3.1.3. Diseño de la implementación

El programa se implementará usando el motor Godot Engine 3. Este entorno de desarrollo no se comporta siguiendo el paradigma tradicional de la programación orientada a objetos ya que añade más funcionalidades y flexibilidad sobre el mismo. Tiene las siguientes características:

- Un programa en Godot es un árbol de nodos. Cada nodo tiene unos métodos y propiedades por defecto, dependiendo de su clase.

- Un conjunto de nodos se denomina escena. Las escenas se pueden instanciar dentro del árbol de nodos de la aplicación y pueden heredar a otras escenas.

- Un nodo puede tener asignado un script. Este script extiende a la clase base del Nodo añadiendo nuevos métodos y propiedades. Un script también puede heredar a otro script.
- Los nodos que tiene una propiedad de tipo Transform la heredan de sus nodos padre en el árbol de la escena, siendo las bases de su padre su espacio local.
- Todos los nodos pueden subscribirse y emitir señales, teniendo un patrón observer implementado por defecto.
- Los nodos pueden contener recursos. Estos recursos pueden estar almacenados en la escena o en un archivo de recurso.
- El programa puede contener scripts globales que se iniciarán automáticamente con el mismo, añadiéndose al árbol acoplados a un nodo base. Estos scrips se comportan como clases con métodos estáticos y sus métodos y propiedades son accesibles desde todo el árbol de nodos.

Estructura general

A la hora de diseñar el programa el primera paso es identificar sus escenas y las responsabilidades de cada una. El programa contiene las siguientes escenas.

- Main: La escena principal del programa en la que se incluyen el resto de escenas. También contiene un nodo llamado UndoStack, cuyos hijos almacenarán diferentes estados del viewport para la funcionalidad de deshacer y el nodo Operador, responsable de iniciar y destruir los operadores que se añadan como hijos del mismo.
- Viewport: Escena responsable de toda la funcionalidad del viewport.
- StatusBar: Escena responsable de la barra de estado, mostrar el operador actual y el estado del sensor Leap Motion.
- Operator: Escena de la que heredan todos los operadores. Los operadores contienen un nodo con un script que implementa su funcionalidad. Para que el operador se ejecute ha de ser instanciado y añadido como hijo al nodo Operator de la escena Main.
- MainMenu: Escena responsable del menú principal y su estado, así como de interpretar el movimiento de la mano izquierda para poder desplazarse por los menús.
- Menu: Escena de la que heredan todos los menús del menú principal. Es responsable de iniciar todos los elementos de dicho menú así como de seleccionar sus items.
- MenuItem: escena responsable del widget que muestra las opciones de los menús y de instanciar los operadores una vez seleccionados.

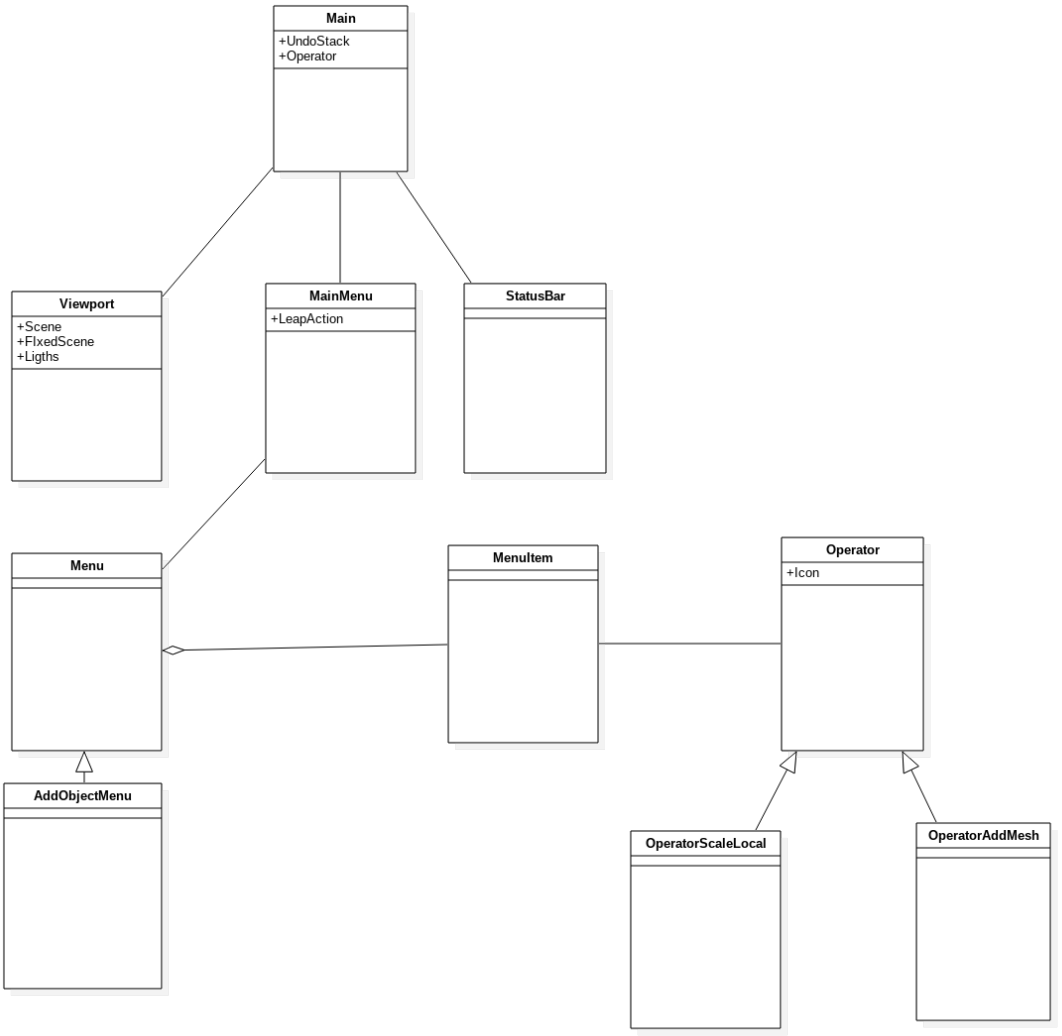

Figura 3.4: Diagrama de escenas del proyecto

En la figura 3.4 se muestra un diagrama con las escenas del proyecto, sus dependencias y organización, así como los nodos que contienen que aportan las funcionalidades más importantes.

Por otra parte, el programa contará con los siguientes scripts globales:

- LEAP: Script encargado de proporcionar los métodos para interactuar con el Leap Motion al resto de la aplicación.

- MeshUtils: Script que proporciona métodos para realizar transformaciones, accesos, lecturas y escrituras de una malla a nivel de vértices. Este script es usado por determinados operadores.

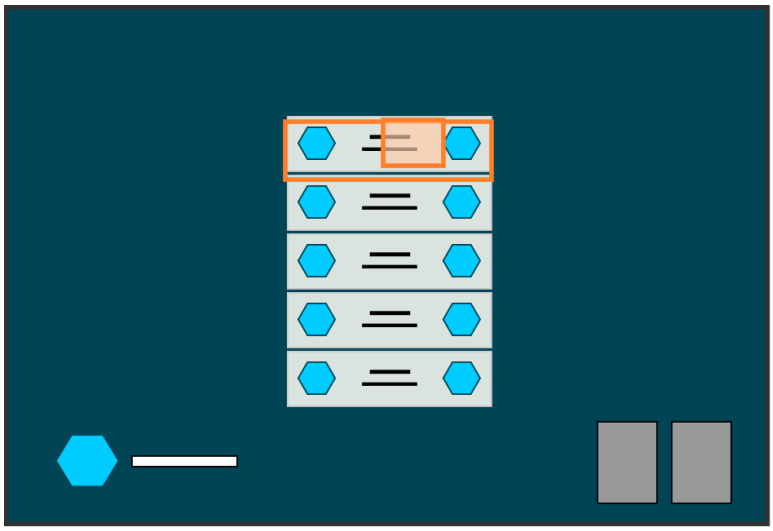

Figura 3.5: Mockup final de la aplicación

Figura 3.6: Mockup de una versión del deslizador para seleccionar opciones no implementada

3.1.4. Mockups

Para prototipar la interfaz del software se han realizado varios mockups, que se han perfeccionado en los sprints 4 y 5 dedicados al funcionamiento de los menús. El mockup mostrado corresponde a la opción final.

A la hora de decidir la apariencia de los widgets del menú se ha considerado una forma similar al botón de "desliza para desbloquear"de las versiones antiguas de iOS para que el usuario pudiera identificar su funcionalidad más fácilmente. Se ha descartado porque ocupaba demasiado espacio, siendo más visible que el propio contenido de los menús (figura 3.6).

3.2. Sprint 2: Interfaz con Leap Motion

Este sprint ha sido dedicado a la creación de la interfaz con el sensor Leap Motion para conseguir leer los datos que proporciona.

Una vez analizados los requisitos y diseñado el funcionamiento del programa, se identifica que la interfaz con el Leap Motion, implementada en el script global LEAP,

contará con los siguientes métodos:

- get_hand_transform(hand): devuelve la matriz de transformación de la mano deseada.
- is_hand_visible(hand): devuelve true si la mano deseada es visible para el sensor.
- get_hand_status(hand): permite saber si la mano deseada está abierta o cerrada.
- is_leap_connected(): permite acceder al estado del sensor y saber si está preparado para su funcionamiento.

3.3. Sprint 3: Viewport

Este sprint está dedicado al diseño del funcionamiento del viewport del programa, así como a definir de forma más precisa su forma de interacción mediante el Leap Motion. Para ello se analizará el funcionamiento de otras aplicaciones de edición 3D, de forma similar a lo realizado en los sprints anteriores.

3.3.1. Analisis de otras aplicaciones

El viewport es el espacio de trabajo en el que se muestran los objetos en 3D que se están editando. Estos objetos se representan a través de una proyección que viene dada por una cámara que no pertenece a la escena. Las propiedades de esta cámara se pueden modificar para poder cambiar la vista o la proyección de la escena y poder trabajar sobre los objetos más cómodamente. Las principales transformaciones que se pueden hacer en estos softwares sobre la posición de la cámara son las siguiente:

- Rotación: la cámara rota sobre un punto no visible en la escena que se puede establecer manualmente o de forma automática dependiendo de la última selección. En la mayoría de los softwares se puede elegir entre mantener la rotación fija en el eje Z (turntable) o rotar libremente en todos los ejes (trackball).
- Panning: el origen de la cámara se desplaza en un plano paralelo al plano de proyección de la cámara. Esto permite poder desplazar la escena lateralmente.
- Zoom: La cámara varía su distancia a un punto invisible situado en la escena. Una vez se alcanza la posición de este punto no se puede hacer más zoom y la cámara se bloquea.

Para activar este tipo de movimientos en la cámara la mayoría de los softwares de edición 3D usan las teclas modificadoras, la pulsación de un botón del ratón y el movimiento del mismo. También es bastante común usar la rueda del ratón con o sin tecla modificadora para controlar el zoom de la cámara. Normalmente esta opción puede desactivarse para facilitar la interacción del programa con tableta gráfica. En el caso de Zbrush, la cámara se controla haciendo clic es espacios del viewport en los que no hay ningún objeto dibujado (figura 3.7). Esto se debe a que es un programa que se

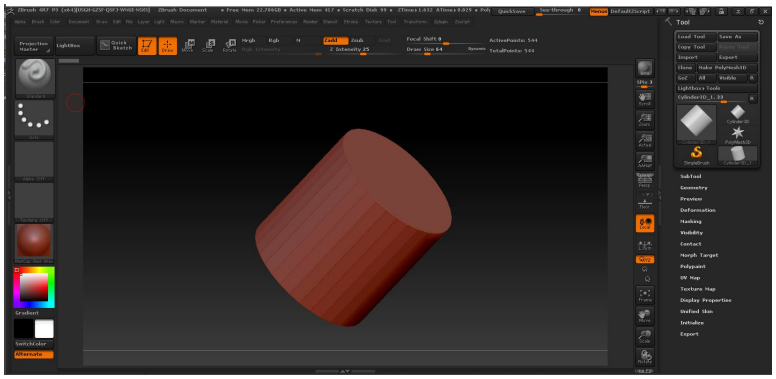

Figura 3.7: Interfaz principal de Zbrush con una Tool cilindro

usa principalmente con tableta gráfica, por lo que es mucho más rápido interactuar con la posición de la cámara de esta forma.

En todos los softwares también se pueden modificar propiedades de la cámara como su su tipo de perspectiva o campo de visión, así como poder activar múltiples vistas simultaneas. También se permite restablecer la posición de la cámara para que el origen de coordenadas de la escena quede en el centro de la pantalla o centrar la cámara directamente sobre uno de los objetos.

A pesar de que este tipo de interacción ya esté muy establecida en el mundo de la edición 3D, para este sofware puede ser necesario una modificación para facilitar su uso con el Leap Motion. Dichas modificaciones se basan en las siguientes puntos:

- A pesar de que un usuario espera mover la cámara cuando está interactuando mediante un dispositivo señalador (ratón o tableta gráfica), esta no es la situación esperada cuando el usuario necesita mover la mano en el espacio para interactuar con la vista de la escena. Las personas están acostumbradas a que los objetos que está en sus manos se muevan en relación a su vista cuando desplazan sus manos por el espacio, no a que el objeto se mantenga fijo y su punto de vista sea modificado. Por lo tanto, la aplicación se diseñará de forma que los objetos sean siempre los que se mueven en relación a la cámara, no la cámara en relación a los objetos.

- No se incluirá la opción de alterar el campo de visión de la cámara. En este programa es de vital importancia que el usuario mantenga una imagen 3D mentalmente de la escena para poder interactuar de forma cómoda con la misma, por lo que alterar parámetros de la perspectiva puede confundirlo a la hora de calcular distancias y relaciones de tamaño entre objetos.

- No se incluirá una opción para activar una vista ortográfica. Al estar usando un dispositivo de entrada que permite el uso de las tres dimensiones no tendría sentido eliminar la información de uno de los ejes en el viewport.

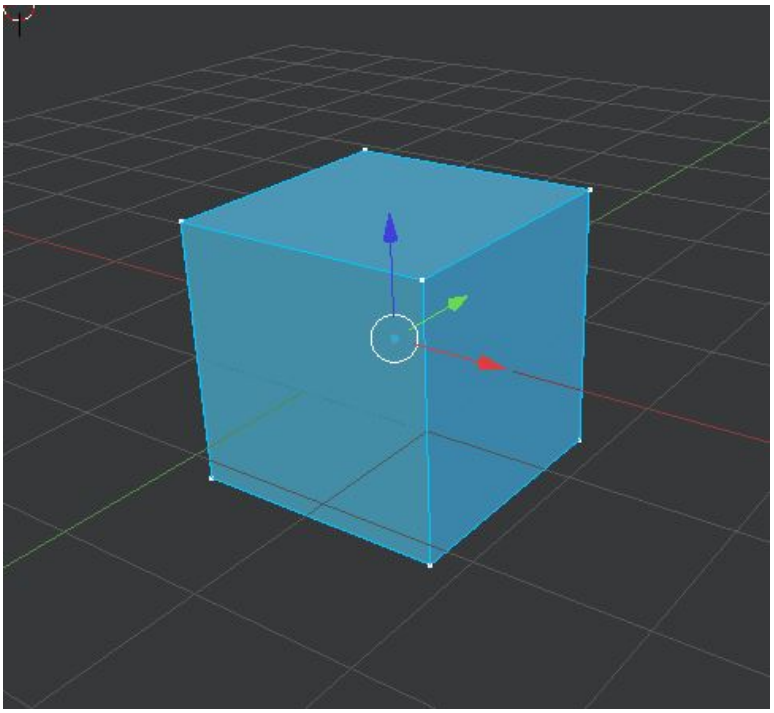

Figura 3.8: Blender en modo objeto con los gizmos del operador grab visible.

3.3.2. Interacción

La mayoría de estas aplicaciones se basan en el uso de gizmos para la transformación de elementos en la escena. Estos son pequeñas indicaciones en forma de flechas que realizan la transformación deseada al seleccionarlas y arrastrarlas con el ratón (figura 3.8). A pesar de que este método aporta una gran precisión a la hora de realizar transformaciones, en un entorno de interacción con un Leap Motion presentaría las siguientes desventajas:

- El uso de gizmos está basado en el uso de un dispositivo señalador para interactuar con los mismos. No sería problema implementar un cursor en el espacio 3D que siguiera los movimientos de la mano, pero sería muy poco gratificante para el usuario tener que seleccionar un elemento en un espacio 3D de un tamaño relativamente pequeño cada vez que tiene que realizar una tansformación.

- El tamaño de los gizmos, su representación en 3D y la representación de los elementos como el cursor tendrían que tener un tamaño y unas características que los harían poco prácticos a la hora de visualizar el resto de la escena.

- El uso de gizmos limita las transformaciones a uno o dos ejes simultáneos. Esto es útil en el caso del uso del ratón ya que como máximo se pueden controlar dos ejes a la vez. Usando este tipo de interacción con un dispositivo como el Leap Motion en el que se permite una entrada de todos los ejes simultáneos sería no aprovechar sus posibilidades.

Figura 3.9: Viewport de la aplicación mostrando a Suzane

- El uso de un gizmo para modificar las transformaciones de los objetos no está extendido fuera de los programas de edición 3D. Las personas están acostumbradas a usar sus manos para poder manipular los objetos en el mundo real y ver sus transformaciones acorde al movimiento de las mismas.

Por lo tanto, teniendo en cuenta los paradigmas de interacción definidos anteriormente y las conclusiones del primer sprint, el funcionamiento del viewport de esta aplicación será el siguiente:

- Los objetos y la escena se moverán siempre en relación a la cámara.
- La escena solamente se moverá cuando el usuario tenga su mano derecha cerrada sobre el sensor, en cuando el usuario abra la mano la escena quedará bloqueada. Esto hará que el usuario perciba el movimiento que realiza como si estuviera agarrando un objeto en el mundo real.
- El movimiento será siempre relativo a la mano del usuario, no al objeto. El usuario no deberá tener la mano en ninguna posición en concreto para mover la vista del viewport, por lo que puede desplazarse por el mismo con mayor velocidad y sin estar pendiente de la posición de su mano (de la misma forma en la que se puede mover la posición de la cámara en cualquier software 3D independientemente de la posición del ratón). Todos los movimiento se calcularán respecto a la posición en la que la mano se ha cerrado por primera vez.

Para resolver el movimiento del viewport la solución es directa: se aplica el vector resultante de restar la posición final a la inicial de la mano derecha (última posición de la mano abierta menos la primera posición de la mano cerrada) al origen de la escena. Si fuera necesario, este vector se escalará para compensar la escala por defecto de los objetos en el programa.

Este tipo de movimiento resuelve tanto el panning como el zoom de un viewport de un software 3D tradicional, ya que para acercar un objeto a la vista el usuario

solamente tendrá que mover la mano hacia sí mismo como si lo estuviera agarrando en el mundo real, dando lugar a una interacción más natural.

A pesar de que el sensor Leap Motion aporta una clara ventaja en este tipo de movimientos, las rotaciones no tienen una solución directa que produzca una sensación agradable al usar el software. Este problema se ha detectado en los primeros prototipos realizados durante el tercer sprint, dando lugar a las siguientes opciones:

- Copiar la rotación a la escena en espacio local: Con esta configuración, la primera rotación que realice el usuario al rotar su mano se realizará correctamente, ya que por defecto los ejes de la mano y del objeto están alineados al no haberse realizado ningún giro previamente. La ventaja de usar el espacio local es que los ejes de la mano se orientarán automáticamente al objeto, no teniendo en cuenta la rotación relativa entre la mano y el sensor (el eje z de la mano será siempre el eje z del objeto al principio de la rotación, independientemente de la rotación en la que se encuentren).

  Al comportarse de esta manera, las rotaciones se acumulan. Una vez rotado el objeto, la nueva rotación se aplicará sobre su rotación actual y ya no corresponderá con lo que el usuario espera ver realizando su rotación con la mano. Esto da lugar a situaciones en las que si un objeto se encuentra rotado 180 grados en el eje Y (con el eje Z apuntando en dirección contraria al eje Z de la mano del usuario), el objeto rotará en sentido contrario, como si se viera reflejado en un espejo.

- Copiar la rotación a la escena en espacio global: Con esta configuración, los objetos rotan siempre de forma consistente, independientemente de la rotación que ya tengan. El problema de rotar en espacio global es que se tiene en cuenta la rotación de la mano con respecto al sensor, por lo que para que la rotación sea la esperada el usuario deberá tener la mano perfectamente alineada con los ejes del sensor. Esto es algo que raramente pasa, ya que en la mayoría de los casos no se comprueba la orientación del sensor antes de realizar la acción, y cualquier error en la alineación entre el sensor y la mano hace que la rotación empiece a realizarse desde un ángulo incorrecto. Además produce demasiadas inconsistencias, ya que es muy difícil tener la mano siempre en la misma orientación inicial a la hora de realizar la rotación. Los usuarios están familiarizados con este comportamiento en cualquier método de interacción (el ratón se mueve el cursor teniendo en cuenta sus coordenadas locales, no depende de la orientación entre el ratón y la pantalla o entre el ratón y la alfombrilla).

Para solucionar este problema ha sido necesario diseñar el siguiente proceso:

- La rotación se realiza en espacio local, de esta forma la orientación inicial de la mano siempre coincide con la orientación del objeto, sin tener en cuenta la orientación del sensor. Esto es lo que el usuario espera.

- Una vez realizada la rotación, se computa un nuevo espacio para el objeto de forma que su rotación se conserve, pero sus coordenadas globales vuelvan a ser las iniciales.

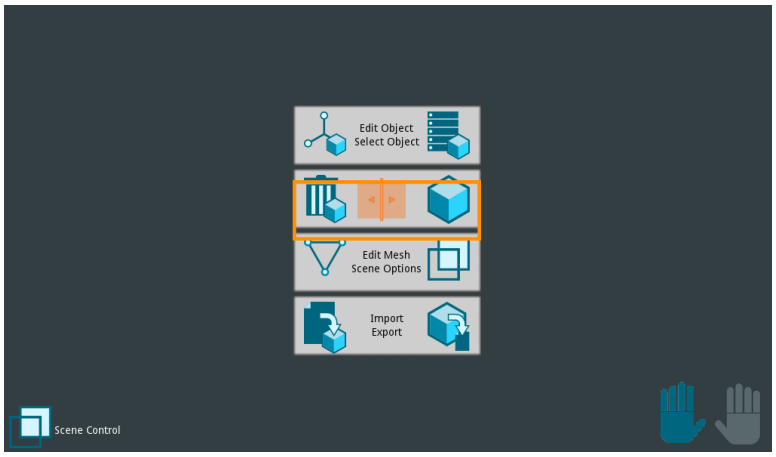

Figura 3.10: Menú principal de la aplicación

- Al realizar una nueva rotación, esta funcionará igual que la primera realizada en espacio local, pero sobre un objeto alterado ya rotado previamente.

3.4. Sprint 4: Menús

El menú de la aplicación es el lugar desde el cual se accede a las diferentes opciones y se lanzan los operadores para editar el contenido de la escena (figura 3.10).

Siguiendo las conclusiones de los apartados anteriores, el menú se han diseñado para que su interacción con el Leap Motion sea lo más satisfactoria posible en el cuarto sprint del proyecto. Para ello, se ha evitado en todo momento el uso de cursores. En su lugar se ha establecido el siguiente funcionamiento:

- Al cerrar la mano izquierda se muestra el menú, al abrirla el menú desaparece. Cuando se abre la mano el menú se cierra sin haber seleccionado ningún operador, de esta forma el usuario puede cancelar la acción en cualquier momento, de la misma manera que puede parar un movimiento del viewport abriendo la mano derecha.
- Los movimientos verticales de la mano izquierda cerrada sirven para cambiar de opción resalta en el menú.
- Los movimientos horizontales de la mano izquierda cerrada sirven para seleccionar opciones del menú o seleccionar submenús (figura 3.11).
- El acceso a una opción del menú deberá ser lo más breve posible, de forma que se pueda acceder a la opción deseada con el mínimo movimiento.

Atendiendo los puntos mencionados se ha diseñado el widget para las opciones del menú. Este widegt cuenta con las siguientes características:

- Su recuadro de selección responde al movimiento vertical de la mano, permitiendo seleccionar otros widgets que se encuentren encima o debajo del mismo.

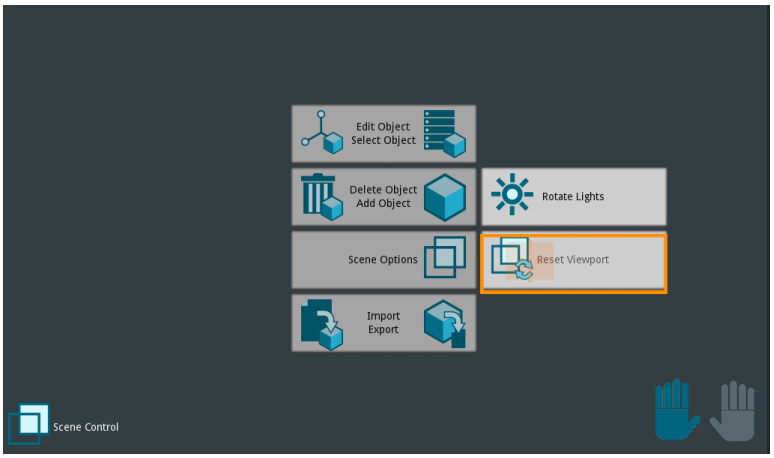

Figura 3.11: Submenú de opciones de la escena de la aplicación

- Su deslizador responde al movimiento horizontal de la mano.
- Contiene dos opciones por widget, reduciendo la longitud de los menús.
- Responde a movimientos sutiles de la mano del usuario, de forma que este puede identificar el movimiento que está realizando antes de finalizarlo y confirmar su selección.

Además, el menú se ha diseñado de forma que el mismo gesto en el espacio resulte siempre en la misma opción seleccionada. Una vez que un usuario se sienta cómodo con la aplicación no tendrá que leer las opciones del menú para poder llegar hasta ellas, ya que, por ejemplo, el gesto cerrar mano, mover a la derecha mover a la izquierda, abrir mano equivale siempre a seleccionar todos los objetos, independientemente de donde se encuentre la mano izquierda en ese momento.

3.5. Sprint 5 - 8: Operadores

Durante los sprints 5, 6, 7 y 8 será necesario implementar la mayor parte de funcionalidad de la aplicación. En el sprint 5 se definirán los principios base para implementar todas estas funcionalidades, dejando los sprints restantes para terminar la implementación de las mismas. Para ello, el primer paso es decidir una forma para organizar dichas funcionalidades de forma que facilite el desarrollo y sea intuitivo para el usuario.

A la hora de organizar y establecer el funcionamiento de las diferentes herramientas de edición que incorporan los software de edición en 3D se han seguido diferentes enfoques. La gran mayoría de softwares disponibles en el mercado usan el concepto de herramienta, cuyas características son las siguientes:

- El usuario selecciona una herramienta. Esta se mantiene activa hasta que el usuario selecciona una herramienta diferente (figura 3.12).

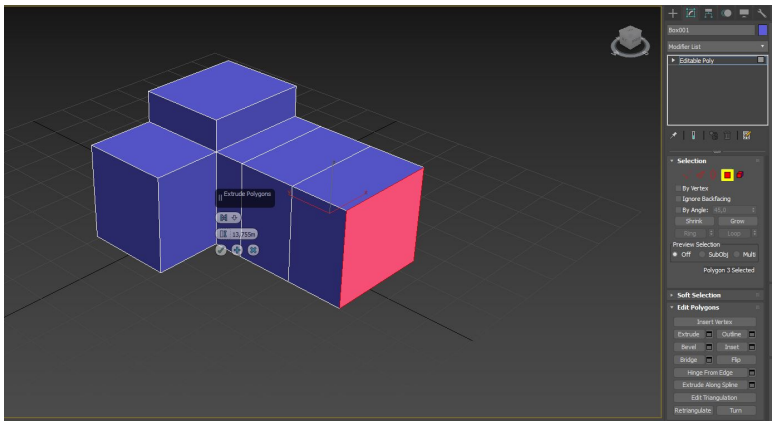

Figura 3.12: 3DS Max con la herramienta extrude activa

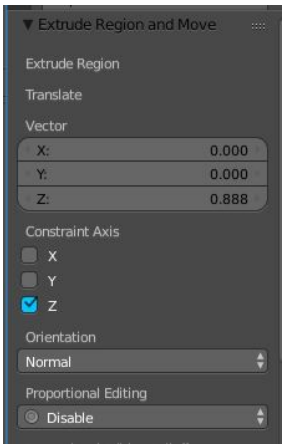

Figura 3.13: Panel de opciones del operador extruir de Blender

- El usuario realiza las ediciones correspondientes en la escena usando la herramienta activa, por lo que el comportamiento de la interfaz del programa depende directamente de la herramienta activa.
- El software suele tener una herramienta activa por defecto, generalmente la herramienta de selección.

Por otra parte está el enfoque de Blender. En este software las herramientas son llamadas operadores (figura 3.13). Se diferencian de las herramientas de los softwares tradicionales en los siguientes puntos:

- El software se encuentra siempre en el mismo estado cuando no hay ningún operador activo, respondiendo siempre a la entrada del usuario de la misma forma.
- Cuando se activa un operador, el comportamiento del software se altera para poder realizar la edición. Esta alteración dura hasta que la edición finaliza, posteriormente el software vuelve a su estado por defecto.

- Para continuar realizando ediciones del mismo tipo es necesario seleccionar el operador de nuevo.

Este enfoque puede no parecer intuitivo a primera vista ya que se necesita realizar una selección de un operador cada vez que se requiere modificar algún elemento del espacio de trabajo. No obstante cuenta con las siguientes ventajas:

- El usuario no tiene que memorizar la herramienta que tiene activa mientras trabaja con el software.
- En un proceso de modelado 3D es muy raro usar una herramienta más de 3 veces seguidas de forma que compense bloquear el software en un estado en el que solo se permita su uso.
- Los operadores pueden activarse rápidamente mediante atajos de teclado, pudiendo substituir el clic de selección de una herramienta por la pulsación del atajo que inicia el operador.

A la hora de implementar este software se usará el enfoque de los operadores. La aplicación tendrá un estado por defecto sobre el cual se podrán activar operadores para realizar las ediciones correspondientes. Una vez finalice la edición el software volverá a su estado por defecto.

3.5.1. Escena operator

La escena operator será la base para la implementación de los operadores en este software.Es la escena base para todos los operadores que permiten realizar ediciones en el programa. Todos los operadores deberán implementar los siguientes métodos en su script de su nodo principal (que también hereda del script operator).

- begin(params): inicia el operador con determinados parámetros.
- process(delta): en caso de que el operador necesite más de un frame para aplicarse, la actualización de su estado se realizará en este método teniendo en cuenta el parámetro delta.
- end(): finaliza el operador, haciendo los ajustes que sean necesarios en la escena antes de ser eliminado.

Además este script ha de implementar una forma de acceder al viewport para que sea más sencillo realizar modificaciones sobre el mismo.

La escena operator también contiene un nodo que almacena el icono y el nombre del operador, que se mostrará en la barra de estado una vez el operador esté instanciado y activo.

El caso de uso más común es iniciar un operador. En la figura 3.14 se ilustra el funcionamiento de la aplicación y la interacción de sus componentes a la hora de realizar esta acción.

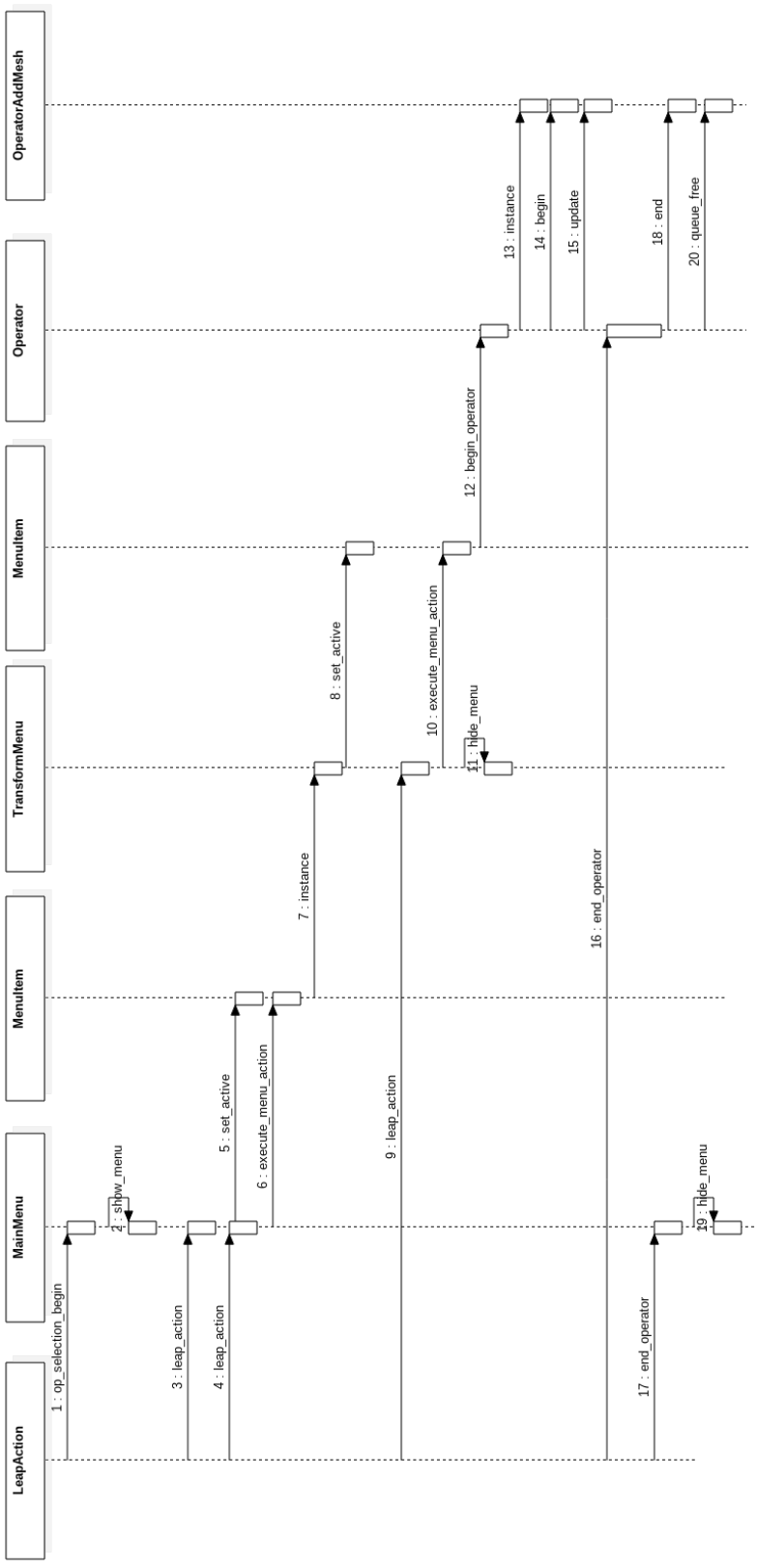

Figura 3.14: Diagrama de secuencia para la ejecución de un operador

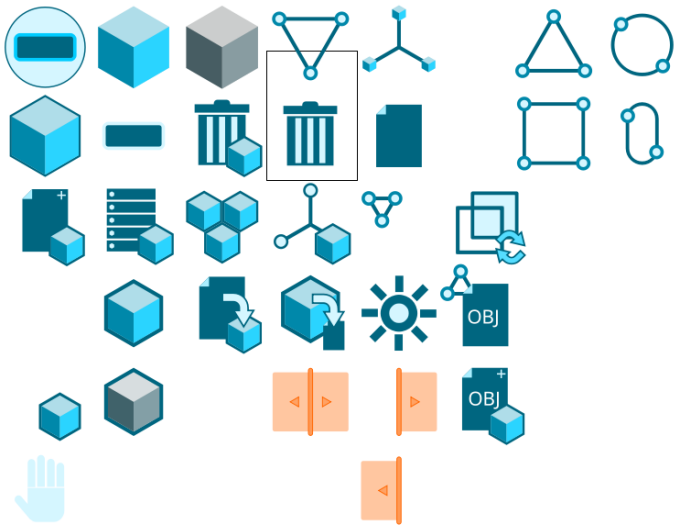

Figura 3.15: Iconos diseñados para la aplicación

3.6. Sprint 9: Iconos

La aplicación cuenta con unos iconos diseñados específicamente para la misma (figura 3.15) que se han realizado en el último sprint. Dichos iconos se han basado en los conceptos de los iconos de Blender, de forma que los usuarios familiarizados con este software pueden identificarlos más fácilmente. Los iconos se han realizado en Inkscape.

También se incluyen iconos para mostrar el estado de las manos (abierta, cerrada o ausente) y así poder detectar más fácilmente errores en el sensor.

Capítulo 4

Implementación

En este capítulo se tratará la implementación del software en Godot Engine, atendiendo a las particularidades de este motor. También se explicarán las técnicas y algoritmos utilizados para cumplir los requisitos funcionales del proyecto.

Contents

4.1. Sprint 2: Interfaz de Leap Motion

A lo largo del Sprint 1 del proyecto se han realizado las tareas correspondientes al análisis y diseño, comenzando su implementación en el Sprint 2. El principal objetivo de este sprint es implementar la interfaz con el sensor Leap Motion.

Para poder acceder a los datos del sensor Leap Motion es necesario usar su SDK. Dicho SDK se encuentra disponible en `https://developer.leapmotion.com/sdk/v2`.

En este proyecto se usará la versión 2 del SDK y no la más reciente, ya que esta no es compatible con Linux. El SDK de Leap Motion es compatible con diferentes lenguajes, como Python, Java, C# o C++. A pesar de que Godot Engine 3 es compatible con C# esta funcionalidad todavía es experimental y no permite la exportación del proyecto. Por lo tanto este proyecto se realizará usando la API de C++ y creando una biblioteca de GDNative que Godot podrá cargar de forma dinámica.

4.1.1. GDNative

GDNative es un módulo añadido a Godot Engine en su versión 3.0 que permite el uso de bibliotecas compiladas como scripts para el motor. Como se había mencionado en el capítulo anterior, cada nodo puede contener un script que lo extiende, añadiendo nuevos métodos y propiedades a las de la clase base del mismo. Compilando una clase de C++ desde la que se llame a la API de C++ del Leap Motion y añadiéndolo a Godot como un script mediante GDNative se podrá acceder a los métodos de la API desde otros scripts del programa.

Para realizar este proceso en primer paso es necesario obtener los headers y los bindings de Godot 3 para C++ desde sus respectivos repositorios:

- `https://github.com/GodotNativeTools/godot_headers.git`
- `https://github.com/GodotNativeTools/cpp_bindings.git`

A continuación se crea un archivo .cpp con código C++ desde el que se llamará a la API de Leap Motion;

```
using namespace godot;

class LeapMotion : public GodotScript<Reference> {
GODOT_CLASS(LeapMotion);
public:

        Leap::Controller controller;

        Leap::Frame get_frame(){
                Leap::Frame frame = controller.frame();
                return frame;
        }
```

```
        bool is_leap_connected() {
                return controller.isConnected();
        }

}
```

Dentro de la misma clase se registrarán los métodos que se declaran para que Godot pueda reconocerlos una vez cargada la biblioteca:

```
static void _register_methods() {
        register_method("is_leap_connected",
        &LeapMotion::is_leap_connected);
}
```

Por último se añaden los métodos necesarios para el inicio, destrucción y registro de la clase que se acaba de crear.

```
/** GDNative Initialize **/
GDNATIVE_INIT(godot_gdnative_init_options *options) {

}

/** GDNative Terminate **/
GDNATIVE_TERMINATE(godot_gdnative_terminate_options *options) {

}

/** NativeScript Initialize **/
NATIVESCRIPT_INIT() {
        register_class<LeapMotion>();
}
```

A continuación se compila el archivo de C++ creado con los siguientes comandos, enlazando la biblioteca del LeapMotion:

```
$clang -fPIC -o src/init.os -c src/init.cpp -g -O3
-std=c++14 -Icpp_bindings/include -Igodot_headers -Ileap_headers

$clang -o lib/libtest.so -shared src/init.os
-Llib -lgodot_cpp_bindings lib/x64/libLeap.so
```

El siguiente paso es crear el archivo leapmotion.gdns. Dicho archivo contiene información para el motor sobre el directorio del código compilado y sus propiedades para poder usarlo como script. Además, también puede especificar diferentes bibliotecas compiladas para diferentes plataformas. La manera normal de crear este archivo es hacerlo mediante la interfaz gráfica pero en la versión beta del editor esta funcionalidad todavía no estaba disponible por lo que ha sido necesario crearlo a mano.

```
[gd_resource type="NativeScript" load_steps=2 format=2]
```

```
[ext_resource path="res://leapmotion.gdnlib"
type="GDNativeLibrary" id=1]

[resource]

resource_name = "LeapMotion"
class_name = "LeapMotion"
library = ExtResource( 1 )
```

Posteriormente dicha biblioteca de GDNative se instancia desde GDScript en la clase Leap indicando la ruta del archivo gdns creado previamente. Esta clase se añadirá a las clases globales del programa, pudiendo acceder a sus métodos de forma estática y por lo tanto a los datos del sensor desde cualquier parte de la aplicación.

```
var leap

func _ready():
leap = load("res://leapmotion.gdns").new()

func get_hand_transform(var hand):
  return leap.get_hand_transform(hand)

func is_hand_visible(var hand):
  return leap.is_hand_visible(hand)

func get_hand_status(var hand):
  return leap.get_hand_status(hand)

func is_leap_connected():
  return leap.is_leap_connected()
```

4.2. Sprint 3: Viewport

Durante este sprint se implementa el viewport del programa, necesario para mostrar los objetos por pantalla y poder verlos desde diferentes ángulos.

La funcionalidad del viewport está implementada mediante la escena Viewport y su script. El viewport es responsable de mostrar los objetos, configurar sus modos de visualización, su iluminación y del movimiento de la cámara. Su árbol de nodos está especificado en la figura 4.1.

- El nodo WorldEnviroment controla las opciones de los efectos del renderizador y el color de fondo del viewport. El programa no tiene ningún efecto aplicado para aumentar el rendimiento.
- El nodo FixedScene almacena como hijos nodos de partes del modelo no seleccionadas para que queden bloqueadas ante cualquier tipo de transformación.

Figura 4.1: Árbol de nodos de la escena Viewport

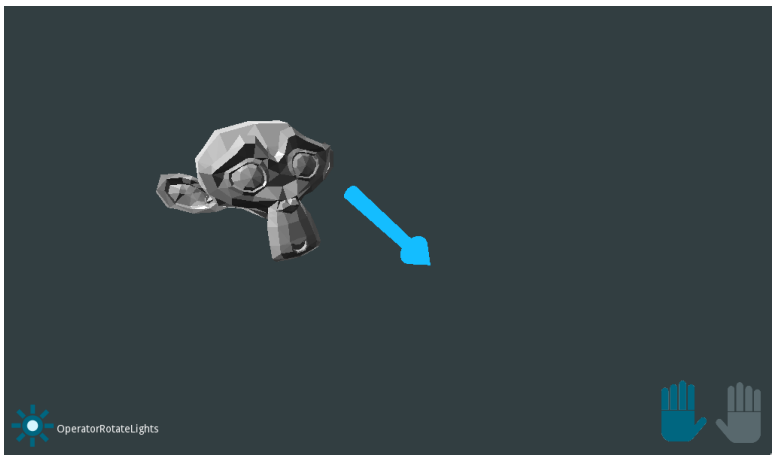

Figura 4.2: Iluminación de la escena siendo editada con el operador RotateLights

Cuando se quiere bloquear un nodo que contenga una malla, este se copia como hijo de FixedScene y se borra como hijo de Model.

- El nodo Lights contiene la iluminación de la escena, en este caso una única luz direccional. También contiene una malla con un gizmo para indicar su dirección al manipularla. Los nodos Gizmo y DirectionalLight son hijos de DirLightA para poder manipular su transformación conjuntamente (figura 4.2).

- El nodo Scene almacena la trasformación de la escena mientras se manipula usando la mano derecha sobre el Leap Motion. Una vez finalizada la transformación, esta se copia al nodo Model y se restablece su posición inicial, haciendo que nuevas transformaciones se realicen en espacio local con las coordenadas iniciales.

  En el siguiente fagmento de código se actualiza la matriz de transformación de nodo Scene calculando la diferencia entre las dos últimas transformaciones de la mano.

```
Scene.transform.origin = $Scene.transform.origin + (
   current_hand_transform.origin -
   last_hand_transform.origin);
var current_rotation = Quat( $Scene.transform.basis)
var last_hand_rotation = Quat(last_hand_transform.
   basis)
var current_hand_rotation = Quat(
   current_hand_transform.basis)
var rotation_diference = current_hand_rotation.
   inverse() * last_hand_rotation
var final_rotation = current_rotation*
   rotation_diference.inverse()
$Scene.transform.basis = Basis(final_rotation)
```

A continuación, una vez abierta la mano y confirmada la transformación, esta se aplica al nodo Model y se restablecen las bases de la matriz del nodo Scene.

```
var scene_rotation = Quat($Scene.transform.basis)
var model_rotation = Quat($Scene/Model.transform.
   basis)
$Scene/Model.transform.basis = Basis(scene_rotation*
   cube_rotation)
$Scene.transform.basis = Basis()
```

Este nodo también restablece automáticamente su posición en caso de que el nodo Model no contenga ningún hijo, de forma que cuando se añade un objeto a una escena vacía este se sitúa siempre en el centro de la pantalla. Los materiales de las mallas de los objetos también se ajustan en el script de la escena haciendo que los modelos bloqueados tengan un material transparente (figura 4.3).

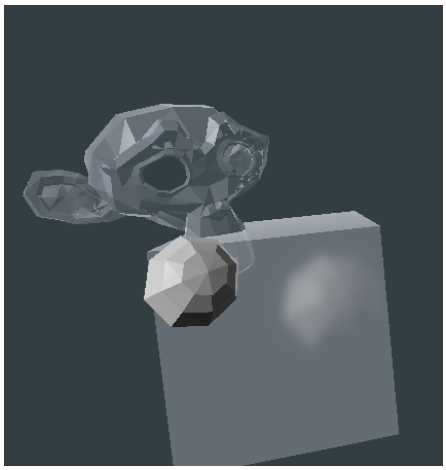

Figura 4.3: Modelo siendo editado con un único objeto activo. El resto se muestran transparentes.

4.3. Sprint 4: Menús

La escena MainMenu implementa el menú principal de la aplicación desde el cual se puede acceder a la mayor parte de su funcionalidad. Su árbol de nodos se muestra en la figura 4.4 Las principales funciones de esta escena son las siguientes:

- Procesar la entrada de la mano izquierda capturada por el LeapMotion y traducirla a eventos de movimiento.
- Mostrar y ocultar el menú principal en función del estado de la mano izquierda.
- Establecer unas reglas para la distribución de los widgets del menú, proporcionando contenedores en los cuales los widgets pueden instanciarse.

4.3.1. Navegación

El nodo LeapAction contiene un script responsable de emitir señales cada vez que detecta un movimiento significativo de la mano izquierda en el plano XY respecto a su posición de origen. También es responsable de detectar cuando la mano se abre o cierra para que los menús puedan responder adecuadamente. El nodo LeapAction emite las siguientes señales:

- signal leap_action(dir) : Se emite cuando la mano se mueve en uno de los ejes más de lo especificado en unos límites configurables en el script. Dicha señal incorpora la dirección en la que la mano se ha movido.
- signal op_selection_begin: Se emite cuando se cierra la mano izquierda.
- signal op_finish: Se emite cuando se abre la mano izquierda.

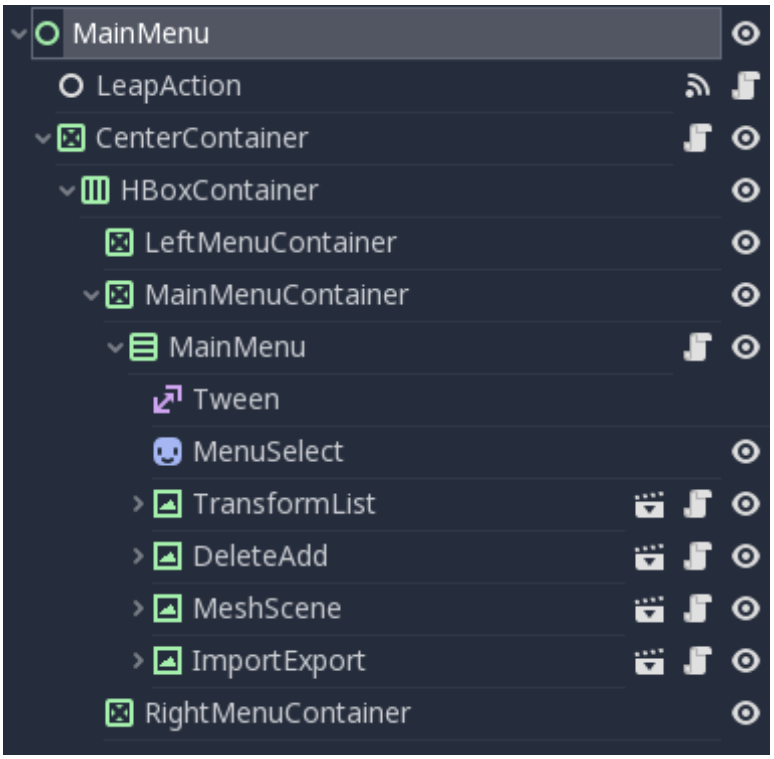

Figura 4.4: Árbol de nodos de la escena MainMenu

Los nodos que heredan de la escena Menu se suscriben a estas señales para determinar su comportamiento. Estas escenas son responsables de guardar la configuración de cada menú, así como de informar a cada widget cuando tiene que ejecutar su acción establecida.

El nodo LeapAction también proporciona una función para obtener el porcentaje de offset respecto a la última posición en la que se ha emitido la señal leap_action. Esto sirve para que otros nodos ajusten ligeramente su posición, proporcionando al usuario una mejor experiencia y usabilidad.

4.3.2. Widgets

El árbol de nodos del widget que representa cada opción del menú se representa en la figura 4.5. Esta escena es la escena base para todas las opciones del menú. Sus funciones son las siguiente:

- Cargar un operador que tenga asignado en el nodo Operator de la escena principal con unos parámetros determinados.
- Mostrar un menú secundario en alguno de los laterales en un contendedor asignado.
- Reproducir las animaciones correspondientes en la interfaz.

Figura 4.5: Árbol de nodos de la escena MenuItem

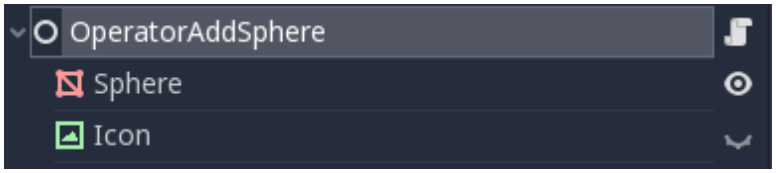

Figura 4.6: Árbol de nodos de un operador

Este nodo contiene gran cantidad de nodos duplicados. Dichos nodos se inician con los parámetros correctos cuando la escena se carga. Estos nodos modulan su opacidad según el offset que se obtiene del nodo LeapAction, proporcionando unas animaciones más suaves en la interfaz y un menú más usable.

4.4. Sprints 5 - 7: Operadores

El código de la gran mayoría de funcionalidades del programa está encapsulado en operadores. En el sprint 5 se implementa el modelo necesario para poder ejecutar un operador dentro del programa, dejando los sprints 6 y 7 para implementar los operadores correspondientes a los requisitos funcionales del mismo.

Todos los operadores del programa heredan de la escena operator. Por esto, todos incluyen un nodo base y un icono para mostrar en la barra de estado. En la figura 4.6 se muestra el árbol de nodos de la escena que implementa el operador AddSphere. La funcionalidad de este operador requiere que se incluya también un recurso de tipo Mesh para que se añada a la escena.

Todos los operadores tienen un script base que hereda del script operator. Proporciona los siguientes métodos:

```
extends Node

onready var viewport = get_tree().get_root().get_node("Main
   /Viewport");
export (String) var export_params

func _ready():
  pass

func begin(params):
  pass

func update(delta):
  pass

func end():
  pass
```

Las funciones begin(), update() y end() representan el ciclo de vida del operador, siendo la función update() la que reemplaza a la función _process() que usa Godot internamente para actualizar los nodos en cada frame.

Los items del menú indican al nodo Operator de la escena principal el operador a instanciar. Esta lo instancia y llama a su método begin(params) con los parámetros correspondientes. Una vez realizado esto, pueden ocurrir dos cosas:

- El operador realiza las operaciones correspondientes y finaliza, solicitando al nodo padre que llame a su función end().
- El operador se mantiene activo llamando en cada frame a su método update(), este finaliza cuando el padre recibe la señal del nodo LeapAction y llana a su método end(), ejecutando en este las acciones necesarias para finalizar el operador satisfactoriamente.

Además, el nodo Operator de la escena Main es responsable de guardar el estado del viewport y añadirlo como hijo al nodo UndoStack antes de llamar al método begin() de cada operador, de forma que cuando se ejecute el operador deshacer este pueda recuperar el último estado y reemplazarlo por el contenido actual del viewport.

4.5. Sprint 8: Algoritmos de edición

La funcionalidad de edición del software está implementada de diferentes formas según el tipo de dato al que afecta (figura 4.7). Los operadores de edición que afectan a objetos usan métodos del motor para hacer estas transformaciones. Por ejemplo, el operador ScaleLocal utiliza la propiedad scale del nodo que contiene la malla para escalarla.

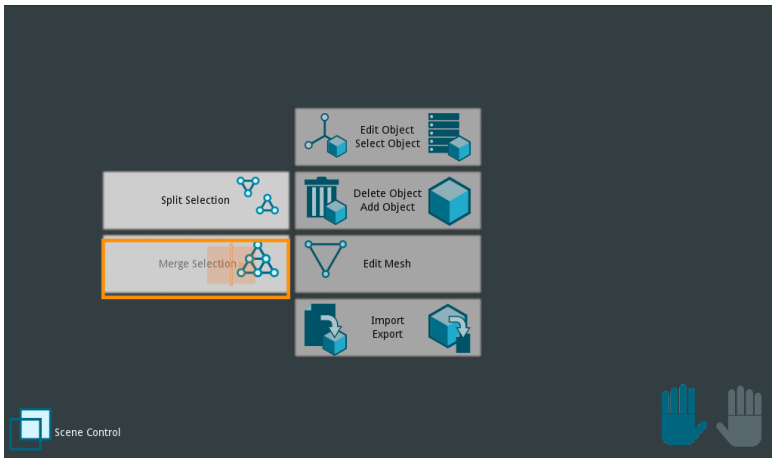

Figura 4.7: Menú mostrando las opciones de algoritmos de edición de la malla

Otro tipo de ediciones que requieren modificaciones en la malla han sido implementadas a parte. Para ello se ha creado el script MeshUtils, que aporta las siguientes funcionalidades:

- Leer y escribir una malla triangular a un archivo Obj.
- Extraer una lista de vértices y caras de un recurso tipo Mesh del motor y crear un recurso Mesh a partir de una lista de vértices y caras.
- Separar y unir listas de vértices y caras por componentes no conectados entre sí.

Dicha funcionalidad está implementada en las siguientes funciones del script:

```
func vertex_data_from_mesh(var mesh, var transform):

func mesh_from_vertex_data(var vertex_data):

func split_vertex_data(var vertex_data):

func join_vertex_data(var vertex_data_array):

func obj_to_vertex_data(var path):

func vertex_data_to_obj(var vertex_data, var path):
```

4.5.1. Conversión entre formatos

Godot almacena las mallas en un recurso interno de tipo Mesh. Este recurso está optimizado para guardar mallas triangulares de forma que el renderizador pueda usarlas con una implementación más sencilla.

Para la creación del recurso Mesh a partir de una lista de vértices y caras Godot facilita la clase SurfaceTool. Esta permite una construcción de la malla similar a la API de OpenGL para el dibujado en pantalla. La clase SurfaceTool también proporciona un método para generar las normales de la malla que se está creando una vez finalizada.

```
var mesh = Mesh.new()
surfTool.begin(Mesh.PRIMITIVE_TRIANGLES)
for face in faces:
surfTool.add_uv(Vector2(0,0))
surfTool.add_vertex(vertex[face.z])
surfTool.add_uv(Vector2(0,0))
surfTool.add_vertex(vertex[face.y])
surfTool.add_uv(Vector2(0,0))
surfTool.add_vertex(vertex[face.x])
surfTool.generate_normals()
surfTool.index()
surfTool.commit(mesh)
```

Para extraer los datos sobre los vértices y caras almacenados en un recurso Mesh Godot proporciona la clase MeshUtils. Está clase no está correctamente documentada ya que no es común usarla en el desarrollo de juegos. Para conseguir la información sobre los vértices y caras se ha consultado la implementación en C++ de la misma, descubriendo que tiene un método para extraer las coordenadas de un vértice dado su índice y otro para extraer el índice del vértices que forma una cara, dado su índice y su posición. Combinando la información de estos dos métodos se puede conseguir reconstruir las listas de vértices y caras con las que se ha creado la malla.

El algoritmo que se usa para reconstruir esta lista combina automáticamente los vértices dobles. Por lo tanto, cualquier edición que se haga sobre una malla con vértices dobles, estos se eliminarán a la primera transformación.

A esta función también se le puede pasar como argumento unas bases, de forma que los vértices se generen con las coordenadas correctas, como se se aplicaran las transformaciones del objeto sobre la malla.

4.5.2. Separación y Unión

La separación de la malla en componentes es el proceso más complejo de la clase MeshUtils. Para realizarse, el programa crea una nueva lista para guardar el estado de cada vértice durante la ejecución del algoritmo (no comprobado o asignado a determinado fragmento de la malla final). Una vez hecho esto, el programa interpreta la malla como un grafo y ejecuta un algoritmo recursivo para encontrar las componentes conexas del mismo. Para ello se utilizan las funciones _get_connected_vertex() y _set_vertex_status().

Una vez identificadas las componentes conexas y habiendo guardando la información sobre la componente a la que pertenece cada vértice en la lista de estados, dicha lista es usada para reconstruir diferentes mallas en función de estos índices.

Este proceso puede causar pérdida de frames en la visualización del viewport ya que la máquina de bytecodes en la que se ejecuta el código de GDSCript no está optimizada para el uso de funciones recursivas (la llamada a una función de GDScrit es el proceso que más penaliza al movimiento del motor). Si llegado a un punto este rendimiento no es suficiente el programa puede actualizarse ejecutando el mismo algoritmo en C# o C++, codificándolo de nuevo y siguiendo los pasos explicados en el apartado anterior.

La unión es un proceso directo, simplemente se concatenan las listas de vértices y caras:

```
var final_vertex = []
var final_faces = []
var offset = 0
for vertex_data in vertex_data_array:
for vertex in vertex_data.vertex:
final_vertex.append(vertex)
for face in vertex_data.faces:
final_faces.append(Vector3(face.x + offset, face.y + offset
   , face.z + offset))
offset += vertex_data.vertex.size()
return {"vertex": final_vertex, "faces":final_faces}
```

4.5.3. Lectura y escritura

La lectura y escritura del programa se hace en formato .obj. Se ha elegido este formato porque es una representación directa de la estructura de lista de vértices y de caras que se usa en el resto del programa para tratar con los datos de una malla.

La lectura y escritura de datos en formato .obj es trivial, simplemente se escribe o lee linea a linea el contenido del archivo siguiendo los contenidos de las listas, ayudándose de la API de Godot para la lectura y escritura de archivos:

```
file.open(path,File.WRITE)
for v in final_vertex:
file.store_line("v " + str(v.x) + " " + str(v.y) + " " +
   str(v.z))
for f in final_faces:
file.store_line("f " + str(f.x+1) + "// " + str(f.y+1) + "
   // " + str(f.z+1) + "//")
file.close()
```

En el formato .obj, los vértices se preceden con una "v" las caras con una "f". No se incluyen las normales de los vértices ya que la mayoría de programas 3D son capaces de recalcularlas a la hora de importar un archivo, pero su inclusión sería trivial. El índice de los vértices se incrementa en 1 ya en en el formato .obj las listas empiezan en 1, no en 0.

Figura 4.8: Árbol de nodos de la escena StatusBar

4.6. Sprint 9: Barra de estado

En el último sprint se implementa la barra de estado del programa. La barra de estado representa el operador que se está ejecutando y el estado del LeapMotion. Su árbol de nodos se representa en la figura 4.8.

Esta escena contiene un script que oculta o muestra sprites en función del estado que obtenga de la clase Leap y del operador instanciado como hijo del nodo Operator de la escena Main. También puede instanciar mensajes de error o información para el usuario como hijos del nodo HUD, situándose estos en la esquina inferior derecha de la pantalla.

Capítulo 5

Pruebas

En este capítulo se documentan las pruebas que se han realizado para comprobar el estado de la aplicación a lo largo de su desarrollo, así como las pruebas finales usadas para verificar que la aplicación cumple con los requisitos establecidos en el análisis del proyecto.

Las pruebas se han ido diseñando para evaluar el desempeño del trabajo en cada sprint, por lo que las comprobaciones son incrementales.

Contents

5.0.1. Pruebas de sprints

En esta sección se muestran las pruebas que se han diseñado en cada uno de los sprints del proyecto para evaluar el trabajo realizado en cada uno de ellos.

Sprint 1

Prueba 1

- Nombre: Frames por segundo
- Descripción: Comprobar que el motor elegido puede llegar a los frames por segundo especificados.
- Requisitos involucrados: RN11
- Validación: Cargar una malla de 1M de polígonos en el motor y crear una escena con esta y una cámara. Ejecutar el proyecto y comprobar que supera los 60fps en un ordenador con una GTX1080 y un i7 6700K.
- Estado: Completado

Prueba 2

- Nombre: Resolución
- Descripción: Comprobar que el motor elegido es compatible con una resolución de 600 x 1024.
- Requisitos involucrados: RN4
- Validación: Ejecutar un proyecto vacío en el motor forzando al resolución a 600 x 1024.
- Estado: Completado

Sprint 2

Prueba 3

- Nombre: Detección del sensor
- Descripción: Comprobar que el sensor es detectado por el ordenador.
- Requisitos involucrados: RF1, RF2
- Validación: Cuando se llama al método Leap::is_connected de la API de Leap Motion con el sensor conectado al ordenador, este devuelve true.
- Estado: Completado

Prueba 4

- Nombre: Detección de estados de la mano
- Descripción: Comprobar que el sensor puede diferenciar una mano abierta de una cerrada.
- Requisitos involucrados: RF2
- Validación: Al llamar al método get_hand_status() este devuelve 0 o 1 dependiendo de si la mano derecha está abierta o cerrada. 0 en caso de estar cerrada, 1 en caso de estar abierta.
- Estado: Completado

Prueba 5

- Nombre: Detección de posición de la mano
- Descripción: Comprobar que el sensor puede detectar las posiciones de las manos.
- Requisitos involucrados: RF1
- Validación: Se verifican los valores de la matriz de transformación que devuelve la API de Leap Motion desde el motor de desarrollo.
- Estado: Completado

Sprint 3

Prueba 6

- Nombre: Alteración de la posición del objeto a visualizar en el viewport
- Descripción: Comprobar que se puede mover un objeto del viewport mediante el sensor.
- Requisitos involucrados: RF3, RF4, RF9
- Validación: Se verifica que el movimiento del objeto en el viewport en su espacio 3D corresponde con el movimiento de la mano.
- Estado: Completado

Prueba 7

- Nombre: Alteración de la rotación del objeto a visualizar en el viewport
- Descripción: Comprobar que se puede rotar un objeto del viewport mediante el sensor.
- Requisitos involucrados: RF3, RF4, RF10
- Validación: Se verifica que la rotación local del objeto corresponde con la de la mano sobre el sensor.
- Estado: Completado

Sprint 4

Prueba 8

- Nombre: Detección del movimiento de desplazamiento
- Descripción: Comprobar que se detectan correctamente las cuatro direcciones de movimiento para navegar por el menú.
- Requisitos involucrados: RN6
- Validación: Desplazar la mano sobre el sensor hacia arriba, abajo, izquierda derecha y comprobar que se emiten los eventos adecuados a cada una de estas acciones.
- Estado: Completado

Prueba 9

- Nombre: Cambio de estado del menú
- Descripción: Permitir ocultar y mostrar el menú con la mano izquierda.
- Requisitos involucrados: RN6
- Validación: El menú se muestra cuando se cierra la mano izquierda y se oculta cuando se abre.
- Estado: Completado

Sprint 5

Prueba 10

- Nombre: Ejecución de operador
- Descripción: Comprobar que el programa puede ejecutar un operador correctamente.
- Requisitos involucrados: RF5, RF6, RF8, RF11, RF12, RF13, RF14, RF15, RF16, RF17, RF18
- Validación: Se crea un operador que imprima "hello world"por cosola al ejecutarse y se comprueba que aparece el mensaje.
- Estado: Completado

Sprint 6

Prueba 11

- Nombre: Añadir objeto al la escena
- Descripción: Asegurarse de que los objetos se añaden a la escena correctamente.
- Requisitos involucrados: RF12
- Validación: Al ejecutar el operador de añadir cubo, un cubo es añadido a la escena y listado como objeto.
- Estado: Completado

Prueba 12

- Nombre: Borrar objeto de la escena
- Descripción: Asegurarse de que los objetos se elimina del viewport correctamente.
- Requisitos involucrados: RF12, RF14
- Validación: Al ejecutar el operador de eliminar objeto eligiendo un cubo creado previamente de la lista este se elimina de la lista de objetos y del viewport.
- Estado: Completado

Prueba 13

- Nombre: Borrar todo
- Descripción: Comprobar que se pueden eliminar todos los objetos del viewport.
- Requisitos involucrados: RF15
- Validación: Al ejecutar el operador de eliminar todo la escena queda vacía, así como la lista de objetos.
- Estado: Completado

Sprint 7

Prueba 14

- Nombre: Escalar objeto
- Descripción: Comprobar que se pueden escalar los objetos de la escena.
- Requisitos involucrados: RF11
- Validación: Con un objeto seleccionado, ejecutar el operador escalar y jugar o separar las manos cerradas mientas este está activo. El objeto deberá escalarse.
- Estado: Completado

Prueba 15

- Nombre: Rotar luz
- Descripción: Comprobar que se pueden rotar la iluminación de la escena.
- Requisitos involucrados: RF6
- Validación: Al ejecutar el operador RotateLight comprobar que la rotación de la luz de la escena corresponde con la rotación de la mano.
- Estado: Completado

Sprint 8

Prueba 16

- Nombre: Leer OBJ
- Descripción: Comprobar que se almacena correctamente en el programa el contenido de un archivo .obj.
- Requisitos involucrados: RF7
- Validación: Crear un cubo en Blender, exportarlo como malla triangular a .obj y cargarlo en el programa. Comprobar las coordenadas de los vértices resultantes y los índices de las caras.
- Estado: Completado

Prueba 17

- Nombre: Escribir OBJ
- Descripción: Comprobar que se puede escribir a un archivo el estado de la escena.
- Requisitos involucrados: RF8
- Validación: Crear un cubo en el programa, usar el operador exportar e importar el archivo resultante en Blender. El cubo exportado aparecerá en el viewport de Blender.
- Estado: Completado

Prueba 18

- Nombre: Separar malla
- Descripción: Comprobar que se puede separar una malla.
- Requisitos involucrados: RF17
- Validación: Importar a Suzane con dos subdivisiones como malla triangular, ejecutar el operador Split y comprobar que se obtienen 3 componentes, una para cada ojo y otra más para la cara.
- Estado: Completado

Sprint 9

Prueba 19

- Nombre: Deshacer
- Descripción: Comprobar que se almacenan correctamente los estados del viewport.
- Requisitos involucrados: RF18
- Validación: Crear un cubo mediante un operador. Crear un cono mediante un operador. Ejecutar el operador deshacer. Comprobar que el cono desaparece.
- Estado: Completado

Prueba 20

- Nombre: Exportar escena
- Descripción: Comprobar que el software puede exportar una escena con varios objetos modificados.
- Requisitos involucrados: RF6
- Validación: Importar a Suzane, separar sus componentes y modificar su escala. Añadir un cubo a la escena. Combinar las mallas y exportar. Comprobar que el resultado al importar el archivo exportado en Blender es el mismo que en el programa.
- Estado: Completado

5.0.2. Pruebas finales

El objetivo de estas pruebas es evaluar la funcionalidad del software como conjunto, comprobando que se adecua a los requisitos establecidos. Para ellos, se establecerá una prueba para cada uno de los requisitos no funcionales del proyecto.

Prueba 21

- Nombre: Ejecutable en un PC con Linux
- Descripción: La aplicación deberá poder ejecutarse en un PC con una distribución de Linux instalada.
- Requisitos involucrados: RN1
- Validación: El software se ejecuta en un PC con Manjaro 17 y en uno con ElementaryOS 3.
- Estado: Completado

Prueba 22

- Nombre: OpenGL 3 o posterior
- Descripción: El programa deberá ejecutarse bajo OpenGL 3 para aumentar su compatibilidad y tener un rendimiento adecuado.
- Requisitos involucrados: RN2
- Validación: El software se ejecuta sin errores de renderizado en una GTS650 compatible con OpenGL 3, una GTX 1080 y una Intel HD 620.
- Estado: Completado

Prueba 23

- Nombre: SDK V2 de Leap Motion
- Descripción: La aplicación deberá ser compatible con el SDK de la versión 2 del driver de Leap Motion para aumentar su compatiblidad, ya que no usa funcionalidades de realidad virtual implementadas en el diver Orion.
- Requisitos involucrados: RN3
- Validación: El software se ejecuta correctamente y detecta el sensor Leap Motion en un ordenador con el driver 2.0 instalado, sin estar actualizado a Orion. Se ha comprobado en hasta 3 ordenadores diferentes, todos con la misma versión del driver instalado.
- Estado: Completado

Prueba 24

- Nombre: Resolución mínima de 600 x 1024
- Descripción: Se establece una resolución mínima de 600 x 1024 para aumentar la compatibilidad de la aplicación y su rendimiento.
- Requisitos involucrados: RN4
- Validación: Toda la interfaz y menús del programa se dibujan de forma completa en una ventana de tamaño 600x1024. Se han comprobado otra resoluciones hasta 4K.
- Estado: Completado

Prueba 25
- Nombre: Único usuario
- Descripción: La aplicación deberá ser diseñada para ser usada por un único usuario.
- Requisitos involucrados: RN5
- Validación: La aplicación no contiene ninguna funcionalidad para soportar roles o múltiples usuarios.
- Estado: Completado

Prueba 26
- Nombre: Usable con Leap Motion
- Descripción: La aplicación deberá ser usable mediante un sensor Leap Motion.
- Requisitos involucrados: RN6
- Validación: Se pueden comprobar todos los requisitos funcionales usando solamente el sensor Leap Motion, sin usar ratón o teclado.
- Estado: Completado

Prueba 27
- Nombre: Iconos para las opciones del menú
- Descripción: Se diseñarán iconos para los menús de la aplicación de forma que su usabilidad aumente.
- Requisitos involucrados: RN7
- Validación: Todas las opciones del menú tienen una representación en forma de icono.
- Estado: Completado

Prueba 28
- Nombre: Inglés
- Descripción: La aplicación tendrá su interfaz en inglés.
- Requisitos involucrados: RN8
- Validación: Todas las opciones del menú tienen sus descripciones en inglés.
- Estado: Completado

Prueba 29
- Nombre: Interfaz no dependiente del color
- Descripción: La aplicación no dependerá del color para distinguir funcionalidades o transmitir información al usuario. Esto hará el software más accesible.
- Requisitos involucrados: RN9
- Validación: El programa debe ser usable y no presentar opciones ambiguas tras aplicar un filtro de blanco y negro al monitor en el que se está usando.
- Estado: Completado

Prueba 30

- Nombre: Carga de mallas
- Descripción: La aplicación deberá ser capaz de visualizar mallas de alta densidad.
- Requisitos involucrados: RN10
- Validación: El programa tarda como máximo 20 segundos en cargar una malla en .OBJ de con un millón de vércites y mostrarla en pantalla en un ordenador con un i7 6700, una GTX 1080 y un SSD Samsung EVO Pro. El programa consigue importar el .OBJ en 14 segundos.
- Estado: Completado

Prueba 31

- Nombre: 60 frames por segundo en el viewport
- Descripción: La aplicación deberá tener una tasa de frames estable para facilitar su uso.
- Requisitos involucrados: RN11
- Validación: El programa es capaz de mantener 60 frames por segundo con una malla de 50k vértices en el viewport con una única luz direccional sin sombras en un ordenador con un i7 6700, una GTX 1080 y un SSD Samsung EVO Pro. Se usa el profiler de Godot y se obtiene una media de 220fps en una GTX 1080.
- Estado: Completado

Prueba 32

- Nombre: Separación de mallas
- Descripción: El rendimiento a la hora de separar una malla en sus componentes deberá ser el adecuado.
- Requisitos involucrados: RN12
- Validación: El programa es capaz de partir en sus componentes la malla de Suzane por defecto que incorpora Blender exportada como malla triangular y 2 niveles de subdivisión en un ordenador con un i7 6700, una GTX 1080 y un SSD Samsung EVO Pro en menos de 2 segundos. El programa consigue realizar dicha operación en 0.3 segundos según el profiler de Godot.
- Estado: Completado

Prueba 33

- Nombre: Software libre
- Descripción: Se usará software libre en la medida de lo posible para realizar el proyecto.
- Requisitos involucrados: RN13
- Validación: Todos los componentes usados en el desarrollo del producto son compatibles con la licencia MIT. El único componente no compatible es el SDK de Leap Motion, pero este no se distribuirá con el proyecto.
- Estado: Completado

5.0.3. Estudio de usabilidad

Una vez finalizada la aplicación se han hecho pruebas con diferentes usuarios para valorar su usabilidad. La aplicación se ha probado con 5 personas diferentes, todas ellas sin experiencia previa en el uso del Leap Motion. Observando su comportamiento a la hora de interactuar con la aplicación y los comentarios posteriores, se ha llegado a las siguientes conclusiones:

- Los movimientos de la mano no son completamente naturales, ya que los usuarios han necesitado un periodo de adaptación hasta poder controlar correctamente la aplicación. Esto es debido a la precisión del propio sensor y a sus algoritmos de reconocimiento de imagen.
- Los usuarios están conformes con la calidad de los gráficos y diseño del programa.
- Algunas acciones se necesitan algún tipo de atajo. Algunos usuarios notan que acciones típicas de un entorno 3D, como girar un objeto 180 grados, son muy lentas con este sistema.

También se ha llevado a cabo un estudio formal sobre la usabilidad del software, basado en los principios heurísticos de Nielsen:

- Visibilidad de estado del sistema: el sistema cuenta con una barra de estado para comprobar el operador que se está ejecutando en cada momento, así como el estado del sensor y las manos.
- Utilizar el lenguaje de los usuarios: el programa usa términos comunes a muchos otros softwares de edición 3D, por lo que cualquier persona familiarizada con los mismo no debería tener problema al usarlo.
- Control y libertad para el usuario: Todas las acciones del sistema pueden cancelarse en cualquier momento abriendo la mano. Además, el sistema cuenta con un operador para deshacer el último cambio realizado.
- Consistencia y estándares: Los iconos de la aplicación son consistentes con lo que representan y solo tienen un uso dentro de la aplicación. El lenguaje de los menús es adecuado para una aplicación de este tipo.
- Prevención de errores: El sistema no puede entrar en un estado de error. De todas formas, todos los cambios realizados pueden deshacerse y el viewport puede devolverse a su posición inicial en cualquier momento.
- Memorizar la carga de memoria: Los objetos cambia de material según su estado para evitar comprobar las listas de objetos seleccionados.
- Flexibilidad y eficiencia de uso: Los movimientos requeridos para activar las opciones del menú son siempre los mismos por lo que un usuario avanzado puede acceder a dichas opciones de forma rápida sin tener que leer todos el menú.

- Estética y diseño minimalista: La aplicación muestra los objetos de la escena de la forma más limpia posible, sin ningún tipo de elemento que distraiga o impida su correcta visualización.
- Ayudar a reconocer y recuperarse de errores: Se informa al usuario cuando el sensor no está conectado o no reconoce bien las manos. Además, ningún cambio en el programa es permanente ya que se puede deshacer.
- Ayuda y documentación: El software incluye documentación. Los operadores son suficientemente intuitivos y responden al movimiento para que el usuario pueda entenderlos nada más ejecutarlos.

Capítulo 6

Conclusiones y posibles ampliaciones

En este capítulo se recogen las conclusiones del desarrollo del proyecto y se enumera una serie de ampliaciones que se le podrían añadir al software en un futuro.

Contents

6.1. Conclusiones

En este apartado se recogen las conclusiones del desarrollo del proyecto.

El resultado del proyecto es un producto software que permite la interacción y edición con modelos en 3D mediante el uso del Leap Motion, así como su memoria y documentación asociada.

Para ello, se ha realizado un análisis de requisitos y una gestión del proyecto. También se ha llevado a cabo un análisis para determinar la tecnología más adecuada para desarrollarlo, en este caso Godot Engine 3.

Posteriormente se han analizado diversas aplicaciones de edición 3D, extrayendo conclusiones sobre sus similitudes y diferencias para poder afrontar el diseño de la interfaz de este software con la mayor información posible. También se han diseñados los paradigmas adaptados al uso del Leap Motion para facilitar la interacción con el mismo.

El proyecto se ha implementado en Godot Engine 3, siendo necesaria la inclusión de una biblioteca de GDNative para permitir el uso del Leap Motion dentro del motor. Una vez resuelto el problema de compatibilidad mediante el uso de dicha biblioteca se ha desarrollado la aplicación en GDScript, adaptado su estructura al funcionamiento de Godot. Esto permitirá que la aplicación sea fácilmente ampliable en el futuro.

Para llevarlo a cabo se ha usado la metodología scrum, en la que se han ido cumpliendo los requisitos del proyecto de forma progresiva en diferentes sprints hasta llegar al producto final. En la tabla 6.1 se muestran como las pruebas diseñadas durante el proyecto cubren los requisitos del mismo:

Cuadro 6.1: Matriz de trazabilidad: Requisitos funcionales y pruebas

	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15	P16	P17	P18	P19	P20
RF1			X		X															
RF2			X	X																
RF3						X	X													
RF4						X	X													
RF5										X						X				
RF6										X										
RF7										X							X			
RF8																		X		
RF9						X														
RF10							X				X				X					
RF11										X										
RF12										X		X	X							
RF13										X										
RF14										X			X							
RF15										X				X						
RF16										X										
RF17										X									X	
RF18										X										X

6.2. Ampliaciones

En este apartado se citan una serie de ampliaciones que se le podrán añadir al software.

El software actualmente permite la visualización de modelos y la edición sencilla de los mismos, así como añadir objetos rápidamente y modificarlos para probar composiciones o volúmenes. Su arquitectura hace que se puedan añadir nuevas funcionalidades muy fácilmente con solo programar más operadores. Además, se puede mejorar su rendimiento pasando partes del código de la aplicación de GDNative a C++.

Entre otras, al programa final se le podrían añadir las siguientes características:

- Operador de escalado global para permitir escalar todos los elementos a la escena a la vez sin necesidad de combinar sus mallas.
- Operadores booleanos para permitir obtener la unión o intersección de dos mallas
- Permitir ocultar o mostrar objetos
- Permitir acceder a una biblioteca de mallas primitivas para la creación de modelos en lugar de los operadores de añadir malla disponibles actualmente

Como conclusión, es un software básico pero que permite visualizar e interactuar con los modelos mediante el Leap Motion de forma correcta y satisfactoria, pero sin funcionalidades de edición avanzadas que necesitan una herramienta de control de precisión para poder realizarse con éxito.

Apéndice A

Manuales técnicos

En este apéndice se dará una explicación sobre como configurar el entorno de desarrollo y modificar la estructura del programa para poder continuar con el desarrollo del mismo en un futuro o transferirlo a una tercera persona.

A.1. Estructura de archivos

El proyecto contiene las siguientes carpetas

- **cpp_bindings**: Contiene los bindings de C++ que se usan para cargar la librería de GDNative.
- **godot_header**s: Contiene los archivos .h de la API de Godot para poder programar bibliotecas en C++.
- **leap_header**s: Aquí deberán colocarse los archivos .h del SDK de Leap Motion. No se distribuyen con el proyecto ya que no son compatibles con la licencia del mismo.
- **lib**: Contiene bibliotecas compiladas. Entre ellas la biblioteca compilada de los bindigs de C++ y la biblioteca que implementa la interfaz de GDNative para el programa. En caso de querer actualizar esta biblioteca deberán reemplazarse estos archivos
- **materials**: Contiene archivos de recursos de los materiales que se usarán en el programa
- **mesh**: Contiene OBJ con las mallas que utiliza el programa para instanciar sus primitivas.
- **models**: Carpeta de entrada/salida del programa. Los modelos que se desean cargar se colocarán en esta carpeta. También es la carpeta a la que el programa exportará por defecto los modelos.
- **scenes**: Contiene los archivos .tscn de las escena de Godot

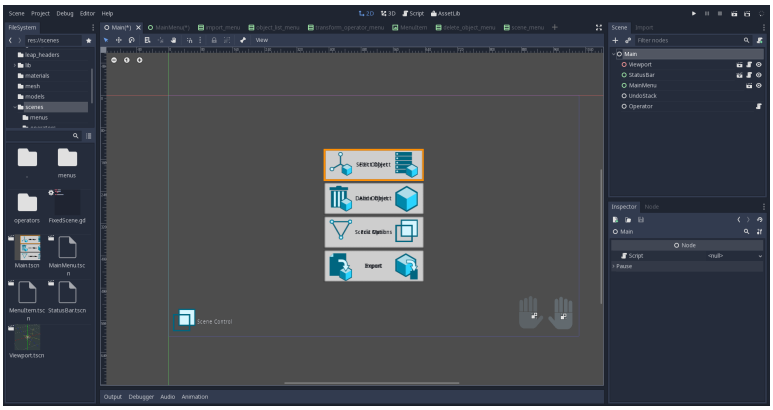

Figura A.1: Entorno de desarrollo en Godot

- **scripts**: Contiene los scripts del proyecto
- **sprites**: Contiene todos los recursos gráficos del programa, como texturas de los botones e iconos.
- **src**: Contiene código C++ de la biblioteca de GDNative.

Otros archivos importantes son:

- **icon.png**: Icono del programa por defecto. En caso de exportar el programa en una versión para Windows este icono no se usa, habría que cambiarlo manualmente modificando el ejecutable con alguna otra herramienta.
- **leapmotion.gdns y leapmotion.gdnlib**: archivos para describir la biblioteca de GDNative. Si se desean añadir nuevas compilaciones de la misma para otros sistemas operativos deberán modificarse estos archivos.
- **project.godot**: Archivo de proyecto de Godot.

Para comenzar a modificar la aplicación simplemente es necesario abrir el archivo project.godot desde una versión de Godot Engine 3.0 o posterior. Para ello, se importa este archivo en el gestor de proyectos de Godot y se selecciona la opción editar. También es necesario tener instalado el SDK .0 de Leap Motion y su driver.

El hilo que gestiona el Leap Motion deberá estar ejecutándose en el sistema. En algunas versiones de Linux esto no ocurre automáticamente y es necesario iniciarlo manualmente mediante el comando leapd.

Una vez abierto el proyecto se cargará la escena Main, la cual es el punto de inicio del programa. Es recomendable seguir las siguientes indicaciones en el caso de querer añadir elementos al proyecto:

- Todos los elementos relacionados con la vista 3D se añadirán a la escena Viewport. Esta escena es la única que debería contener nodos que hereden de Spatial.

- Si se desean añadir elementos a la interfaz que se superpongan sobre el viewport estos deberán añadirse a la escena StatusBar.
- Los nodos Operator y UndoStack no debería ser necesario modificarlos

A.2. Operadores

Si se desea añadir más funcionalidades al programa es necesario añadir nuevos operadores. Para ello es necesario seguir los siguientes pasos:

1. Crear un escena para el nuevo operador en la carpeta scenes/operators que herede de la escena operator situada en la misma carpeta
2. Añadir al nodo principal de esta escena un nuevo script que herede del script scripts/operators/operator.gd
3. Añadir los nodos correspondientes para implementar la funcionalidad deseada y un nodo Icon en caso de querer una representación del operador en la barra de estado.

Para lanzar el nuevo operador es necesario crear nuevas entradas en el menú. Para ello puede añadirse a un menú existente o crear un nuevo menú en el menú principal.

Para añadir el operador a un menú existente:

1. Buscar el menú correspondiente en la carpeta scene/menus
2. Instanciar bajo el nodo MainMenu la escena MenuItem
3. Seleccionar la opción editar hijos en el editor. Colocar a la escena el nombre e icono correspondiente.
4. Seleccionar la escena del operador que se va a ejecutar en en la variable exportada Operator que se muestra en el inspector

En caso de necesitar un nuevo menú:

1. Instanciar una nueva escena MenuItem bajo el nodo MainMenu de la escena MainMenu
2. En el inspector, seleccionar los contenedores en los que se va a instanciar el nuevo menú. En este caso, se seleccionarán los nodos LeftMenuContainer y RightMenuContainer
3. Crear una nueva escena Para el Menú que herede de scene/menu y seleccionar la opción de editar hijos
4. Añadir a esta escena las opciones correspondientes en forma de escenas MenuItem como se detalla en el apartado anterior.
5. Referenciar a esta escena en el primer MenuItem creado, dependiendo de la dirección en la que se pretenda abrir el menú

A.3. Interacción

Si se desea calibrar o modificar la forma de interacción del programa deberá hacerse en los siguientes scripts:

- Para modificar cualquier interacción de la mano derecha sobre el viewport es necesario modificar el método _process() del script viewport.gd. Dicho script contiene las constantes para ajustar la sensibilidad y calibrar el movimiento del viewport.
- Para modificar la interacción de la mano izquierda sobre los menús del programa es necesario modificar el script LeapAction.gd. Este script contiene las constantes que delimitan la rejilla virtual sobre la cual se interpretarán los movimientos de la mano a la hora de traducirse a señales.

Apéndice B

Manual de usuario

En este apéndice se explicarán los pasos a seguir para utilizar las funcionalidades del programa. En primer lugar se describirán las partes de la interfaz y su objeto. A continuación se detallarán los pasos necesarios para llevar a cabo las acciones más comunes del programa. Por último, se listarán posibles soluciones a problemas que el usuario pueda tener

B.1. Interfaz

La interfaz cuenta con las siguientes partes:

- Viewport: Espacio dedicado a mostrar los modelos en 3D. Si la ventana se maximiza este espacio ocupa todo lo que sea posible
- Menu: Se abre cerrando la mano izquierda sobre el sensor. Contiene todas las opciones y funcionalidad del programa
- Barra de estado: Su lado izquierda muestra información sobre el operador que se está ejecutando, por defecto SceneControl. En la parte derecha se muestra el estado de las manos. Aquí se mostrará un mensaje de error en caso de que el Leap Motion no esté conectad y las manos no aparecerán.

B.2. Acciones

Desplazar la cámara del Viewport:

1. Colocar la mano derecha abierta sobre el sensor
2. Cerrar levemente la mano hasta que el indicador de la mano derecha cambie
3. Mover la mano libremente
4. Abrir la mano derecha para confirmar el movimiento

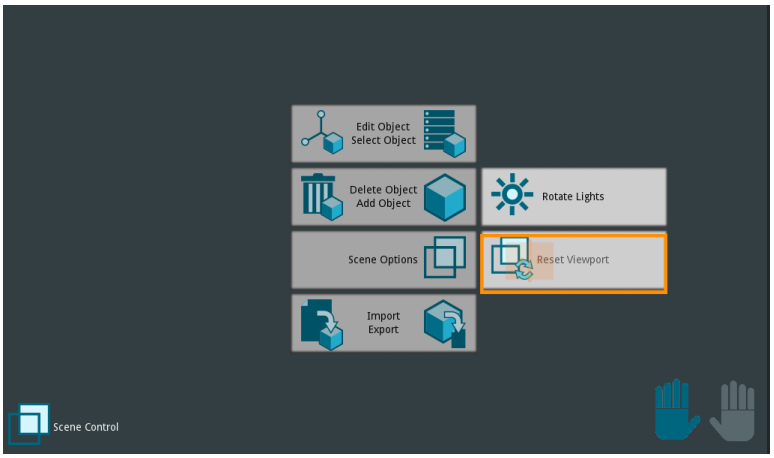

Figura B.1: Interfaz del programa

Moverse por los menús:

1. Colocar la mano izquierda abierta sobre el sensor
2. Cerrar levemente la mano izquierda hasta que el menú principal aparezca
3. Desplazar la mano izquierda cerrada hacia arriba o hacia abajo para moverse por las opciones del menú
4. Desplazar la mano izquierda hacia la derecha o hacia la izquierda para seleccionar la opción correspondiente dentro del menú
5. Abrir la mano izquierda para cancelar la operación.

Importar Modelo:

1. Colocar el modelo en formato obj en la carpeta Models del programa
2. Dentro del programa, seleccionar el modelo correspondiente en el desplegable que aparece al seleccionar la opción Import del menú principal

Exportar Modelo:

1. Dentro del programa, seleccionar la opción Export del menú principal. El modelo se exportará a la carpeta Models.

Seleccionar y deseleccionar objeto

1. En el menú principal, seleccionar el objeto deseado bajo la opción Select Object
2. Para anular la selección, usar la opción Select All dentro del mismo menú

Escalar Objeto

1. Seleccionar la opción Transform/Scale Local del menú principal
2. Sin abrir la mano izquierda, cerrar la mano derecha
3. Juntar o separar ambas manos
4. Abrir la mano izquierda para confirmar la operación

Añadir Objeto

1. Seleccionar la opción Add Object del menú
2. En el menú que de despliega seleccionar el objeto deseado
3. Sin abrir la mano izquierda, posicionar el objeto en la escena
4. Abrir la mano izquierda una vez el objeto se encuentre en la posición deseada

Eliminar Objeto

1. Seleccionar la opción Add Object del menú
2. En el menú que de despliega seleccionar el objeto deseado
3. Sin abrir la mano izquierda, posicionar el objeto en la escena
4. Abrir la mano izquierda una vez el objeto se encuentre en la posición deseada

Separar Objetos

1. Seleccionar la opción Edit Mesh/Split del menú
2. Los objetos seleccionados se separarán

Unir Objetos

1. Seleccionar la opción Edit Mesh/Merge del menú
2. Todos los objetos de la escena se juntarán en un único objeto

Restaurar vista

1. Seleccionar la opción Scene Options/Reset viewport del menú
2. La escena volverá al centro de la pantalla

B.3. Solución a problemas comunes

Aparece un error en pantalla y no aparecen los iconos de las manos en la barra de estado

- Asegurarse de que el sensor esté conectado al equipo
- Comprobar que la versión instalada del diver de Leap Motion es la versión 2.0
- Comprobar que el hilo del sensor Leap Motion se está ejecutando en el equipo. En caso contrario ejecutar el comando leapd

El movimiento es poco preciso

- Limpiar la parte superior del sensor
- Apagar todas las fuentes de luz que se encuentren directamente encima del sensor

La malla no se importa correctamente

- Probar a exportar el modelo en una escala diferente
- Comprobar que la malla es una malla triangular, no puede contener otro tipo de polígonos para que el programa la interprete correctamente.

Apéndice C

Licencia

Bibliografía

La bibliografía citada a continuación es de carácter general.

[1] Leap Motion SDK (*https://developer.leapmotion.com/documentation*).

[2] Unity C# API (*https://docs.unity3d.com/ScriptReference*).

[3] Blender developer portal (*https://wiki.blender.org/index.php/Main_Page*).

[4] Godot 3.0 documentation (*https://docs.godotengine.org/en/latest*).

[5] Mischa Spiegelmock, *Leap Motion Development Essentials*, Packt Publishing, 2013.

[6] HTC Vive (*https://www.vive.com/us/*).

[7] TiltBrush (*https://www.tiltbrush.com/*).

[8] Leap Motion (*https://www.leapmotion.com/*) visitada el 3 de febrero de 2018.

[9] Project Orion (*https://developer.leapmotion.com/orion/*) visitada el 2 de febrero de 2018.

[10] Leap Motion Rig in Three.js (*http://blog.leapmotion.com/manipulating-rigged-hand-with-leap-motion-in-three-js/*) visitada el 2 de febrero de 2018.

[11] How to build a Leap Motion education app (*http://blog.leapmotion.com/build-world-class-education-app/*) visitada el 2 de febrero de 2018.

[12] 5 Medical and Assistive Technologies Being Transformed with Leap Motion(*http://blog.leapmotion.com/5-medical-and-assistive-technologies-being-transformed-with-leap-motion-tech/*) visitada el 2 de febrero de 2018.

[13] Changing How People Look at Physical Therapy(*http://blog.leapmotion.com/changing-people-look-physical-therapy/*) visitada el 2 de febrero de 2018.