Universidad de Alcalá
Escuela Politécnica Superior

GRADO EN INGENIERÍA TELEMÁTICA

Trabajo Fin de Grado

Estudio y desarrollo de una aplicación de retrato robot en Blender.

Autor: Jose Maria Oliet Villalba

Tutor/es: Bernardo Alarcos Alcázar

2021

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

GRADO EN INGENIERIA TELEMÁTICA

Trabajo Fin de Grado

Estudio y desarrollo de una aplicación de retrato robot en Blender.

Autor: Jose Maria Oliet Villalba

Tutor/es: Bernardo Alarcos Alcázar

TRIBUNAL:

Presidente: Luis de la Cruz Piris

Vocal 1º: Antonio García Herráiz

Vocal 2º: Bernardo Alarcos Alcázar

FECHA: 22/09/2021

INDICE:

Contenido

INDICE:

Resumen.

Este trabajo consiste en el estudio y desarrollo de una herramienta de edición de rostros tridimensionales, basada en el entorno de modelado 3D Blender y programada en Python, con objetivo de servir de apoyo a las tareas policiales de elaboración de rostros o retratos robot para posibles identificaciones. Esta herramienta supone el primer paso de un proyecto mayor en el que se busca trasladar la funcionalidad de las aplicaciones de retrato robot tradicionales a un nuevo entorno tridimensional.

Summary.

This proyect consists in the study and development of a 3D modelling tool focused on face editing, programmed in Python in the 3D open source suite Blender, with the objective of supporting identikit labours in the field of criminalistics. The ultimate goal of this proyect is starting a long-term development of a tool that translates the technics and knowledge of tradicional identikit solutions to a new, unexplored 3D environment

Resumen extendido del trabajo.

Este trabajo consiste en la búsqueda de una solución para labores de identificación y reconstrucción facial, también conocidas como retratos robots, para uso policial. Este trabajo trata de complementar los conocimientos de un profesional del campo del reconocimiento facial para desarrollar un entorno de edición 3D que permita alterar las distintas características morfo-faciales de un "busto plano" para poder recrear el rostro tridimensional deseado.

Al tratarse del primer paso de un proyecto mayor, se ha buscado ofrecer una herramienta que sirva como punto de partida para proyectos derivados posteriores, para lo que se han tenido en cuenta diversos objetivos como portabilidad, compatibilidad, sencillez y eficiencia de código y funcionamiento, para conseguir una solución transparente y que pueda actuar de "caja negra" dentro de un proyecto mayor. Así mismo, parte del código desarrollado puede extrapolarse a nuevas herramientas.

En primer lugar, se ha hecho un estudio en colaboración con el personal de la policía para establecer el enfoque que queremos darle al trabajo, ya que el usuario final será personal especializado y tanto el programa como la interacción con el mismo deben corresponderse con las expectativas especificadas.

Con el esbozo de lo que sería el trabajo, se ha realizado un estudio de los recursos software existentes sobre los que fundamentar el proyecto, entre los que se encuentran Maya, Blender, Poser y Daz Studio. Finalmente se ha optado por el entorno de modelado Blender: es un entorno de edición 3D con muchas herramientas, es software de código abierto y cuenta con gran cantidad de recursos online, en forma de herramientas y de tutoriales y tiene habilitada una API basada en Python, que permite el desarrollo y *testing* de los scripts en los que se tenía pensado basar el trabajo. Por último, no hay coste de licencia y permite un desarrollo sin ataduras económicas relacionadas.

Adicionalmente, en las primeras etapas de familiarización con el entorno, se estuvo trabajando con la herramienta FaceBuilder, elaborada por KeenTools: esta herramienta permite la elaboración de modelos craneofaciales tridimensionales a partir de una o más fotos de un rostro, mediante una sencilla sobreposición de una máscara a las fotos tomadas, con las que se elabora más tarde la piel de ese modelo. Los resultados eran muy llamativos, pero se optó por descartarla para no estar atados a una licencia de pago continuo. Sin embargo, se aprendió mucho del entorno Blender al trabajar con ella.

Una vez escogido el entorno, se ha investigado las formas de trabajar con el mismo, tanto en la edición de la *mesh* como en la interacción entre la interfaz gráfica y el programa: una mesh es un tipo de objeto de Blender constituido por una colección de vértices, bordes y caras; alterar la mesh significa alterar la geometría del objeto y, por lo tanto, su forma. Como resultado de estas investigaciones se han elaborado distintas versiones y distintas herramientas que se han refinado más tarde, siendo estas una interfaz gráfica básica y ciertos scripts de interacción con los vértices del objeto, que ha sido el primer paso en lo que sería más tarde la base del programa. El funcionamiento de esta interacción se detallará más tarde, pero lo más relevante es que los vértices son accesibles mediante punteros, gracias a los cuales se pueden elaborar scripts que pueden registrar distintas variables relacionadas, como valor del puntero y coordenadas del vértice.

Una vez concretadas las posibilidades del entorno y elaborados scripts básicos de interacción, se realizó una reunión con los colaboradores del proyecto para acotar el alcance del proyecto

en función de los requisitos en base a los que se había conceptualizado y también en las posibilidades realistas en base al tiempo. Las conclusiones se extrajeron de esta reunión fueron:

- Es necesario adaptar el trabajo previamente realizado a un entorno cambiante y abstraer sus funcionalidades a un usuario que probablemente no contaría con la posibilidad de interactuar cómodamente con ellas en el estado actual.
- La funcionalidad deseada es similar a un editor de personajes, es una herramienta presente en la industria del videojuego que permite ajustar la apariencia física del avatar del jugador en tiempo real, pudiendo hacer cambios a partes concretas de su cuerpo o cara.

El trabajo final se ha concebido como un creador de editores de personajes, o bien como una herramienta que permite crear perfiles de edición de un objeto, pudiendo este importarse y exportarse a cualquier objeto similar. Dada la necesidad de crear un editor que se adapte a los conocimientos anatómicos de un profesional, se ha optado por crear un entorno que facilite las herramientas para que cada persona pueda configurar su propio editor de manera sencilla, sin límite en cuanto a complejidad. El editor creado es sencillo de navegar e imita en interfaz y funciones a un editor de personaje al uso.

Después de establecer los objetivos, se ha estado esbozando la arquitectura del programa: esto ha supuesto un largo tiempo dedicado a ello dado que, aunque parte del código alude a elementos pertenecientes al contexto de Blender, una parte muy importante del funcionamiento del programa requiere de conceptos que no se pueden reflejar directamente en Blender; las metas principales que se han logrado en este trabajo han sido:

- Grupos de vértices: un grupo de vértices es un conjunto de distintos vértices identificados bajo un nombre propio y único. Se comportan como una unidad si son seleccionados y responden como tal si son alterados. Responde a la necesidad de una mejor interacción con los vértices, la manera primitiva era poco eficiente y difícil de compatibilizar con las herramientas más complejas que se iban a emplear. Un grupo de vértices es una solución óptima dentro del contexto de Blender. En esta etapa se han creado scripts que permiten la creación de grupos de vértices e interacción mediante ratón con ellos.
- Zonas faciales o zonas: las zonas faciales o zonas se han creado para poder conservar la información contenida en los grupos de vértices fuera de Blender, ya que los grupos de vértices no son exportables y es necesario que la información contenida en ellos sea fácilmente accesible y almacenable. Sobre estas se cimienta el concepto de macrozonas y transformaciones. Para ello se han creado scripts que permiten crear e interactuar con zonas faciales con nombres personalizados.
- Interacción con ficheros: para hacer persistente la información y las opciones creadas en un editor dado, es necesario que dicha información se acceda de manera externa al editor de Blender. Las herramientas elaboradas sirven además para numerosas aplicaciones posteriores de gran importancia. Los scripts comprendidos aquí son los de guardar un perfil y cargarlo. Estos scripts se han ido refinando constantemente a lo largo del desarrollo.
- Macrozonas: esta etapa comprende el desarrollo de las macrozonas, así como de las herramientas necesarias para darles soporte, ya que estas no son representables en el contexto de Blender. Para ello se han adaptado los scripts de guardado de perfil de edición para darle soporte, se ha implementado una jerarquía de ficheros para separar los distintos perfiles, se ha incorporado una variable global para dar

persistencia a un perfil dentro del contexto de Blender y por último se han elaborado scripts de interacción con macrozonas.

- Transformaciones: no se han podido concretar en código, pero sí que se ha podido crear la infraestructura que les dará soporte, como la relación que tendrán con el resto de herramientas.

Habiéndose completado los objetivos concretados, he extraído las distintas conclusiones acerca de este proyecto: ha sido un proyecto muy provechoso, en el que he mejorado aspectos profesionales como la búsqueda de recursos y su adaptación a mis objetivos, así como la optimización de las distintas partes para un funcionamiento eficiente. Sin embargo, la experiencia más valiosa ha sido sin duda la conceptualización y elaboración de la arquitectura de un programa desde cero, desde el desarrollo de las distintas herramientas individuales hasta su interoperabilidad y establecimiento del papel de cada una de ellas en el conjunto del programa.

Palabras clave.

Blender, Python, Identikit, 3D Modeling.

Glosario.

Vértice: junto a los bordes y caras, son elementos de la mesh de un objeto de Blender. Está caracterizado por un valor de índice, un valor numérico que lo identifica de manera única, y por las coordenadas tridimensionales del mismo.

Objeto: dentro del contexto de Blender, es una entidad unitaria seleccionable. **que comprende un objeto como un conjunto de su geometría y textura.** [coteja]

Perfil de edición: es el conjunto de opciones e información de los elementos creados dentro de un escenario concreto del programa. Estos perfiles son almacenados externamente y pueden importarse y exportarse a cualquier objeto similar, manteniéndose su funcionalidad. Dentro de Blender, el perfil actual se almacena en una variable global.

Objeto similar: dentro del contexto de este proyecto, se dice que es un objeto similar aquel que tiene el mismo número de índices, indexados exactamente igual. Solo así un perfil de edición creado puede mantener la coherencia entre dos objetos.

Zona facial: consiste en un fichero *txt* en el que están almacenados el nombre del grupo de vértices asociado (comparten nombre) y los índices de los distintos vértices que lo componen. Juegan un papel clave para la persistencia de un entorno de edición.

Macrozona: es un término acuñado personalmente, es un conjunto de zonas faciales y de transformaciones, almacenados externamente en un documento .txt. Este concepto permite reunir distintas zonas faciales bajo un grupo que las caracteriza, pero permite acceder a cada una de ellas por separado, dando soporte a las transformaciones.

Transformación: es la alteración de la geometría de una o más zonas faciales, identificada bajo un nombre único y accesible desde una macrozona.

Rigging: es el proceso de crear un sistema de controles digitales y agregárselos a un modelo 3D para que así pueda ser animado fácil y eficientemente.

Capítulo 1 Introducción.

Las aplicaciones de retrato robot empleadas por los cuerpos policiales juegan un papel muy importante en las acciones de reconocimiento facial. Actualmente hay numerosas aplicaciones desarrolladas y con un largo historial de uso como FACES que, sin embargo, están enfocadas en un entorno estrictamente bidimensional.

El objetivo del proyecto es ampliar la utilidad de estas aplicaciones a un nuevo entorno tridimensional, aplicando la tecnología ya existente en el campo de modelado 3D.

El modelado 3D es un área de investigación en el que se están desarrollando muchos estudios, incluidos varios en el ámbito del reconocimiento facial [1], por lo que existen numerosos antecedentes, referencias y software de código abierto con los que trabajar. Tras una investigación, se ha considerado el entorno de modelado 3D Blender como la opción idónea, ya que es de código abierto y cuenta con mucho apoyo en forma de software de terceros.

El mayor nivel de detalle de un modelo 3D mejoraría sustancialmente las posibilidades de reconocimiento por parte de testigos o de personas que podrían ayudar a localizar a un delincuente, a una víctima o a una persona desaparecida. Además, se podría hacer uso del resto de herramientas de Blender para dar lugar a nuevas aplicaciones, como la simulación de la iluminación de la escena en la que se produjo el avistamiento o la animación de una cara para simular gestos o el pronunciamiento de frases.

El planteamiento del trabajo ha pasado por numerosas fases de conceptualización y de replanificación, debido a que las sucesivas fases de desarrollo marcaban límites y nuevas posibilidades, por lo que ha sido necesario ir cambiando parte de los objetivos, así como el desarrollo que conduce a ellos.

El objetivo principal del proyecto es desarrollar una aplicación que sirva como como punto de apoyo para el desarrollo de herramientas futuras, para ello se han desarrollado los siguientes objetivos:

- Construir una herramienta que adapte las funcionalidades y apariencia de un editor de personajes al entorno elegido, concretamente las referentes a alteraciones físicas de los distintos elementos faciales.
- Diseñar una herramienta que se adapte a distintos objetos sin problemas de compatibilidad.
- Hacer que esta herramienta se comporte de manera transparente, esto es sin realizar cambios en la composición del objeto y siendo por lo tanto compatible con herramientas de terceros.
- Abstraer la funcionalidad para hacer de su uso algo sencillo para un usuario de Blender básico.
- Hacer de los perfiles de edición creados herramientas exportables y compatibles con objetos similares.

Como consecuencia del desarrollo de estos objetivos, se han adquirido también competencias en el entendimiento y manejo de Blender, así como en el establecimiento de los objetivos en base a los requisitos de tiempo y de recursos y en su consecuente ejecución.

Por último, se define la estructura de la memoria, este capítulo es el capítulo 1 y sirve de introducción. En el capítulo 2 se explicarán las bases teóricas del trabajo, entre las que se encuentra un estudio de Blender, la arquitectura general del programa y la explicación de

conceptos propios del trabajo, como zonas, macrozonas y transformaciones. El capítulo 3 ahondará en las partes del programa cuyo código es especialmente relevante de cara a respaldar los conceptos teóricos previamente explicados. El capítulo 4 está constituido por un manual de usuario, pensado para ofrecer una guía para los tres perfiles de usuario para los que está pensado este programa, como se explicará más adelante. El capítulo 5 trata las conclusiones extraídas del desarrollo del proyecto y el capítulo 6 contiene las referencias y la bibliografía.

Capítulo 2 Bases teóricas del proyecto.

En este capítulo se explicarán los objetivos de la aplicación, las características del entorno elegido para desarrollarla, los elementos básicos en los que se fundamenta y los conceptos teóricos de las herramientas desarrolladas.

Objetivo de la aplicación y competencias deseadas.

Este proyecto está enfocado a crear un entorno de modelado 3D para modelos craneofaciales, con el objetivo final de servir en las tareas de retrato robot llevadas a cabo por los servicios policiales. La aplicación desarrollada debe contar con herramientas para dividir un modelo facial en distintas partes como nariz, boca u ojos y poder hacer transformaciones sobre ellos. Debe posibilitar a un experto desarrollar a un editor lo suficientemente complejo como para llevar a cabo las labores de retrato robot, y además debe contar con una interfaz gráfica que permita tanto al diseñador como a un usuario novel del programa hacer uso de dicho editor previamente creado.

Este entorno de edición está inspirado en los "editor (también dicho creador) de personaje" empleados ampliamente en el sector del videojuego: estos programas permiten crear al personaje jugable a gusto del jugador, alterando un amplio espectro de rasgos físicos, entre los que se encuentran los rasgos faciales. En la **figura 1** podemos ver el editor de personajes de uno de estos juegos.

Figura 1) Apartado facial del creador del personaje del videojuego *Monster Hunter: World*.

La interfaz de usuario es fácil de comprender y es intuitiva, por lo que sirve de referencia para diseñar la de esta aplicación.

Otro aspecto muy importante del proyecto es que la aplicación servirá de base a un proyecto mayor, con herramientas más avanzadas desarrolladas posteriormente, por lo que es muy importante que disponga de la mayor compatibilidad posible con las nuevas herramientas y con otras aplicaciones.

Blender se presentó como la opción ideal para llevar a cabo los objetivos deseados, ya que además de ser un software gratuito y de código abierto, cuenta con numerosas herramientas

que permiten no solo diseñar el código de la interfaz de usuario y del modelado en 3D, si no que además dan la posibilidad de ampliar las posibilidades del programa a nuevas aplicaciones muy útiles como la animación del modelo y la iluminación. Además, dispone de software creados por terceros que podrían elevar las posibilidades de la aplicación a un nuevo nivel.

La aplicación final permite crear distintos "editores de personaje", a gusto del usuario, y almacenarlos de manera que sean exportables a cualquier equipo que cuente con Blender instalado. Cada editor es independiente del resto y puede ser tan complejo como lo permita el nivel de detalle de la geometría del modelo 3D en el que se ha creado.

Facebuilder y software de terceros.

En la investigación previa al establecimiento de las herramientas de trabajo se jugó con la aplicación de terceros FaceBuilder, una aplicación creada por el estudio Keentools que, mediante el uso de fotografías, realiza una construcción tridimensional del modelo del cráneo y de la cara de la persona fotografiada, al que se le aplica una "piel" realizada a partir de esas fotografías.

Los resultados que ofrece esa aplicación son muy llamativos, y se pudo recabar mucha información del funcionamiento de la geometría de un objeto y de cómo realizar transformaciones en la misma. Sin embargo, el uso de esa herramienta se acabó descartando dado que nos interesa tener total independencia de software de terceros en el programa final.

Se ha utilizado el modelo de cabeza base de FaceBuilder para realizar las pruebas de funcionalidad de las herramientas, dado que es un archivo *blend* (extensión por defecto de archivos de Blender) y es un objeto con un elevado número de vértices y una geometría bastante detallada, ideal para comprobar la funcionalidad de las herramientas que se han ido desarrollando, aunque es perfectamente sustituible. Sin embargo, no se ha empleado nada del contenido, propiedad de KeenTools, para la realización de esas herramientas, ni se hace referencia a elementos de FaceBuilder en ningún lugar del código, dado que nuestro proyecto es independiente de él.

Para terminar de aclarar la relación del proyecto con el software de terceros: el proyecto se apoya única y solamente en el entorno Blender y en Python como lenguaje de programación, empleando librerías propias, sin utilizar librerías que pertenezcan a una tercera parte.

El entorno Blender.

Blender es una suite de creación 3D de código abierto, en la que se recogen distintas utilidades relacionadas con el modelado, la edición, el rigging, la animación, la simulación, el renderizado, la composición y la captura de movimiento de objetos 3D. Además, cuenta con una API de Python, que hace muy fácil la integración de scripts.

Al abrir Blender se despliega el escenario tridimensional que muestra la imagen, en cuyo centro se encuentra un objeto, por defecto un cubo. Es en este escenario donde tienen lugar las utilidades descritas anteriormente, y en él se puede añadir cualquier objeto con el que se quiera trabajar. En la **figura 2** puede verse dicho escenario.

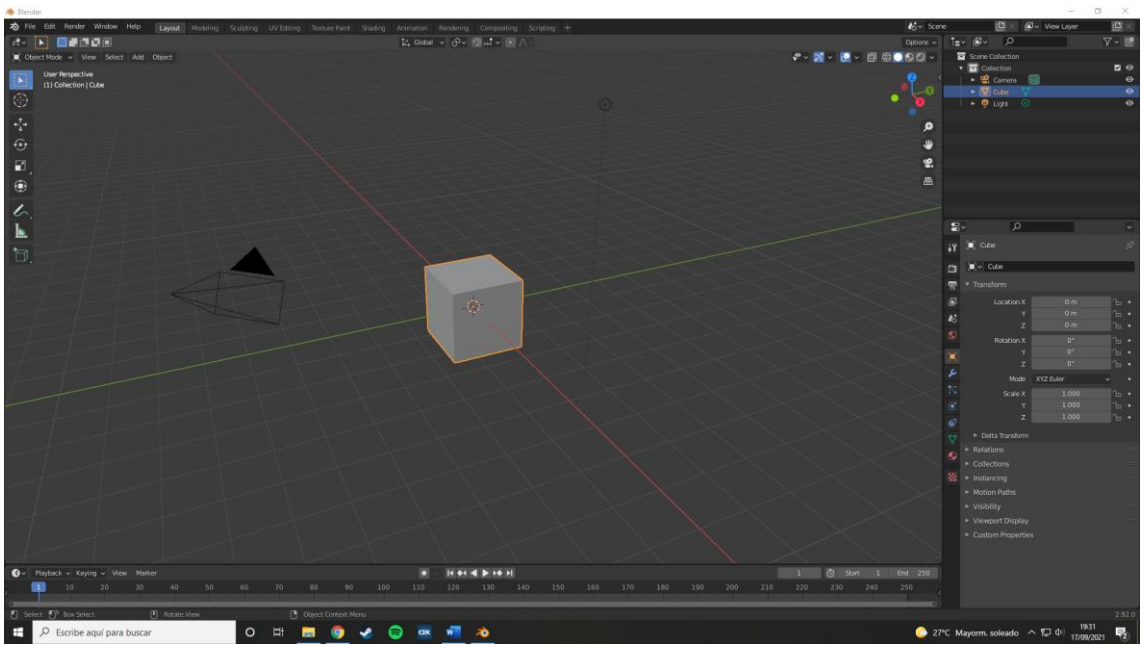

Figura 2: escenario general recién desplegado

Blender tiene distintos modos de visualizar un objeto, y cada uno muestra información diferente. Aquí se muestran los modos que se han empleado en la duración de este proyecto, su modo de uso se detallará en el manual de usuario:

- Modo “layout” o objeto: es el modo que se muestra por defecto al abrir un nuevo archivo *blend*, muestra el objeto en sí y su ubicación en el escenario.
- Modo “modeling” o edición: este modo muestra la geometría del objeto, compuesta de vértices, bordes y caras. Aquí se pueden hacer modificaciones a esa geometría.
- Modo scripting: despliega varias interfaces dirigidas a la interacción por comandos con el usuario, desde un editor de texto hasta ventanas de consola.

En este trabajo nos centramos la *mesh*, que es el conjunto de vértices, bordes y caras que definen la forma de un objeto [2]. La **figura 3** muestra el modelo de Blender *Suzanne*, visto mediante el modo edición, que permite ver la mesh o geometría del objeto y todos los elementos que la constituyen.

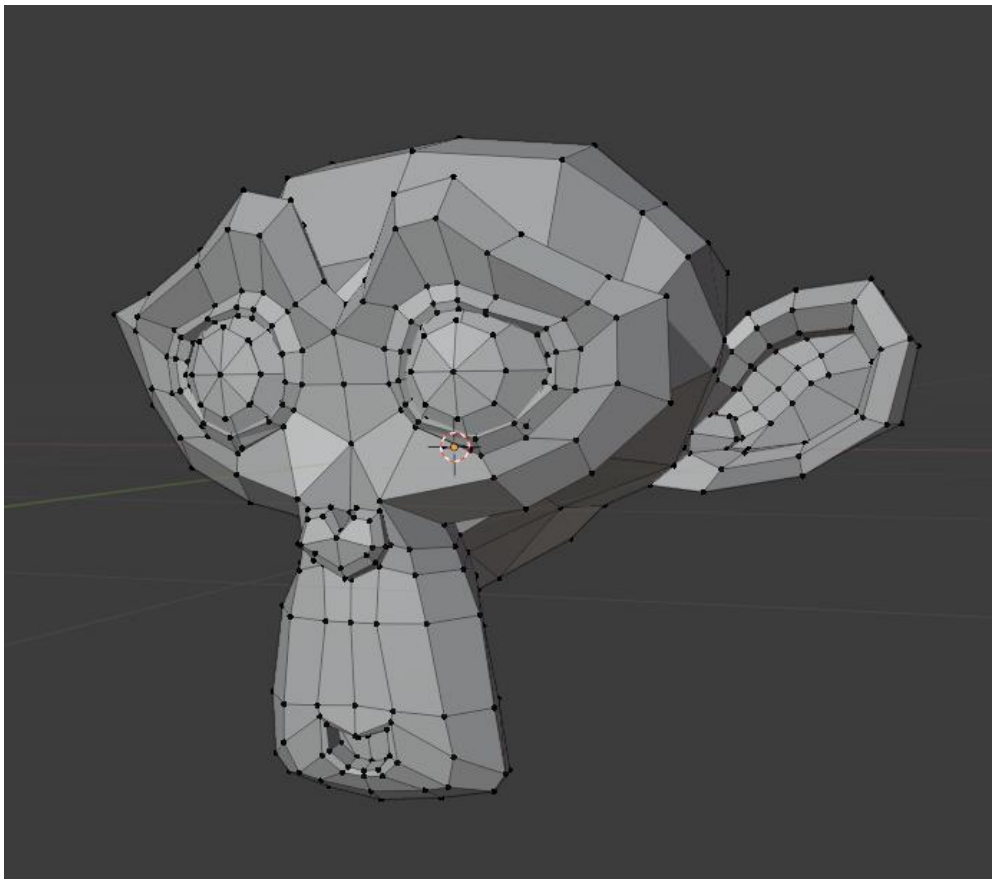

Figura 3: mesh de Suzanne

En la imagen podemos ver el escenario en el que halla el objeto, el objeto mismo y su geometría, siendo los elementos más destacados los vértices, visibles como pequeños puntos negros. Un aspecto muy relevante es que la cantidad de vértices definen el nivel de detalle de la geometría del modelo.

Realizar alteraciones en la mesh requiere de alterar la posición de los elementos de la geometría, especialmente la de los vértices. Esto se puede hacer mediante la interfaz gráfica, con ratón y teclado. Esta manera de por si es bastante cómoda, incluso para un usuario inexperto, pero dado que queremos incorporar funciones más complejas, requerimos de poder interactuar por comandos con la mesh.

Es aquí donde la API de Python juega un papel fundamental: Blender permite acceso a casi todas sus herramientas con métodos y funciones de su librería propia denominada *bpy*, mediante las cuales podemos acceder a la información de un objeto y a todas las herramientas para alterarlo. Las herramientas de programación internas permiten el testeo en tiempo real de scripts, cuya salida es cotejable o bien en el escenario tridimensional o en la herramienta de consola incorporada.

Perfiles de usuario del programa, desde programador a usuario básico.

Una de las consideraciones que se han realizado a lo largo del trabajo es la probabilidad de que múltiples usuarios vayan a dar uso de ella, cada uno con un manejo muy distinto del programa y de las herramientas informáticas en general. Un experto en reconstrucción facial no tiene por que tener noción alguna del funcionamiento interno del programa, y ello no tiene por qué significar un obstáculo a la hora de realizar sus tareas. Así mismo puede darse el caso de un usuario que solo quiera hacer uso de un editor creado previamente y que no disponga ni conocimientos en reconstrucción facial ni en programación, y que sin embargo sea capaz de hacer uso de la aplicación como si se tratase de un "editor de personaje" de un videojuego.

Por ese motivo, se ha creado una interfaz de usuario dirigida a personas que no tengan conocimientos de programación, para que puedan crear perfiles de edición e interactuar con ellos de manera sencilla y eficaz.

El programa está enfocado pensando en tres niveles de usuario: programador, diseñador y usuario básico.

- **Programador**: grupo de usuarios que entienden la arquitectura interna del programa, tanto a nivel de código como conceptos teóricos, y saben cómo interactuar con la API de Blender; son capaces de hacer tareas de testeo y de interactuar por comandos con el entorno.
- **Diseñador**: grupo de usuarios que desean crear un perfil de edición; no necesitan conocer la arquitectura a nivel de código, pero si deben entender el concepto de zonas, macrozonas y transformaciones, así como deben saber manejar todas las herramientas presentes en la UI. Este grupo estará compuesto en parte por profesionales de la construcción facial, que reflejará sus conocimientos diseñando el perfil de edición, por lo que se espera también que tengan un manejo básico de Blender.
- **Usuario básico**: grupo de usuarios destinado a interactuar con un editor ya creado, no tienen por qué tener conocimiento de la arquitectura interna, ni a nivel de código ni teórico, así como tampoco se espera que sepan el funcionamiento de todas las herramientas de la UI. Este grupo interacciona con el programa de una manera superficial y sencilla, muy similar a un editor de personajes de un videojuego. Se requiere un manejo muy básico de Blender.

La **figura 4** muestra visualmente las distintas competencias de los perfiles de usuario.

	Modo de usuario		
Conocimiento necesario:	**Programador**	**Diseñador**	**Usuario básico**
Interacción por comandos con la mesh del objeto:	✓		
Conocimiento a nivel principiante del entorno Blender	✓	✓	
Conocimientos básicos del entorno Blender	✓	✓	✓
Conocimiento total de las herramientas del programa	✓	✓	
Conocimiento de las herramientas básicas del programa	✓	✓	✓

Figura 4: modos o perfiles de usuario y las competencias asociadas.

Vértices, acceso e interacción.

Los vértices son los elementos en los que se apoya el trabajo, debido a los siguientes motivos:

- Son la unidad básica para interactuar con la geometría de la mesh.
- Están identificados por un valor de índice.

Este último punto es también importante, aunque por un motivo menos evidente, dado que el índice es la puerta entre el script y el entorno tridimensional: podemos seleccionar un número de índice y obtener el vértice asociado, así como seleccionar un vértice y obtener su número de índice. Es una información que permite identificar de manera única a un elemento de la geometría y que es además accesible y almacenable, permitiendo todo tipo de trabajo con ella. A continuación (**figura 6**) se muestra un cubo en modo edición, cuyos índices de vértice son visibles.

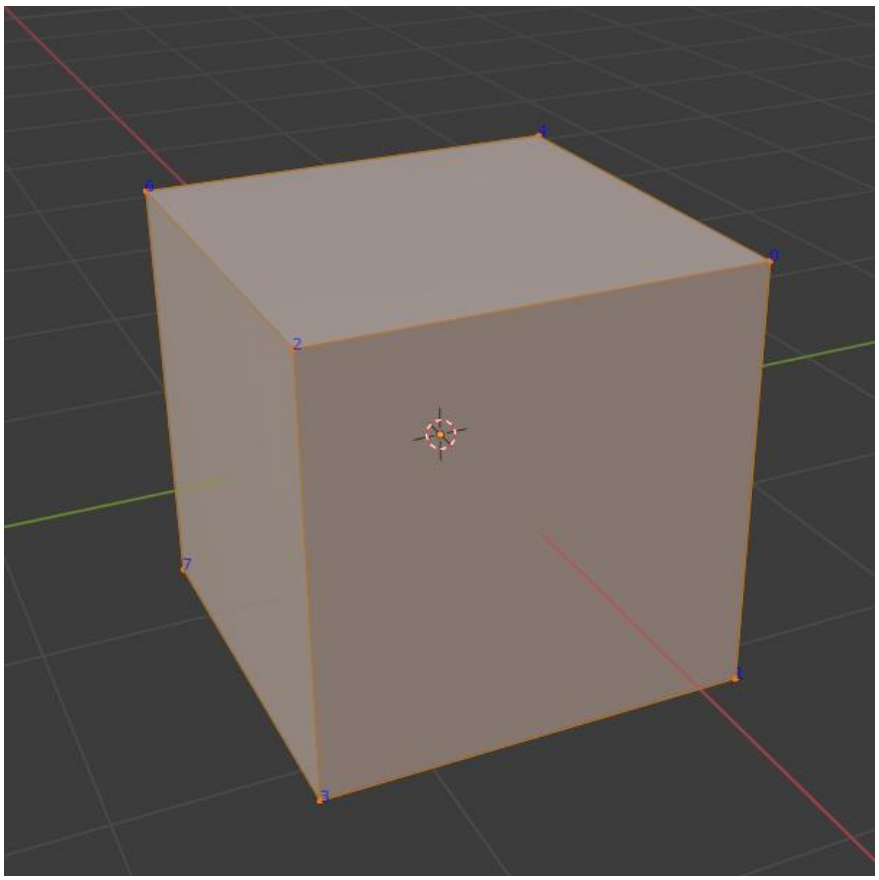

Figura 5: mesh de un objeto con índices de vértice visibles.

Dentro del contexto de Blender, distintos vértices pueden juntarse en un grupo de índices, que se identifica con un nombre propio. Todos los vértices del grupo se comportan como uno solo en translaciones, rotaciones y demás transformaciones, y además son accesibles con comandos propios.

Paneles de UI e interacción con el usuario.

El enfoque que he querido darle a este programa es de un add-on de Blender, y como tal todo el software implementado es accesible desde el mismo Blender, evitando problemas de compatibilidad y permitiendo apoyar mucho de su funcionamiento en las múltiples utilidades con las que cuenta. Las herramientas más visibles son sin duda los paneles de UI o *user interface,* que permiten acceder a distintas herramientas con un solo clic. Están situados a la derecha del escenario principal o *layout*, una ubicación más que conveniente para un usuario que quiera utilizar el programa. En la **figura 6** se encuentra la versión actual de la interfaz de usuario del programa, con todas las herramientas a disposición del usuario

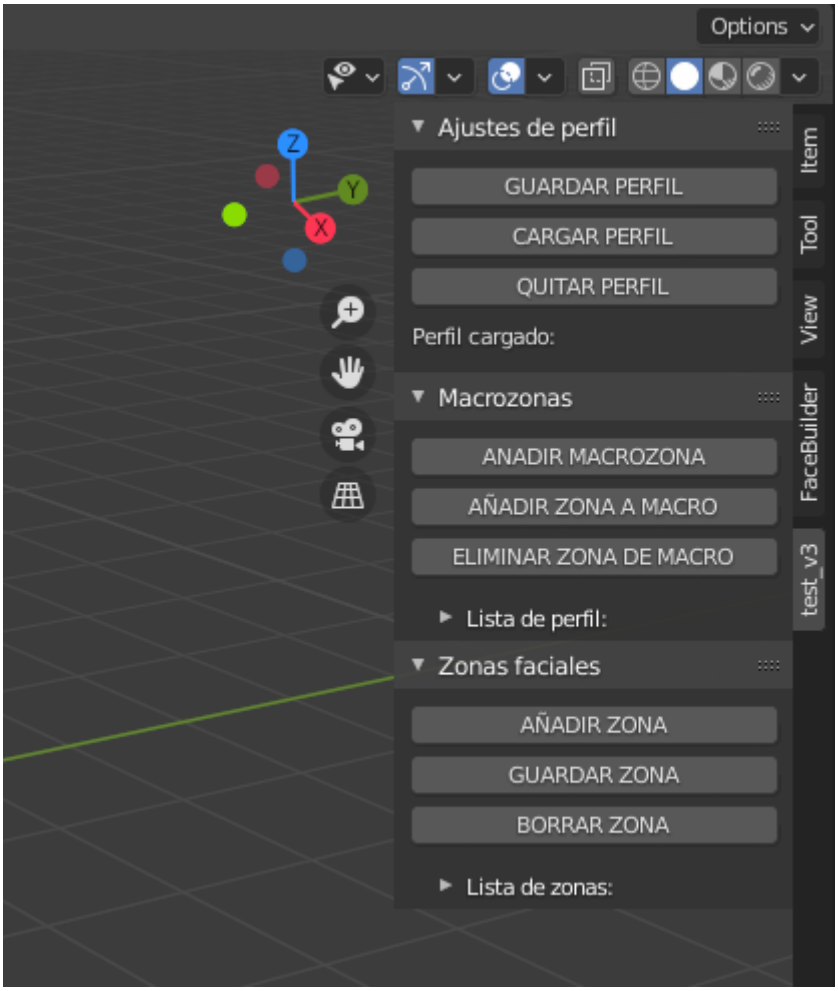

Figura 6: versión actual de la interfaz de usuario de la aplicación

En el **capítulo 4** se aclarará como instalar Blender y la aplicación, además dará unos conocimientos básicos para interactuar tanto con Blender como con el programa.

Operadores.

Para conciliar la funcionalidad de los scripts con la necesaria interacción con el usuario, es necesaria la construcción de operadores: un operador permite la ejecución de un script haciendo clic a un botón en Blender, o desplegar un menú gracias a la ejecución de un script; es el punto de unión entre interfaz de usuario y script [3].

Un operador es semejante a una clase de objeto, que permite describir a uno mediante características y métodos. Una característica fundamental de los operadores es que no retornan ningún valor, por lo que cosas como la persistencia de variables y el retorno de parámetros han tenido que ser incorporadas por otros métodos, como operadores de dialogo para parámetros de entrada, variables globales o lectura y escritura en ficheros.

Zonas faciales y primera versión de persistencia de información.

Mientras que acceder a los grupos de vértices es sencillo, el acceso a la información contenida en un grupo es bastante complicado. Esto es, sin embargo, un requisito imprescindible para incorporar los grupos de vértices y sus posibilidades a una arquitectura más compleja, por lo que para ello se ideó el concepto de "zona facial", que se manifiesta como una lista de Python en la que están contenidos el nombre del grupo de vértices y los índices de los vértices que lo componen.

Tener la información en texto plano, en una lista de Python, hace que sea útil para todos los scripts destinados a emplear dicha información.

Como consecuencia directa, esta información puede ser guardada en un documento ***txt***, en una ruta que puede ser determinada por el usuario, lo que constituye el primer paso de lo que será el sistema de archivos.

Cada zona facial tiene la información del grupo de vértices asociado a ella, y su utilización sirve a nivel de programa para importar o exportar la información de los grupos de vértices fuera de Blender, y a nivel teórico sirve para identificar con términos similares un grupo de vértices con la macrozona a la que está asociado (es más fácil decir que una macrozona está compuesta por varias zonas faciales que por distintos grupos de vértices). A nivel teórico pueden usarse indistintamente grupo de vértices o zonas faciales, y se usa tanto "zona facial" como "zona" a lo largo de la memoria.

Arquitectura del programa y perfiles de edición.

La arquitectura del programa utiliza como base los grupos de vértices, pero una gran parte del contenido que se apoya en ellos no es representable de manera explícita en el entorno de Blender: hace falta una capa de código que haga de enlace entre él y la interfaz de usuario, que a su vez debe representar explícitamente esos conceptos para que el usuario trabaje con ellos.

La interacción entre el código de enlace y la información con la que trabaja se fundamenta en un mecanismo de gestión de ficheros, que además permite la persistencia total del trabajo realizado más allá de un escenario concreto.

En la siguiente figura (figura 7) puede verse el esquema del funcionamiento de esta estructura.

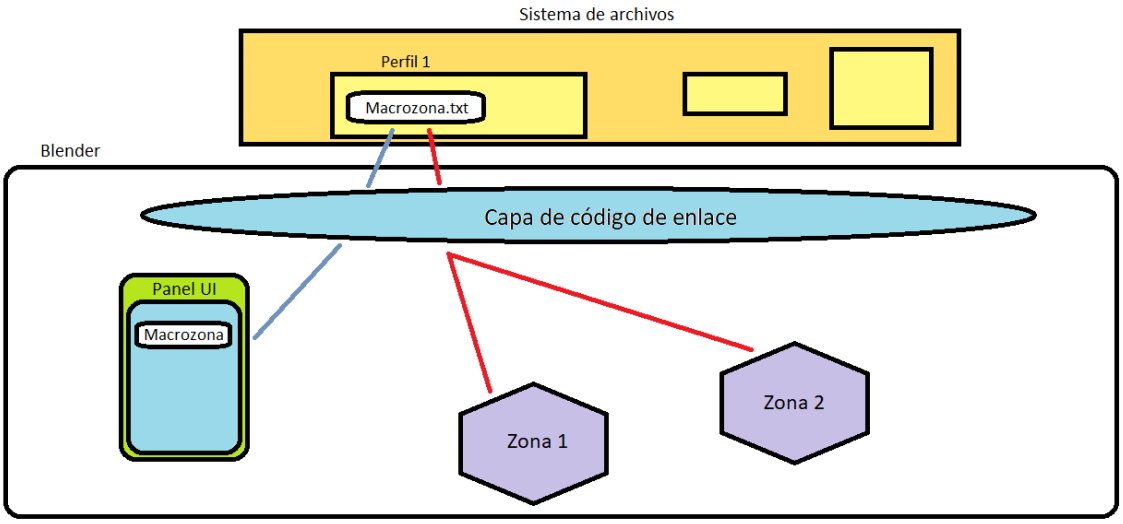

Figura 7: comportamiento del programa al seleccionar una macrozona

En la imagen se ve el procedimiento que sigue el programa al seleccionar una macrozona; esta puede ser por ejemplo una nariz, compuesta por las zonas faciales "punta de la nariz" y "tabique nasal": al ser seleccionada la nariz, el programa busca esa macrozona y obtiene la referencia a las zonas faciales asociadas, que selecciona a continuación.

La capa de código de enlace está compuesta por los distintos operadores desarrollados, en especial los que interactúan directamente con los ficheros.

La información externa sigue una estricta jerarquía de archivos, que gira en torno al concepto de perfil de edición: este recoge toda las opciones e información de un perfil y lo almacena en un directorio con nombre propio, separando y organizando el contenido para adaptarse a distintos usuarios.

Cada perfil de edición contiene la información de las zonas faciales, macrozonas y transformaciones de un objeto concreto, definido por la forma general y la indexación de índices que tiene. Todos los objetos que compartan esos atributos son considerados a criterio del proyecto como objetos similares, y son compatibles a nivel de perfil de edición.

La flexibilidad que aporta un perfil de edición permite alternar distintos editores sobre un mismo objeto, cada uno encargándose de partes concretas del mismo, o poder hacer perfiles diferentes por cada tipo de objeto que se desee editar, siempre en el mismo entorno.

Macrozonas y variables globales.

Para explicar el concepto de la macrozona se requiere de un ejemplo: supongamos que se pretende seleccionar la nariz del modelo de cabeza que tenemos en el escenario para decidir qué transformaciones sobre ella queremos hacer; si la nariz se entendiese como un único grupo de vértices, las transformaciones afectarían a todos por igual, imposibilitando cualquier transformación compleja como alterar el pronunciamiento del tabique nasal, o el tamaño de las fosas nasales. La nariz está compuesta por numerosas partes, que interesa poder modificar por separado, como muestra la siguiente imagen (**figura 8**).

Figura 8: Partes externas de la nariz. Fuente: enlace.

Es por ello que el concepto de macrozona es necesario: una macrozona es un conjunto de zonas faciales, asociadas bajo un nombre de grupo común, pero siendo cada una independiente al resto; esto permite que un usuario seleccione la nariz, que se entiende ahora como un conjunto de zonas faciales, y realice transformaciones que solo afecten a ciertas zonas concretas. La **figura 9** muestra la composición de una macrozona.

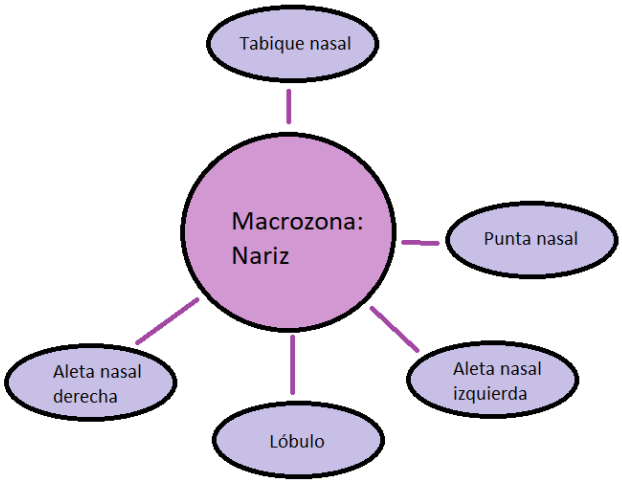

Figura 9: ejemplo de macrozona y de las zonas asociadas

Las macrozonas no pueden existir dentro del escenario de Blender, por lo que están concebidas como documentos *txt* constituidos por los nombres de las zonas faciales y de las transformaciones asociadas.

Nariz.txt → [5, 'Tabique nasal', 'Punta nasal', "Aleta nasal izq", "Aleta nasal der", "Lóbulo", 0]

La línea anterior indica la composición del ***txt*** que caracteriza a la macrozona de la **figura 9**: el valor numérico de la primera posición indica el numero de zonas faciales asociadas a la macrozona, que se encuentran contenidas en las 5 posiciones siguientes. El valor numérico que sigue (0), indica el número de transformaciones asociadas, que se posicionarían de manera análoga a las zonas faciales.

Cuando un usuario selecciona una transformación de una macrozona, Blender accede a la información de esta y realiza cambios en los grupos de vértices pertinentes, que sí que existen en el contexto de Blender, por lo que la comunicación entre usuario y entorno se establece.

Para implantar las macrozonas de manera efectiva, ha sido necesaria la creación de una variable que se almacenase en el contexto de la escena, con la cual el programa puede interactuar en cualquier momento, solventando las limitaciones de los operadores.

Transformaciones.

Las transformaciones están asociadas a una macrozona, y están concebidas como alteraciones espaciales que son realizadas a uno o más grupos de vértices. Estas tampoco pueden residir como tales en Blender, por lo que dependen de las herramientas creadas anteriormente para poderse realizar. Pese a no haber podido ser integradas a nivel de software, sí que se ha podido elaborar la idea en plano teórico.

Las transformaciones están compuestas por las zonas a las que afectan, la transformación realizada y algunos parámetros auxiliares como el vector de la transformación.

Dentro de cada macrozona están almacenados los nombres de las transformaciones que tienen asociadas, aunque cada una cuenta con su propio fichero donde están almacenadas sus características. La intención detrás de este diseño es que, al seleccionar una macrozona, se despliegue un menú en el que se muestren las zonas que la componen y las transformaciones que se le pueden aplicar, semejante a un editor de personajes al uso.

El obstáculo más importante que se presenta es la manera de registrar dichas transformaciones: el objetivo es lograr que una transformación se registre de manera sencilla y eficiente y con una precisión suficiente, sin que eso signifique un lastre importante para el diseñador.

Los dos caminos explorados son:

Realizar una transformación a un grupo en el escenario mediante los controles de ratón y teclado, y que el programa la registre para poder repetirla. Esta solución se presenta como la más adecuada, ya que da total libertad al diseñador y es fácil de usar.

Introducir en un formulario los parámetros deseados y que el programa realice la transformación. Aun siendo la solución más sencilla de implementar, presenta muchas trabas al diseñador.

Capítulo 3 Desarrollo de la aplicación:

En este apartado se describen las partes del código más relevantes en relación con los conceptos teóricos explicados anteriormente.

La librería de Blender y nuestra propia extensión.

Blender cuenta con su propia librería de Python, que contiene métodos que permiten acceder a la casi totalidad de las herramientas disponibles en el entorno, lo que ha hecho relativamente sencillo interactuar con el entorno.

La librería de blender se denomina bpy, y es a partir de ella que se localizan las extensiones que contienen las distintas utilidades.

Las extensiones que se han empleado son:

bpy.data: con ella se accede a los datos internos de Blender, entre sus posibilidades está la de encontrar la ruta donde está ubicado el programa, útil para registrar módulos.

bpy.context: con ella se accede a la información contenida en el contexto, permite trabajar con el material en escena y hace posible la interacción entre los operadores y el objeto activo (*bpy.context.active_object*). A partir de esta última interacción podemos trabajar con los grupos de vértices del objeto, como se detallará más adelante.

bpy.props: permite trabajar con propiedades, que hacen la vez de variables: su principal uso ha sido en operadores de dialogo, que demandan información por parte del usuario antes de ejecutarse, ya que se comportan como argumentos de entrada. Un uso particular pero imprescindible de esta extensión es la creación de la variable global *perfil_elegido*, una propiedad de tipo *string* del contexto que permite a Blender saber con qué perfil de edición se está trabajando en el momento.

bpy.types: con esta extensión se crean los operadores, que son el núcleo de la funcionalidad del programa (*bpy.types.Operator*), así como los paneles de interacción con el usuario (*bpy.types.Panel*). También permite habilitar la propiedad que establece el perfil de edición elegido (bpy.types.PointerProperty).

bpy.ops: permite hacer operaciones varias, entre ellas cambiar el modo en el que se encuentra el objeto (**p.e** *bpy.ops.object.mode_set(mode = 'EDIT')*) o realizar transformaciones en los grupos de vértices seleccionados (**p.e**. *bpy.ops.transform.resize*). Los operadores que se han creado para este proyecto emplean la extensión personalizada *bpy.ops.**pep***.

bpy.utils: se ha empleado sobre todo para registrar operadores o eliminarlos del registro.

Para el desarrollo del trabajo, se ha evitado emplear librerías creadas por terceros, y se han usado métodos de funciones nativas de Python como la librería *os* para el trabajo con ficheros.

Como se ha mencionado antes, se ha creado la extensión personalizada *.pep*, que se ha usado para localizar de manera sencilla los operadores del programa. No ha supuesto ningún tipo de configuración especial, ya que se crean durante el registro de las clases de los operadores.

Programar en Blender: consideraciones personales.

Blender facilita su propio entorno para editar código de Python, en el que he desarrollado en su totalidad, salvo anecdóticas excepciones. La experiencia en general ha sido muy buena, aunque se han dado casos como interrupciones en la ejecución poco predecibles, o inconvenientes para probar cambios al código, que requerían reiniciar el programa. Esto se debe en parte a que el entorno está ejecutándose a tiempo real en todo momento, y ciertas acciones parecen interrumpir la ejecución un tiempo excesivo, resultando en que Blender no responda. No obstante, la ejecución y utilización de nuestro programa no ha dado problema al respecto.

Una característica muy favorecedora de Blender es que al mantener el cursor encima de un recurso o herramienta, aparece un mensaje flotante donde se indica la extensión de bpy mediante la cual se puede obtener dicho recurso empleando comandos a la consola. Sin embargo, en ocasiones este comando no es efectivo y es necesaria una breve búsqueda para averiguar el comando adecuado.

Para entrar en este entorno debemos elegir el modo Scripting en la parte superior de la ventana: al hacerlo la ventana del escenario disminuirá de tamaño y tres nuevas ventanas ocuparan el espacio liberado, como puede verse en la imagen de la **figura 10**.

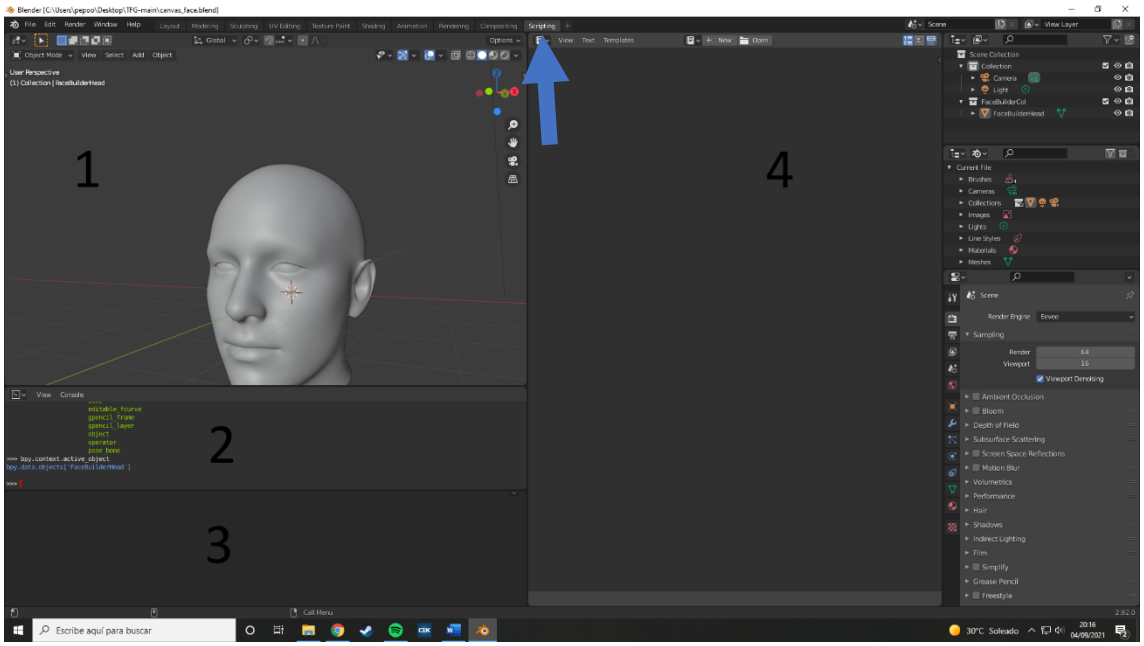

Figura 10: distribución de ventanas del modo Scripting de Blender.

En la figura se pueden ver las distintas ventanas en las que se divide la pantalla al seleccionar el modo scripting (seleccionado en el botón indicado por la flecha azul).

La **ventana 1** sigue mostrando el escenario con el objeto que ya se mostraba en el modo layout u objeto, que es el modo por defecto al abrir Blender.

En la ventana inmediatamente inferior a la del objeto (**ventana 2**) pueden introducirse comandos y comprobar su funcionamiento, siendo muy útil en la creación de scripts más complejos.

La ventana inferior (**ventana 3**) muestra información de las acciones que se ejecutan en el entorno general: acciones como alterar la mesh o geometría del objeto con una transformación

con el ratón o guardar los cambios en la edición de un script se reflejan en esta ventana. Esta ventana ha agilizado mucho el aprendizaje del entorno.

El panel de la derecha (**ventana 4**) es un editor de código al uso, e incorpora las funciones clásicas de uno. Al abrir un script o al guardar uno recién creado aparece la opción de ejecutarlo

Para ver los resultados completos es mejor emplear la ventana de consola (se accede pulsando *Window -> Toggle System_Console*), en la que se muestra en profundidad el resultado de la ejecución del código y es la que se debe consultar para obtener información de la ejecución, así como los errores que pudieran surgir. Para imprimir mensajes concretos se ha utilizado la función *print()*.

Operadores.

Los operadores son, como se explicó anteriormente, el puente entre la interfaz de usuario y la funcionalidad bruta del código. Han sido necesarios para casi todas las herramientas, desde añadir una zona a cargar un perfil. Para poder emplear los operadores creados, es necesario registrarlos con el método *bpy.utils.register_class(operador)*, a partir de lo cual pueden invocarse en la consola del modo scripting.

Los operadores están concebidos como objetos, y cuentan con métodos propios que determinan su comportamiento en el programa. La **figura 11**, a continuación, muestra la estructura típica de un operador, si bien no están incluidas todas las funciones de las que dispone.

```
#aqui se importan las librerias necesarias para el funcionamiento del operador
import bpy
import os

#definicion de la clase del operador
class cargar_grupos_vertices(bpy.types.Operator):

    #id del operador, se guardará en la extension bpy.ops.pep.cargar_grupos_vertices
    bl_idname = "pep.cargar_grupos_vertices"

    #la etiqueta del operador, el botón que invoque a este operador tendra como nombre la etiqueta
    bl_label = "CARGAR GRUPOS VERTICES"

    #opciones de despliegue del operador
    bl_options = {'REGISTER'}

    #Ejemplo de propiedad
    directory : bpy.props.StringProperty(
        name="Directorio seleccionado",
        description="El directorio cuyo escenario cargaremos"
        )

    def execute(self, context):
        #Aqui se encuentra el código que ejecuta la funcionalidad del operador
        return {'FINISHED'}

    def invoke(self, context, event):
        #Aqui están las acciones que se desarrollan al invocar al operador
        #Este método se suele emplear en operadores que solicitan
        #informacion previa al usuario antes de ejecutarse
        return {'RUNNING_MODAL'}

#Las lineas comentadas a continuación sirven para tareas de testing

#registro de la clase para poderse utilizar
#bpy.utils.register_class(cargar_grupos_vertices)

#invocación por comando del operador
#bpy.ops.pep.cargar_grupos_vertices('INVOKE_DEFAULT')
```

Figura 11: estructura del operador cargar_grupos_vertices, ya obsoleto.

En la figura pueden verse comentadas las funciones de cada parte del operador.

Los operadores no retornan ningún valor a parte del estado del mismo al final de la ejecución, cuyos valores están dentro de una lista fija [4], por lo que es necesario un sistema a parte para almacenar información que requiera persistir más allá del operador.

Arquitectura del código.

La estructura a nivel de código del programa es bastante sencilla y está compuestas por dos bloques esenciales:

- Bloque principal: compuesto por un único script, denominado de manera temporal ***testv3.py*** y de manera final ***main.py***, en el que se encuentran las clases de los paneles de la interfaz de usuario y el código necesario para desplegar la aplicación, como registrar los distintos operadores.
- Bloque de operadores: compuesto por los scripts restantes en el que se declaran las clases de los operadores que son registrados en el bloque principal. El código perteneciente a este bloque apenas interacciona entre sí, sirven de herramientas al bloque principal.

Para poder compartir información entre operadores y bloques se ha creado un sistema de archivos externo a Blender. La arquitectura general se muestra en la siguiente figura (**figura 12**).

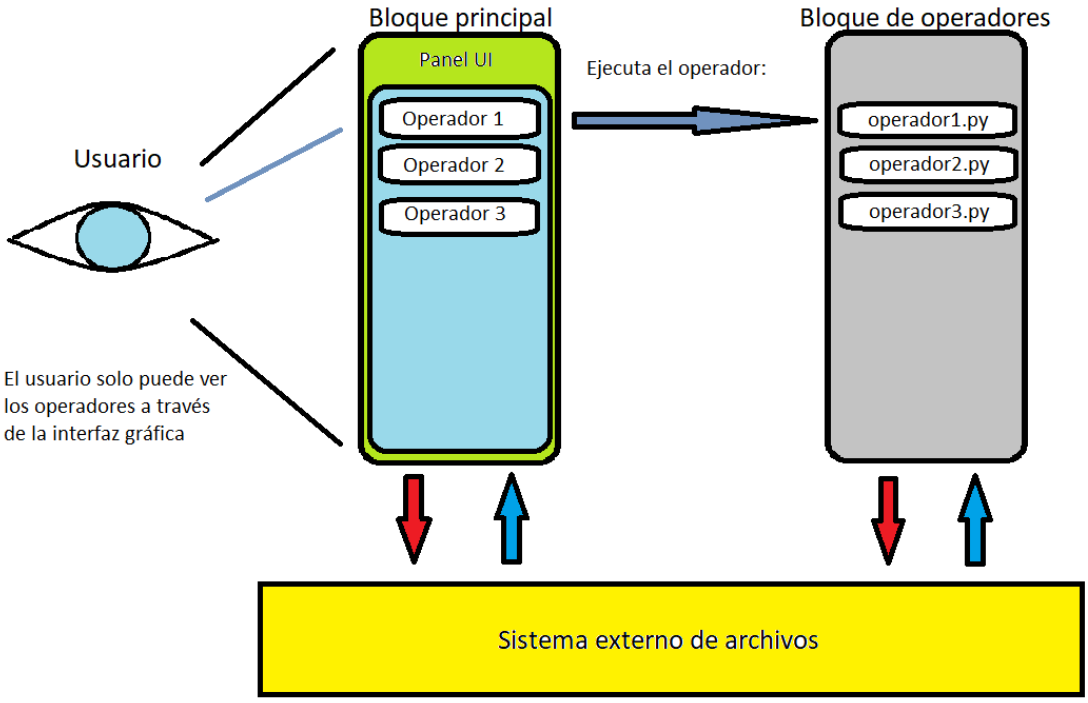

Figura 12: arquitectura general del funcionamiento del programa

Al ejecutar el script ***main.py*** se registran todas las clases, desde paneles a operadores, y se despliega la interfaz de usuario que muestra las herramientas a nuestra disposición. Cuando el usuario selecciona uno de los botones que representan a los operadores se ejecuta el operador correspondiente.

La información generada durante la ejecución de esos operadores se puede almacenar en el sistema de archivos externo para poder acceder a ella en cualquier momento.

Interfaz de usuario y función draw.

Los paneles de interfaz de usuario permiten acceder a las distintas herramientas sin tener que preocuparse por el código que yace debajo de ellas. Estos paneles permiten a un usuario sin nociones de programación interactuar con los operadores del programa sin el uso de comandos, solo mediante uso del ratón.

Para que pueda llevarse a cabo esta tarea, además de la infraestructura que se ha desarrollado, se ha contado con las funciones ***draw*** de los paneles: estas funciones representan en la interfaz los elementos deseados, que en el caso han sido en su mayor parte los diferentes operadores del programa. Un rasgo muy valioso de estas funciones es que representan los elementos a tiempo real, lo que permite añadir un elemento nuevo sin tener que actualizar manualmente la interfaz de usuarios.

Las macrozonas solo pueden ser representadas explícitamente aquí, y para poderse visualizar se ha empleado el siguiente fragmento de código dentro de la función ***draw*** (**figura 13**):

```
class lista_macrozonas(panel_editor, bpy.types.Panel):
    bl_parent_id = "macrozonas_editor"
    bl_label = "Lista de perfil:" # + bpy.context.active_object.pep.perfil_elegido
    bl_options = {'DEFAULT_CLOSED'}
    def draw(self,context):
        file_path = os.path.dirname(bpy.data.filepath)
        actual_filepath = os.path.join(file_path, "Presets")
        perfil = bpy.context.active_object.pep.perfil_elegido
        perfil_filepath = os.path.join(actual_filepath,perfil)
        macrozonas_filepath = os.path.join(perfil_filepath, "Macrozonas") #/Macrozonas
        if os.path.exists(macrozonas_filepath):
            for v in os.listdir(macrozonas_filepath):
                menu_macro = self.layout.operator('pep.menu_macrozona', text = v.split(".")[0])
                menu_macro.macrozona = v.split(".")[0]
        else:
            pass
```

Figura 13: clase lista_macrozonas de main.py

Este código accede al directorio de las macrozonas de un perfil determinado y las muestra en la interfaz de usuario como un botón que ejecuta el operador *menú_macrozona*, que a su vez despliega un menú con las transformaciones disponibles para dicha macrozona.

Este fragmento de código sirve para mostrar de forma análoga las distintas zonas faciales, ya que se muestra para cada zona el operador *selecciona_zona*, que permite visualizar los vértices asociados.

Acceso a los vértices.

Anteriormente explicamos que el programa se fundamentaba en la interacción con los vértices de la mesh del objeto. Para ello es necesario poder acceder a su información y, por ende, acceder a cada uno mediante un identificador único.

El índice de vértice es un valor de tipo entero que identifica a un vértice en un objeto, es el que se usa como puntero en los métodos relacionados. En la imagen siguiente (**figura 14**) se pueden visualizar los índices de los vértices del objeto, cuyo número es visible gracias a una opción para desarrolladores que es posible habilitar, como se detallará más tarde.

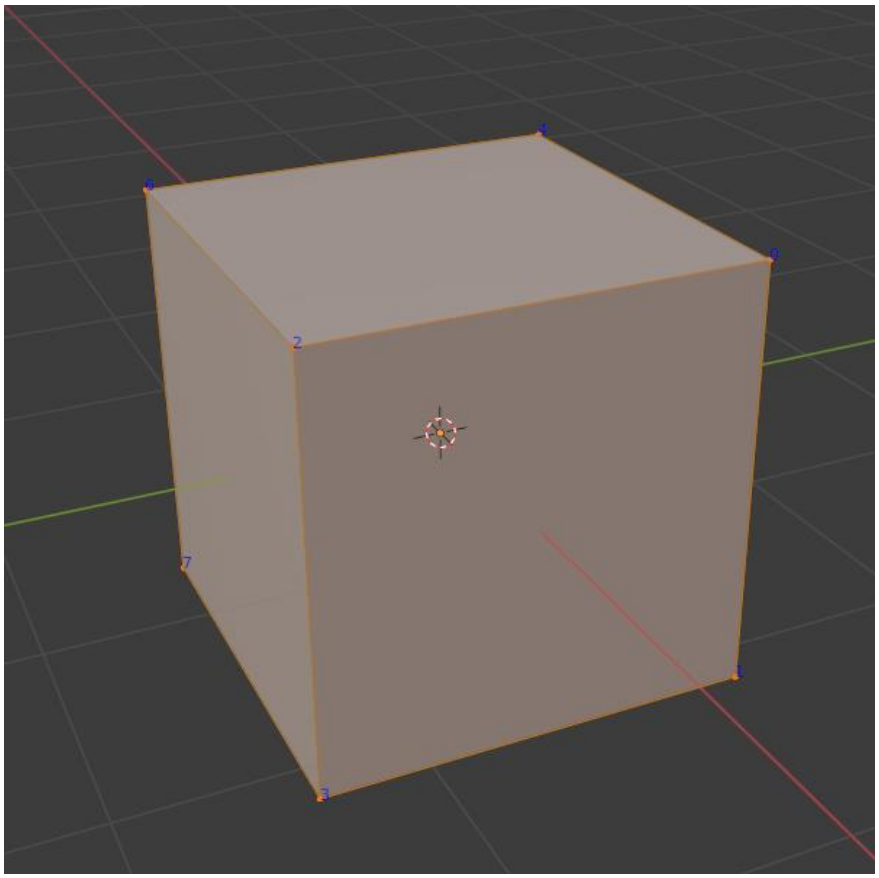

Figura 14: *mesh* de un cubo con índices de vértices visibles

Entre los métodos que se han utilizado en versiones actuales o anteriores para trabajar con vértices están los siguientes:

vertice = bpy.context.active_object.data.vertices[<número de indice>] -> accedes al vértice del objeto actualmente seleccionado. La variable resultante puede acceder al método select.

vertice.select -> este método tiene dos usos:

- **seleccionado = vertice.select()** -> permite saber si el vértice está seleccionado o no.
- **vertice.select = <True o False>** -> permite establecer la selección de un vértice.

El desarrollo que ha tenido este aspecto del proyecto ha dictaminado el camino que iba a recorrer en el futuro, siendo la primera versión un script denominado *alterar_ancho_ojos.py* que selecciona de vértice en vértice y realiza en cada uno una transformación, dando como resultado una transformación compleja que es en este caso el aumento del ancho de los ojos.

Este script funcionaba adecuadamente y era una muy buena primera aproximación, pero fue evidente al poco tiempo de que era una solución poco eficiente y se acabó descartando, los motivos son los siguientes:

Al seleccionar y deseleccionar un vértice se requiere el cambiar el modo en el que se encuentra el objeto a modo “OBJETO” (*bpy.ops.object.mode_set(mode = 'OBJECT')*), mientras que para realizar una transformación al vértice seleccionado se requería cambiar a modo “EDICION” (*bpy.ops.object.mode_set(mode = 'EDIT')*). El flujo del script a grandes rasgos consistía en

cambiar a modo objeto, seleccionar un vértice, cambiar a modo edición, aplicarle una transformación, cambiar otra vez a modo objeto y seleccionar otro vértice, como puede verse en el siguiente esquema (**figura 15**).

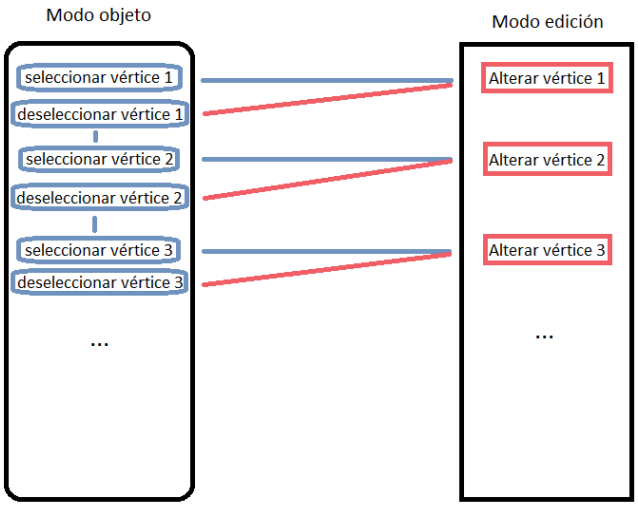

Figura 15: procedimiento seguido para seleccionar 3 vértices y alterarlos en la primera versión

Este número de transiciones entre modos era muy inestable hasta cuando se operaba con un número muy pequeño de vértices, llegando incluso a interrumpir Blender.

El segundo motivo es que la manera de seleccionar vértices individualmente lo hace muy poco atractivo de cara a diseñar herramientas: en muchas ocasiones se deben manipular varios vértices seguidos, a los que les aplica una misma función o transformación; en estas situaciones este método no aporta ganancia alguna y solo acaba lastrando el rendimiento del programa.

A raíz de los resultados palpables y de una rápida investigación [5], se optó por cambiar de modelo, manteniendo la utilidad de los índices de los vértices pero cambiando el enfoque a uno más estable y flexible de cara a las herramientas posteriores.

Grupos de vértices.

Esta es la evolución del concepto expuesto anteriormente: Blender conoce como grupo de vértices a un conjunto de vértices de la mesh, que se identifica bajo un nombre propio y único, es accesible a través de métodos propios y donde todos los vértices se comportan como uno. Esto es extremadamente útil en primera instancia, comparándolo con el método anterior, ya que reduce el número de transiciones de modo del objeto si se realiza una transformación y permite gestionar los vértices de una manera más ordenada y sencilla en todos los scripts. El siguiente esquema sirve de comparación directa con el método anterior (**figura 15**), también expuesto en imagen (**figura 16**).

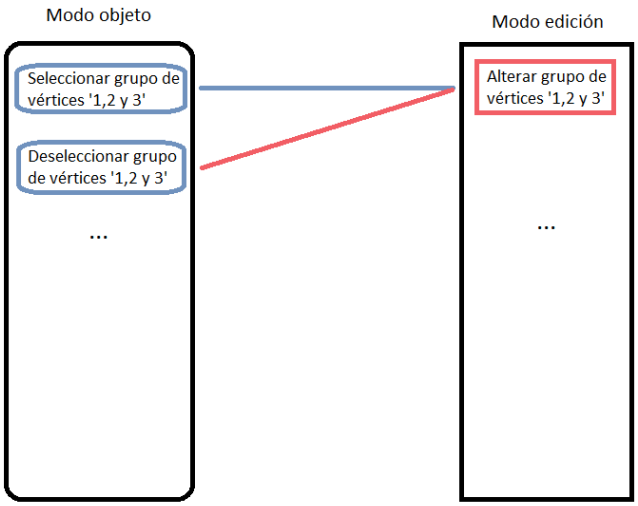

Figura 16: procedimiento seguido al seleccionar un grupo de 3 vértices y alterarlo, en la versión actual.

Los métodos de Blender que han sido utilizados en el trabajo han sido:

nuevo_grupo *= bpy.context.active_object.vertex_groups.new(name=**'nombre'**)* → asocia a una variable un grupo de vértices con nombre propio.

nuevo_grupo*.add(**lista_indices**, 1.0, 'ADD')* → añade una lista de índices de vértices al grupo de vértices asociado a la variable creada anteriormente.

*bpy.context.active_object.vertex_groups.remove(**nombre**)* → elimina el grupo de vértices identificado por el nombre escogido.

grupo *= bpy.context.active_object.vertex_groups.get(**nombre**)* → obtiene el grupo indicado por el nombre y lo asocia a una variable de tipo "vertex group", especifica de los grupos de vértice.

El esquema que sigue el programa en la aplicación de estos métodos es el siguiente: al emplearse el primer método se crea un grupo de vértices con el nombre elegido (en estre caso **nombre**), al que se le añade una lista con los índices de los vértices que se desea que pertenezcan al grupo (la lista está denominada en este caso como **lista_indices**). Una vez creado, este grupo existe en Blender como tal y puede ser eliminado o asociado a otra variable con los dos métodos restantes, respectivamente.

Los grupos de vértices existen dentro del contexto de Blender y, por lo tanto, pueden ser accesibles directamente por cualquier parte del programa. Al seleccionar una macrozona y al hacerle una transformación, siempre se estará trabajando en última instancia con grupos de vértices.

Zonas faciales.

Los grupos de vértices son los elementos con los que interactúan la mayoría de los operadores del programa y los que serán afectados por las transformaciones. Sin embargo, estos no se pueden exportar como tales fuera de Blender y además es bastante complicado acceder a la información contenida en ellos, como los índices de los vértices que los componen. Por ello, y dado que la información debe ser almacenada en un sistema de archivos externo para poder ser

persistente y debe ser fácilmente accesible, se ha creado el concepto de zonas faciales: son listas de Python en las que están contenidos los índices de los vértices que comprenden un grupo de vértices determinado, precedidos por el nombre del mismo grupo. Un ejemplo de zona facial es el siguiente:

Punta nariz: [**'Punta nariz'**, 276, 338, 360, 416, 417, 439, 550, 551]

Punta nariz es el nombre del grupo de vértices al que esta asociada esta zona facial, los valores de tipo *int* que le siguen son los índices de los vértices que constituyen dicho grupo.

Las zonas faciales son los mismos grupos de vértices, representados en una lista de Python para poder ser almacenados de manera externa a Blender. Los operadores *guardar_perfil*, *cargar_perfil* y *borrar_perfil* son los únicos que interactúan con ellas, y se encargan de exportar, importar y eliminar respectivamente los grupos de vértices asociados a esas zonas faciales en el escenario de Blender.

- **guardar_perfil**: recoge los distintos grupos de vértices de un perfil de edición y los almacena en el sistema de archivos como zonas faciales con el mismo nombre.
- **cargar_perfil**: recorre el directorio *proyecto/Presets/Perfil/Zonas*, que contiene las distintas zonas faciales de un perfil determinado, y crea en Blender tantos grupos de vértices como zonas haya en el directorio, con los mismos nombres.
- **eliminar_perfil**: elimina del entorno de Blender los distintos grupos de vértices, con la idea de reiniciar el entorno para trabajar con un perfil distinto.

Como se explicará en el manual de usuario, es imprescindible guardar las zonas faciales si se quiere dar persistencia a todo el conjunto (zonas, macrozonas y transformaciones), ya que si no están guardadas, las macrozonas pierden la referencia de las zonas faciales que la componen, así como las transformaciones.

A continuación, se muestra la parte del operador *guardar_perfil* encargada de acceder a los datos y de guardar la información (**Figura 17**).

```
        #Recorre los grupos de vertices presentes, omite temp
        #Obtiene nombre de cada grupo y su indice del grupo
        grupos_vert = [v for v in bpy.context.active_object.vertex_groups]
        for v in grupos_vert:
            #print(v.name)
            if v.name == "temp":
                pass
            else:
#                print(v.name)
                nom_grupos.append(v.name)
                idx_grupos.append(v.index)
        print(nom_grupos)
        print(idx_grupos)

        #Obtenemos los vertices de cada grupo
        for vg_idx in idx_grupos:
            #Doble bucle, encuentra a que grupo de vertices pertenecen los vértices seleccionados
            vs = [ v for v in ob.data.vertices if vg_idx in [ vg.group for vg in v.groups ] ]
            lista_escribir = []
            #Abrimos un fichero con nombre <nombre_grupo_vertices>.txt en la ubicacion deseada
            #with open(actual_filepath + bpy.context.active_object.vertex_groups[vg_idx].name + ".txt", "w") as file:
            with open(os.path.join(zonas_filepath, bpy.context.active_object.vertex_groups[vg_idx].name + ".txt"), "w") as file:
                lista_escribir.append(bpy.context.active_object.vertex_groups[vg_idx].name)
                for v in vs:
                    lista_escribir.append(v.index)
                file.write(str(lista_escribir))
        bpy.context.active_object.pep.perfil_elegido = self.my_string
```

Figura 17: principio del operador guardar_perfil.

Cabe destacar el bucle para detectar qué vértices están seleccionados y a que grupo pertenecen, implementado a partir de la información obtenida de un foro, cuya referencia no he podido encontrar de nuevo y cuyo funcionamiento a grandes rasgos es el indicado en la figura siguiente (**figura 18**).

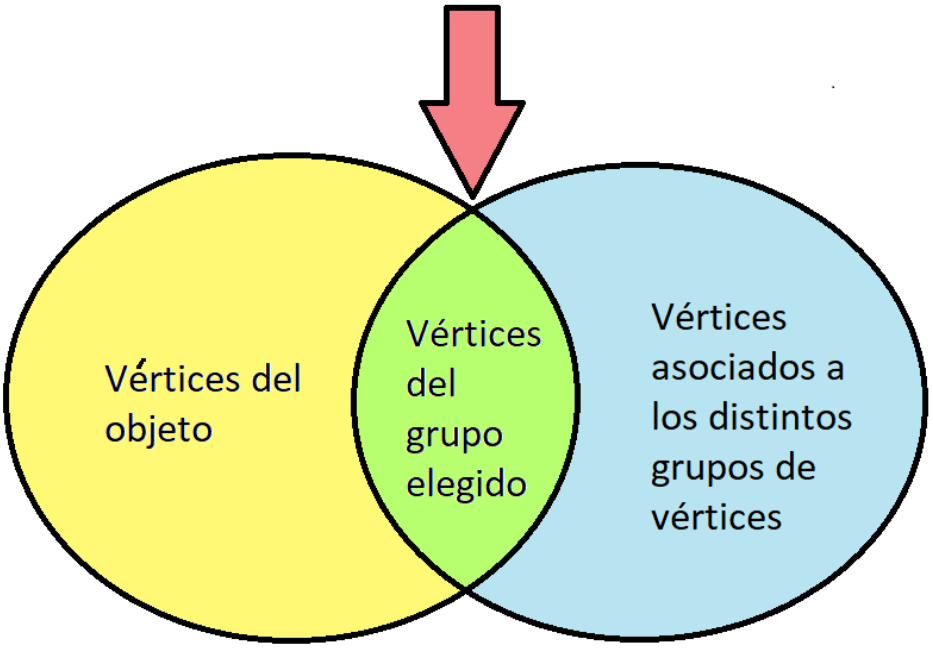

Figura 18: funcionamiento del doble bucle de selección de vértices de un determinado grupo.

El bucle va recorriendo los distintos grupos de vértices del modelo, selecciona de uno en uno y coteja los vértices del grupo con los vértices del objeto, si dichos vértices están seleccionados se almacenan junto al nombre del grupo en una variable temporal *lista_temp*, que posteriormente se escribe en un fichero de texto, que será la zona facial asociada a ese grupo.

Como pequeño apunte, en la primera versión de guardado de índices se hacía un acceso de escritura al fichero de la zona facial asociada por cada índice que se quería añadir. Posteriormente, este sistema fue descartado, y ahora los índices se almacenan previamente en la variable *lista_temp*, para luego escribirse en un solo acceso de escritura en el fichero.

La estructura del sistema de archivos se detallará a continuación, y se podrá ver que lugar ocupa cada parte del programa.

Sistema de archivos.

Dado lo compleja que puede resultar la tarea de crear un entorno de edición que cumpla con los estándares de un profesional, es lógico pensar que es necesario un método por el cual las zonas creadas se puedan independizar del archivo *blend* que las contiene en ese momento, pudiéndose exportar e importar en cualquier mesh que sea similar al objeto en donde se creó ese perfil de edición. De esta manera, no haría falta crear un perfil de edición de cero cada vez que se ejecute la aplicación.

Así mismo, las macrozonas no pueden existir como tales en Blender y los operadores no devuelven valores que puedan servir para interactuar entre ellos, limitando las opciones a crear un espacio común donde se accede a la información.

Para superar estos obstáculos, se ha creado un sistema de archivos externo a Blender, ubicado bajo el nombre de la carpeta *Presets* (nombre en la versión actual) en el directorio donde se ubica el archivo *blend* en el que está operando junto a todos los scripts del programa. La figura mostrada a continuación (**figura 19**) muestra la estructura de los directorios que tiene el trabajo actualmente.

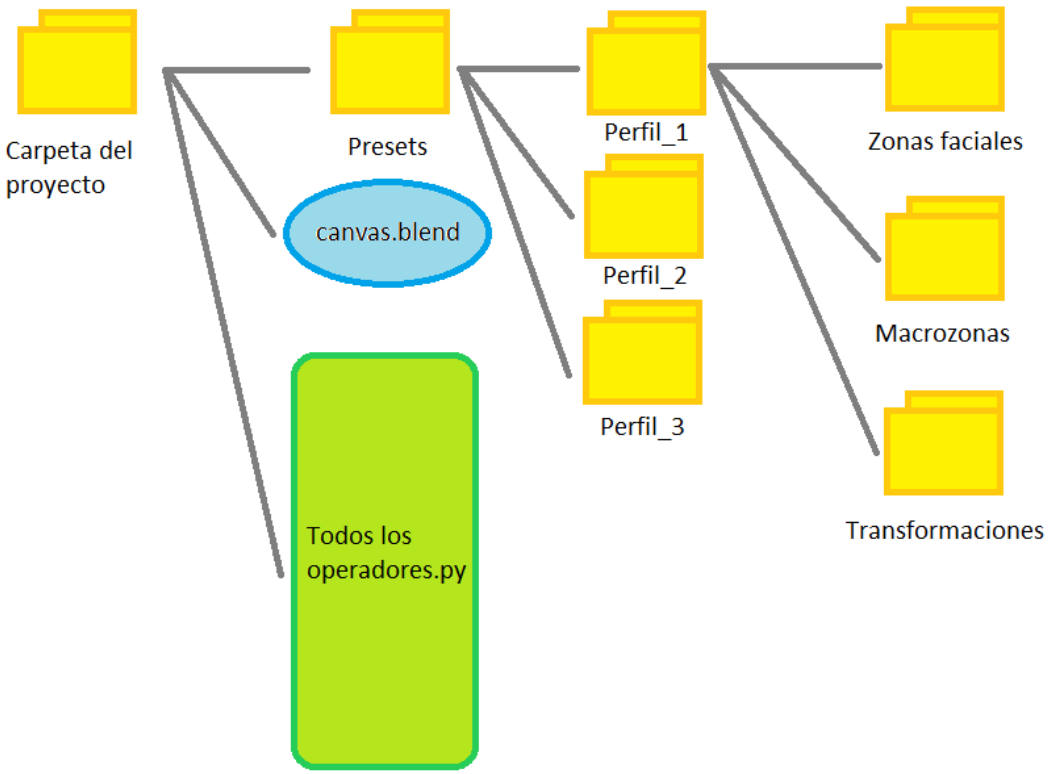

Figura 19: estructura del sistema de archivos local.

El proyecto está contenido en una sola carpeta, en la que están contenidos los distintos scripts que sostienen el código de los operadores, el archivo *blend* en el que se ejecuta la aplicación y la carpeta *Presets*, en la que está contenida localmente toda la información de los distintos perfiles de edición que se han creado anteriormente.

El archivo *canvas.blend* es el archivo en el que está contenido el modelo y es donde se está ejecutando el programa, en su mismo directorio puede añadirse a placer cualquier *blend* en el que se quiera desarrollar un editor.

Cada perfil de edición cuenta con 3 carpetas en su interior:

- Zonas faciales: en ella se encuentran tantas zonas faciales (archivos *txt*) como grupos de vértices se han creado para ese perfil de edición.
- Macrozonas: aquí están los distintos archivos *txt* que representan a las macrozonas que han sido creadas en el perfil.
- Transformaciones: aquí están los archivos *txt* que representan a las distintas transformaciones que han sido creadas.

Por último, los operadores se encuentran en los distintos archivos *py* ubicados en la carpeta del proyecto. A ellos accede *main.py* al comienzo de su ejecución, y es importante que se respete su ubicación.

La estructura actual es bastante estricta, ya que de cambiarse la ubicación de cualquiera de las partes la aplicación no funcionará correctamente. Esto es así por que muchos de los operadores dependen de una ruta relativamente fija para acceder a los datos de los perfiles de edición. Sin embargo, la aplicación funcionará correctamente en cualquier directorio convencional siempre que se respete la estructura mostrada en la **figura 19**.

Los scripts que desarrollan la mayor parte del trabajo con el sistema de archivos son los operadores contenidos en los scripts *guardar_perfil.py* y *cargar_perfil.py*, que guardan y cargan un perfil de edición respectivamente, con toda la información contenida en ellos. Otros operadores realizan también funciones sobre este sistema, pero se limita al acceso de ficheros individuales (es el caso de los operadores *anadir_macrozona* y *anadir_zona_a_macrozona*).

Macrozonas y la variable *perfil_elegido*.

Las macrozonas son el concepto más importante del proyecto: son espacios donde el usuario y el programa interactúan y son donde las transformaciones pueden tener lugar. Sin embargo, a nivel de Blender no son representables y por lo tanto requieren de las metas que han sido superadas anteriormente, junto a una variable que detallaremos un poco más tarde. En la imagen a continuación (**figura 20**) se ilustra cómo funcionan las macrozonas y en que partes se apoya.

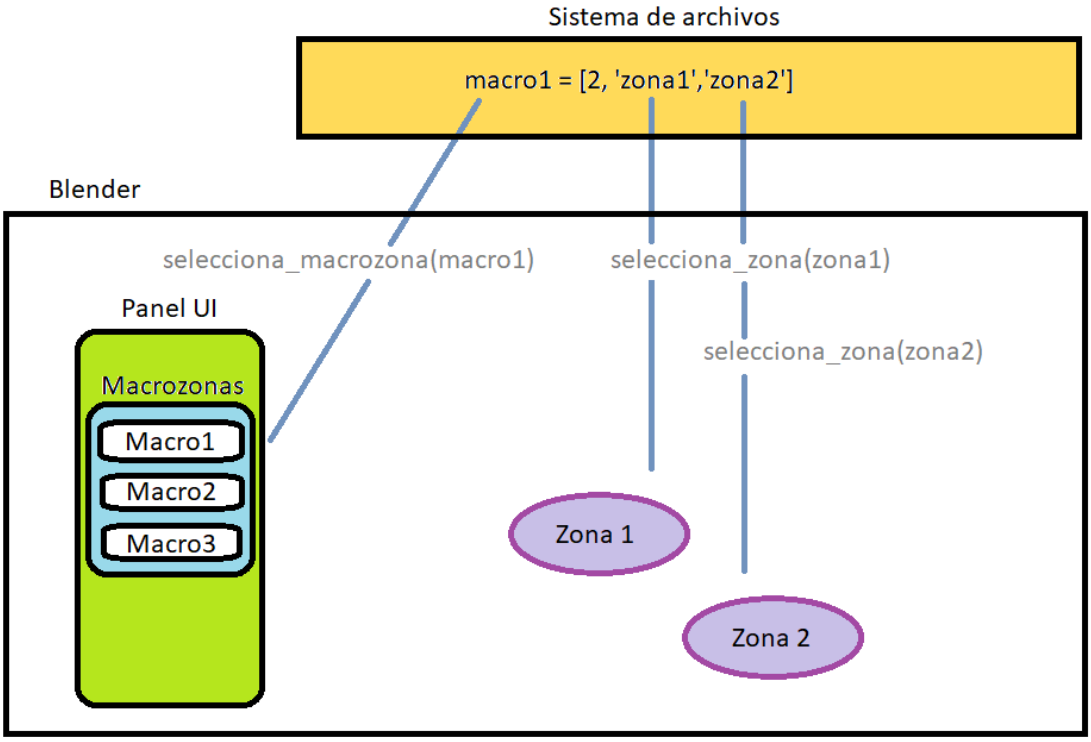

Figura 20: funcionamiento del operador selecciona_macrozona

Una macrozona es en esencia una lista de punteros: cuando el usuario la selecciona se despliega el operador *selecciona_macrozona*, que la busca en el sistema de archivos y obtiene los nombres

de las zonas faciales que están asociadas. Este operador invoca al operador selecciona_zona para cada una, que selecciona el grupo de vértices correspondiente.

Conociendo a nivel teórico en qué consisten las macrozonas y habiendo preparado a nivel de código la infraestructura necesaria para soportarla, se procedió a producir los scripts que darían soporte a este concepto.

Una necesidad que surgió justo en esa etapa era la de disponer de un punto de anclaje, una información accesible dentro de Blender que permitiese saber al programa en qué perfil de edición se estaba trabajando actualmente, no solo para evitar preguntar al usuario en cada operador en que perfil se está trabajando (así era en una versión previa del proyecto), sino también para que la interfaz de usuario pudiese mostrar las distintas zonas y macrozonas del perfil actual, sin mostrar las de otros perfiles.

Para ello se creó la variable global *perfil_elegido*, almacenada en el propio contexto de Blender; a continuación (**figura 21**) se muestra el código encargado de definirla.

```
import bpy


class ajustes_perfil(bpy.types.PropertyGroup):
    perfil_elegido: bpy.props.StringProperty(name = 'Perfil_seleccionado', default = '')


bpy.utils.register_class(ajustes_perfil)

bpy.types.Object.pep = bpy.props.PointerProperty(type=ajustes_perfil)
```

Figura 21: clase ajustes_perfil, con el código necesario para desplegarlo.

Como podemos ver en la figura, declaramos la clase *ajustes_perfil*, a la que le pertenece la propiedad de tipo *string perfil_elegido*. Seguidamente se registra la clase y se asigna a la extensión *bpy.types.Object.pep*. Ahora ya existe de manera global en el ámbito global de Blender, y se puede acceder a ella con comandos muy simples como:

bpy.context.active_object.pep.perfil_elegido = "**Perfil 1**" → asigna a la propiedad la cadena de caracteres "**Perfil 1**", se emplea en el operador *cargar_perfil*.

Perfil = bpy.context.active_object.pep.perfil_elegido → obtiene la cadena de caracteres del perfil elegido y la asigna a la variable **Perfil**.

Estos comandos permiten interactuar a los demás operadores con la variable.

Con este obstáculo resuelto se procedió a crear los scripts necesarios para crear macrozonas y añadir o eliminar zonas de las mismas.

Transformaciones.

Una vez se ha logrado crear macrozonas e incluir o eliminar zonas asociadas se ha procedido a incorporar el código que dará soporte a la implementación de las transformaciones. Para ello se ha trabajado en los campos de la macrozona y se ha incorporado la posibilidad de añadir transformaciones.

Este apartado no se ha elaborado a nivel de código pero sí que se ha elaborado un concepto teórico de cómo se podrían implementar. Sin embargo, faltan por concretarse los elementos del código que respalden ese concepto de manera adecuada.

Por ahora, la idea sería crear las siguientes funciones:

anadir_transformacion: registra una transformación en la mesh y la almacena en un fichero con el nombre propio que se le quiera dar, además de añadir el nombre de la transformación a la macrozona asociada.

eliminar_transformacion: elimina una transformación de una macrozona.

aplicar_transformación: al seleccionar el botón de la interfaz de usuario que representa la transformación se invoca este operador, que aplica la transformación a la mesh.

Estas herramientas se añadirán en futuras versiones del trabajo.

Capítulo 4 Manual de usuario

En este capítulo daré indicaciones para el despliegue de Blender y de la aplicación, unos controles básicos en el entorno Blender y la descripción de las distintas herramientas y una demostración de su funcionamiento. Por último, daré recomendaciones de tutoriales online de Blender, fáciles de seguir y muy efectivos para coger las bases.

Instalación de Blender

En primer lugar, hay que instalar Blender en el ordenador: para ello debemos ir a la página oficial de Blender y pulsar el botón "*Download Blender <**version**>"* siendo **versión** la versión más actual del programa.

Una vez descargado el instalador, se instala como cualquier otro programa.

Instalación de la aplicación.

Hay que descargar el archivo comprimido, y extraerlo en la ubicación que se desee.

Una vez hecho eso, hay que añadir el archivo *blend* que contiene el modelo tridimensional que nos interesa editar en la carpeta descomprimida, de manera que esté ubicado en el mismo lugar que los scripts y la carpeta *Presets*.

Ahora hay que abrir Blender y abrir ese archivo (File → Open; también se puede haciendo doble clic sobre el archivo *blend* en la carpeta del proyecto). Una vez abierto, nos desplazamos al modo scripting (**figura 22**) y pulsamos el botón **Open** del panel derecho (**figura 23**), seleccionamos el archivo **main.py** (o **testv3.py** en la versión actual) y le damos al botón de ejecutar ( ) (**Figura 24**). Una vez pulsado ya se ha instalado en el archivo actual, habría que repetir este último párrafo por cada archivo nuevo *blend*.

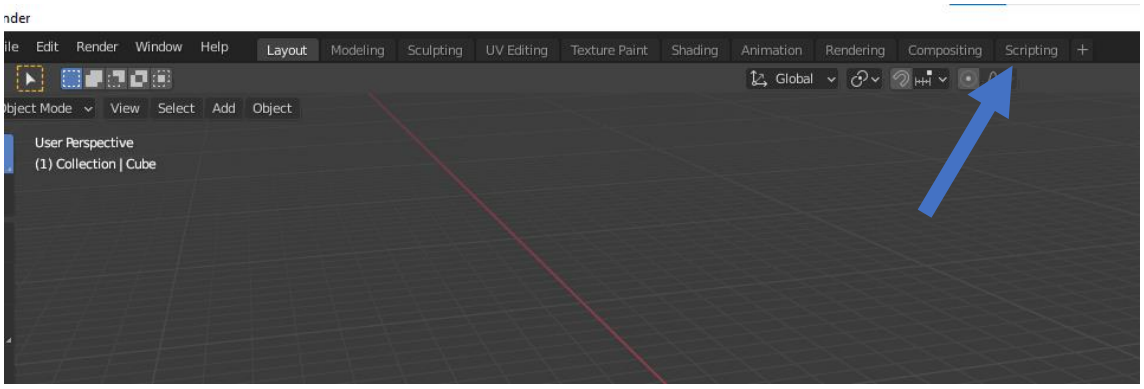

Figura 22) los distintos modos de Blender.

La flecha azul indica el modo scripting

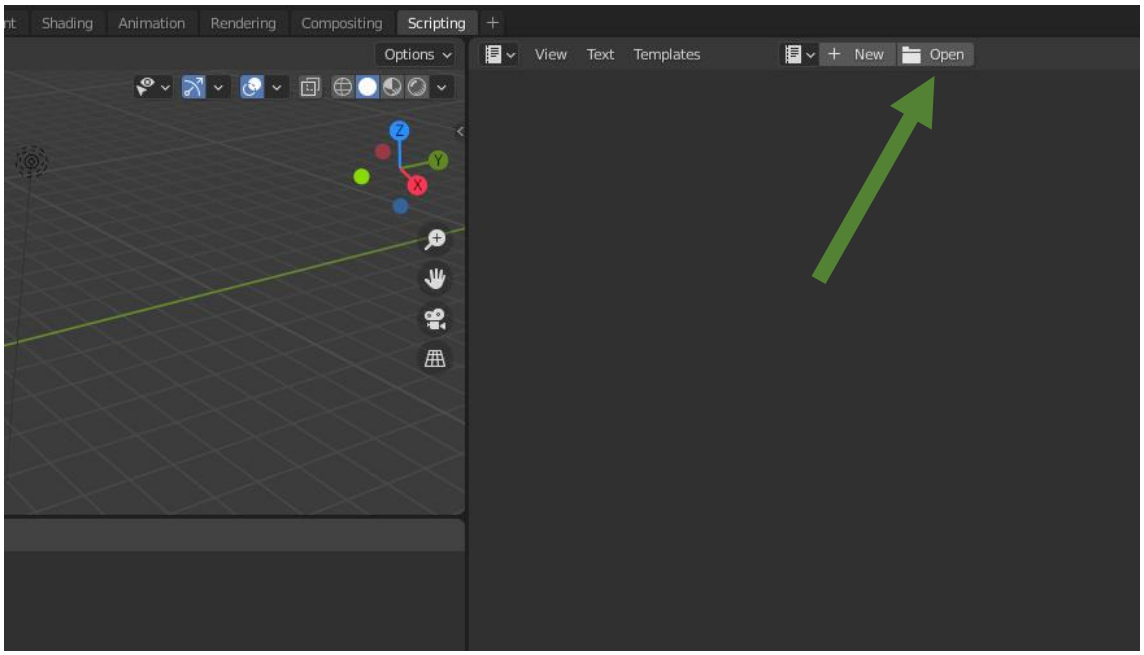

Figura 23) Panel de edición de texto del modo scripting

La flecha verde indica el botón Open.

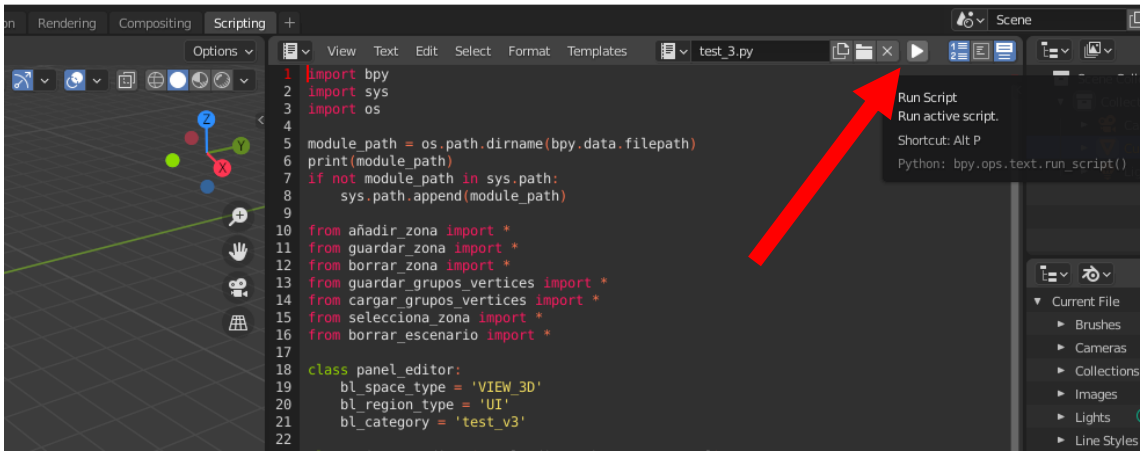

Figura 24) archivo main.py, en el editor de texto del modo scripting.

La flecha roja indica el botón de ejecutar.

Controles básicos de Blender: Modo layout o modo objeto.

Una vez instalado Blender, se ha de comprender como moverse aún de manera básica por el mismo. En primer lugar, se muestra en la imagen a continuación (**figura 25**) el entorno que se despliega al abrir Blender.

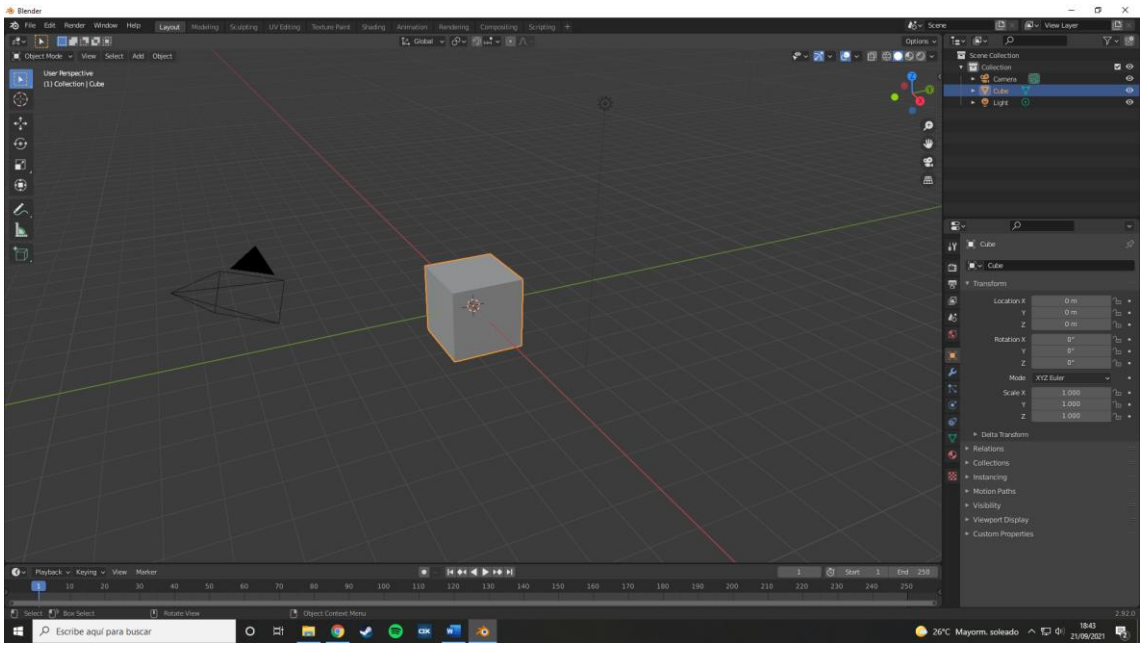

Figura 25: el modo layout es el modo por defecto

Aquí puede verse el escenario tridimensional, donde el modelo 3D está ubicado en el centro.

Arriba a la izquierda se encuentran las distintas ventanas: *File, Edit, Render, Window, Help*, aunque de cara a este proyecto solo usaremos las ventanas *File, Edit y Window*.

Los modos de Blender pueden cambiarse en las ventanas de arriba (entre ellos están los modos Layout, Modeling y Scripting, que son los más relevantes).

A la derecha están las ventanas de la colección de la escena, donde se pueden visualizar los distintos elementos presentes en esta. Aunque también muestra los grupos de vértices del modelo con los distintos objetos 3D de la escena, no nos es de particular interés en este proyecto, ya que hay mejores maneras de visualizarlos, pero en el futuro podría ser de utilidad.

La ventana inmediatamente inferior contiene varias herramientas, que sirven para varias tareas. Entre ellas está la herramienta que permite ver los distintos grupos de vértices presentes en la escena, asociados a un modelo concreto. La **figura 26** muestra dicha herramienta.

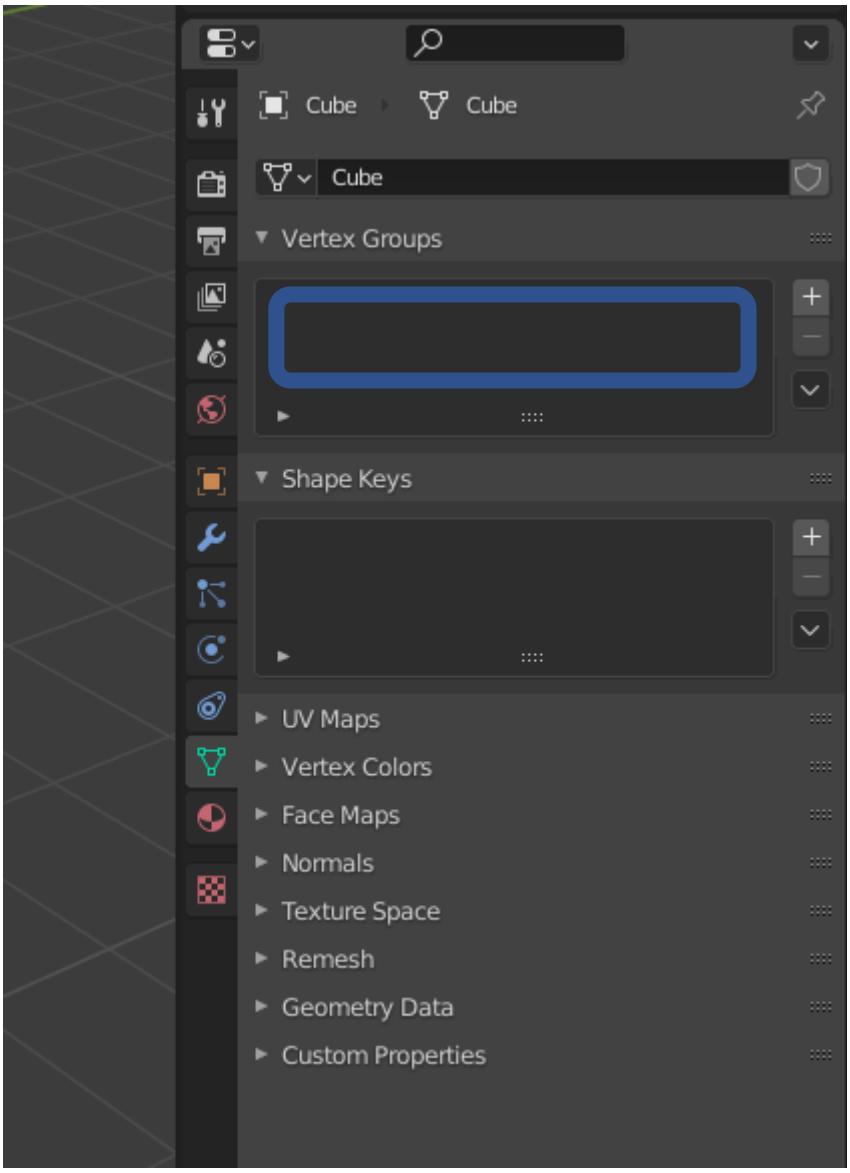

Figura 26: el panel de herramientas y la herramienta de grupos de vértices

En el rectángulo azul se encontrarían los distintos grupos de vértices del modelo.

Conocidos los distintos elementos del modo layout, que es donde nos encontramos, procedemos a movernos por el escenario 3D. Los controles básicos son los siguientes:

Botón izquierdo del ratón: permite seleccionar a los distintos elementos del escenario, al seleccionar un modelo, su silueta se ilumina de color amarillo (**figura 27**)

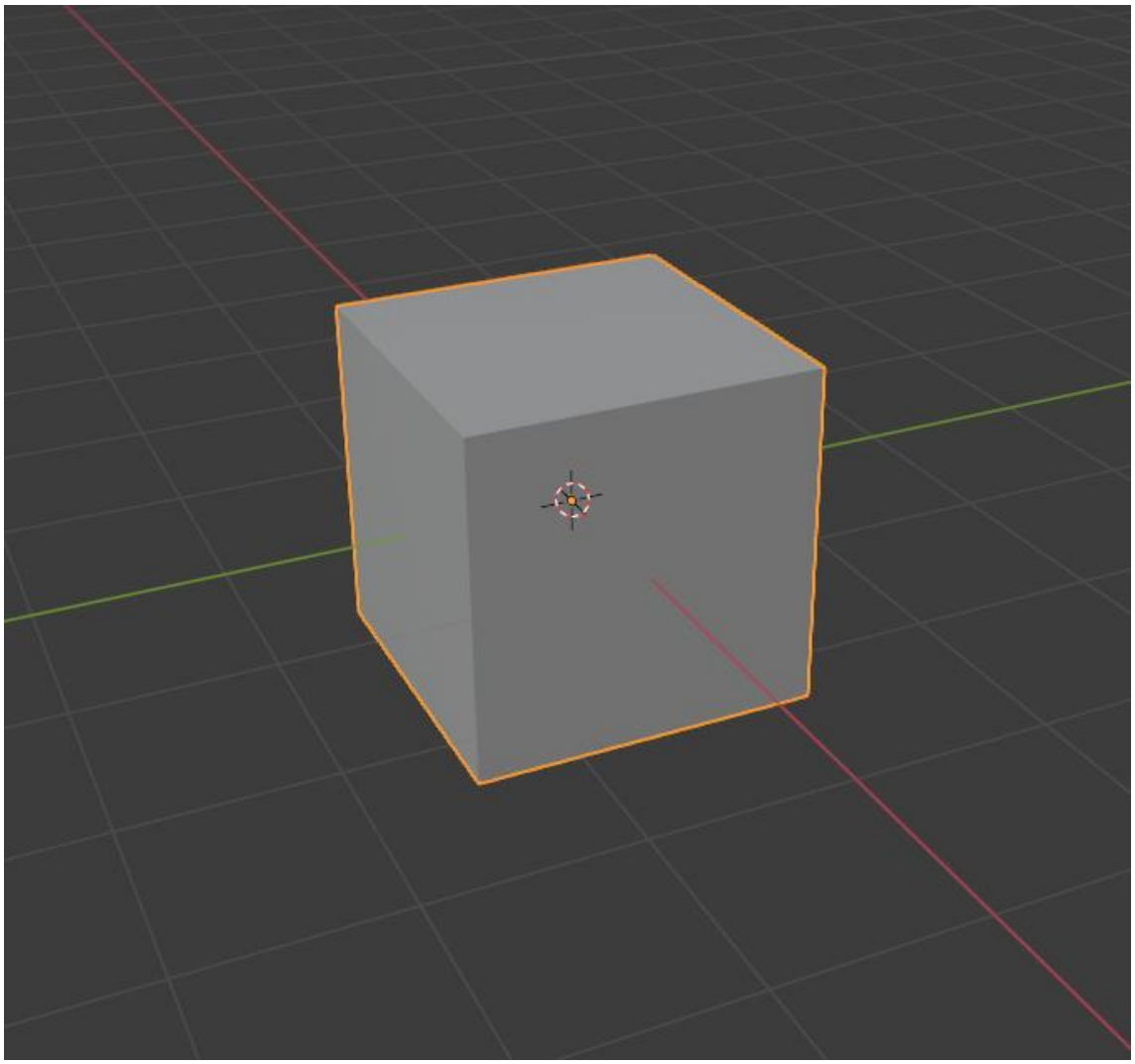

Figura 27: objeto seleccionado

En el centro de la imagen se puede ver el centro punto del origen del objeto, que actúa de referencia para la cámara que visualiza el objeto y para ciertas alteraciones, no se ha hecho uso de el en la aplicación.

Botón derecho del ratón: menú contextual, sus funciones cambian según donde se pulse, no se utiliza en esta aplicación por ahora.

Rueda del ratón: permite alejar y acercar la cámara del escenario tridimensional, que hace las veces de zoom.

Mouse 3 (pulsar el botón de la rueda): permite rotar la cámara en torno al punto de origen de la escena, sirve para ver desde distintos puntos de vista al objeto.

Shift + mouse3: permite mover la cámara en un plano separado del objeto, útil para mover la cámara en los distintos ejes espaciales

Tecla G (en objeto seleccionado): permite trasladar el objeto en un plano paralelo al de la visión de la cámara, si se pulsa además **mouse3** permite escoger el eje espacial concreto en el que trasladarlo.

Tecla S (en objeto seleccionado): permite escalar el objeto (aumentar o disminuir su tamaño), si se pulsa además **mouse3** permite escoger el eje espacial concreto en el que escalarlo.

Tecla R (en objeto seleccionado): permite rotar el objeto, si se pulsa además **mouse3** permite escoger el eje espacial concreto en el que rotarlo.

Controles básicos de Blender: Modo Modeling o modo edición.

En primer lugar, para seleccionar este modo debe seleccionarse la ventana Modeling, situada arriba, a la derecha de la ventana Layout.

Una vez seleccionada, entramos en el modo edición, en el que se puede visualizar la mesh del modelo 3D en cuestión.

Los controles escritos en el apartado anterior se mantienen idénticos, salvo que en este modo el clic izquierdo selecciona los distintos vértices de la mesh y las alteraciones se hacen en los vértices.

Botón izquierdo del ratón: selecciona un vértice de la mesh. Si se arrastra manteniendo pulsado se seleccionan todos los vértices contenidos en el área contenida. Los vértices seleccionados se ven de color amarillo.

Botón izquierdo del ratón + Shift: permite seleccionar más de un vértice.

Botón izquierdo del ratón + Control: permite seleccionar más de un vértice, incluyendo los vértices presentes en la ruta más corta entre dos vértices seleccionados.

Tecla C: Cambia el modo de selección circular, que actúa como una brocha mediante la cual si se mantiene pulsado el botón izquierdo se seleccionan todos los vértices por los que pasa el cursor, si se mantiene el botón central del ratón se deseleccionan. Con la rueda del ratón puede ampliarse o disminuirse el radio del círculo de selección.

Tecla O: habilita o deshabilita la opción de edición proporcional en el escenario: en este modo la alteración de la *mesh* no da lugar a cambios abruptos en la forma del modelo. Mas adelante se ve un ejemplo práctico de este modo.

Herramientas de la aplicación:

En esta sección se describe la funcionalidad de las distintas herramientas de la aplicación que ha sido desarrolladas, como se pudo ver en la **figura 6**, pero que recuperamos aquí por conveniencia.

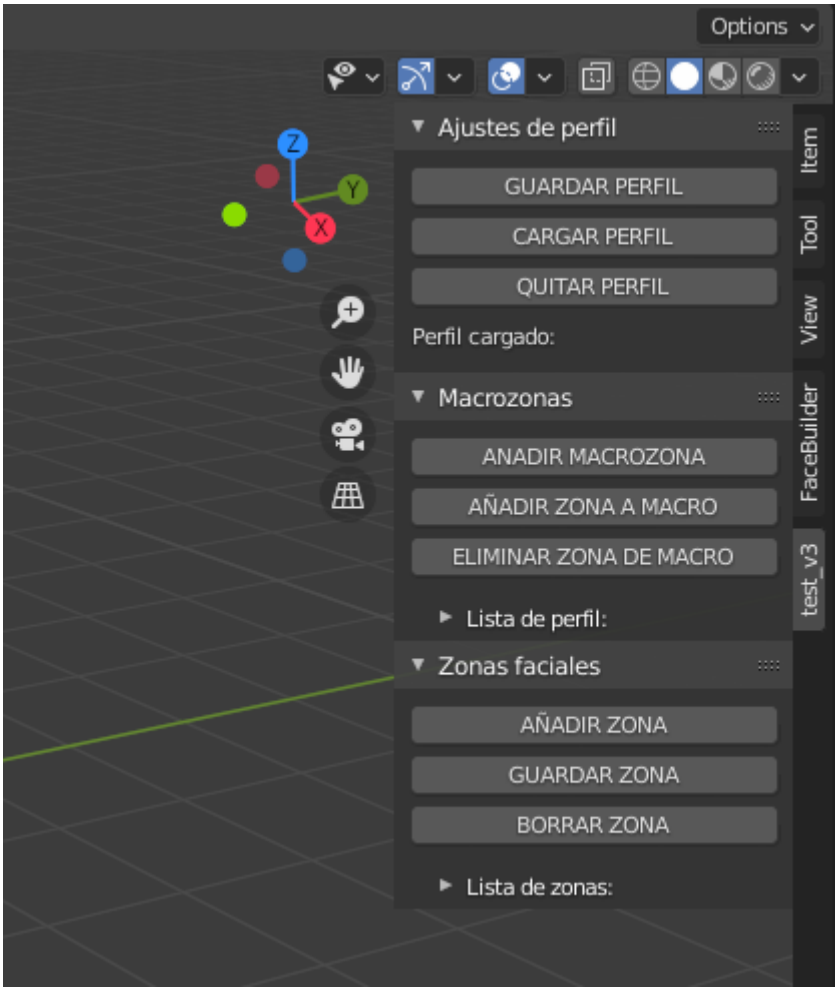

Figura 6.

En el primer panel se encuentran los ajustes del perfil, con los botones GUARDAR PERFIL, CARGAR PERFIL y BORRAR PERFIL, cuyas funciones son las siguientes:

- GUARDAR PERFIL: en primer lugar, aparece una ventana emergente que pregunta al usuario en que perfil se desea guardar la información. Una vez escrito, se guarda la información contenida en los grupos de vértices presentes en la escena de Blender en las distintas zonas faciales, que se almacenan en la carpeta Zonas del perfil indicado. En caso de existir el perfil indicado, se sobreescribirán las zonas existentes y se añadirán las zonas nuevas. También cambia el valor de la variable perfil_elegido al perfil escogido.
- CARGAR PERFIL: permite seleccionar el perfil de edición y carga las zonas faciales y crea los grupos de vértices que les corresponden. Cambia el valor de la variable perfil_elegido al perfil seleccionado. Es muy importante que solo se seleccione la carpeta del perfil cuando aparezca el dialogo de selección de directorio, y una vez dentro no se seleccione ninguna de las carpetas contenidas (Zonas, macrozonas o transformaciones), de hacerlo no funcionará.

- BORRAR PERFIL: elimina de la escena de Blender los distintos grupos de vértices y borra el valor de la variable perfil_elegido, lo que reinicia el escenario.

Debajo de estas opciones se encuentra la etiqueta Perfil cargado, que permite visualizar el valor de la variable perfil_elegido y, por lo tanto, el perfil en el que se está editando actualmente.

En la **figura 28** se puede ver la interfaz de usuario habiendo seleccionado un perfil, con todos los sub-paneles desplegados.

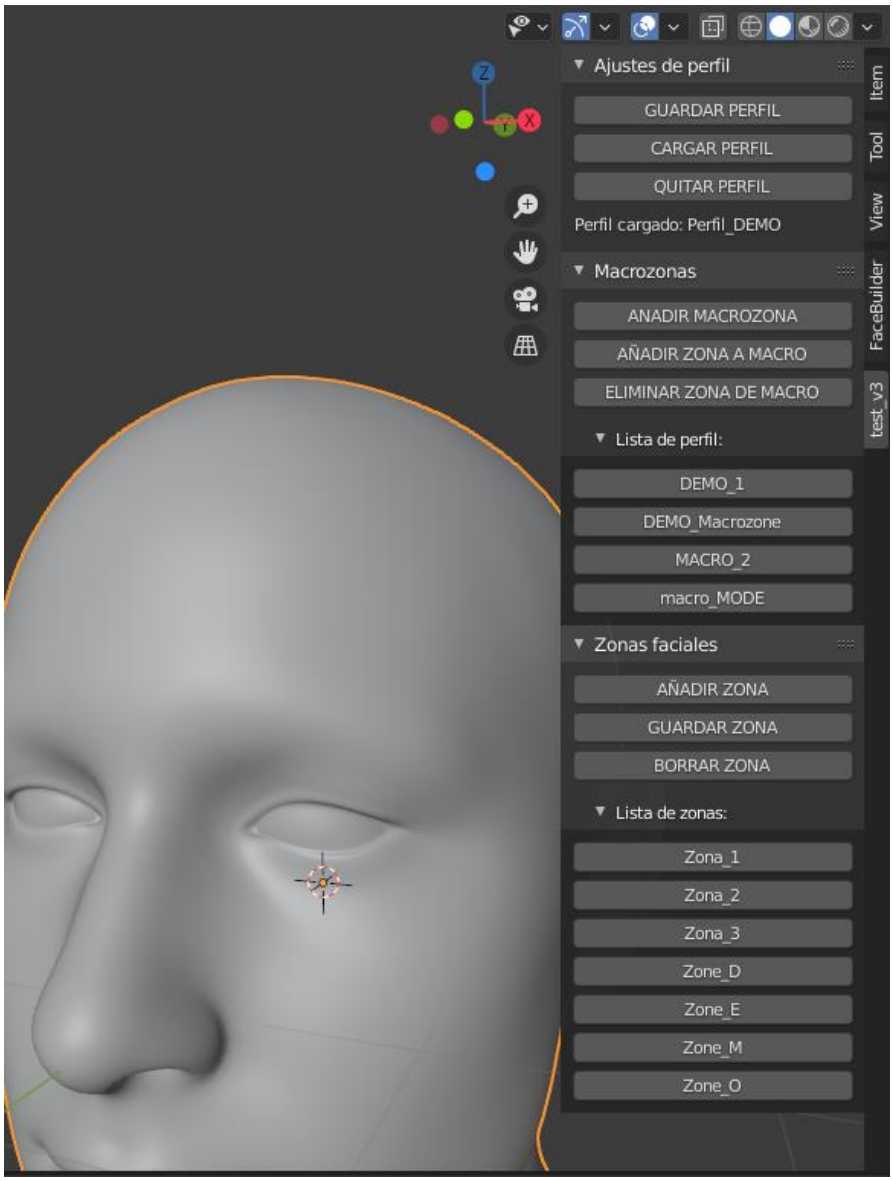

Figura 28: Panel de usuario, en el que se ha seleccionado el perfil *Perfil_DEMO*.

El segundo panel muestra las opciones de las macrozonas y una lista desplegable que contiene las distintas macrozonas.

- AÑADIR MACROZONA: permite añadir una macrozona al perfil seleccionado.

- AÑADIR ZONA A MACROZONA: teniendo una zona seleccionada, permite añadirla a la macrozona indicada en el diálogo emergente.
- ELIMINAR ZONA DE MACROZONA: elimina la zona indicada en el diálogo de la macrozona indicada.

Debajo de estas opciones se encuentra una lista desplegable con las distintas macrozonas del perfil, si seleccionamos una se despliega a su vez un menú con las distintas opciones disponibles para ella, incluidas las transformaciones (**figura 29**).

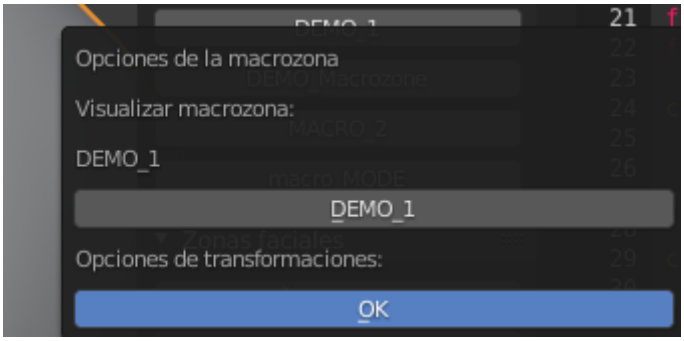

Figura 29: menú de la macrozona DEMO_1

El botón DEMO_1 permite seleccionar la macrozona (**figura 30**), las opciones de las transformaciones están aún en desarrollo.

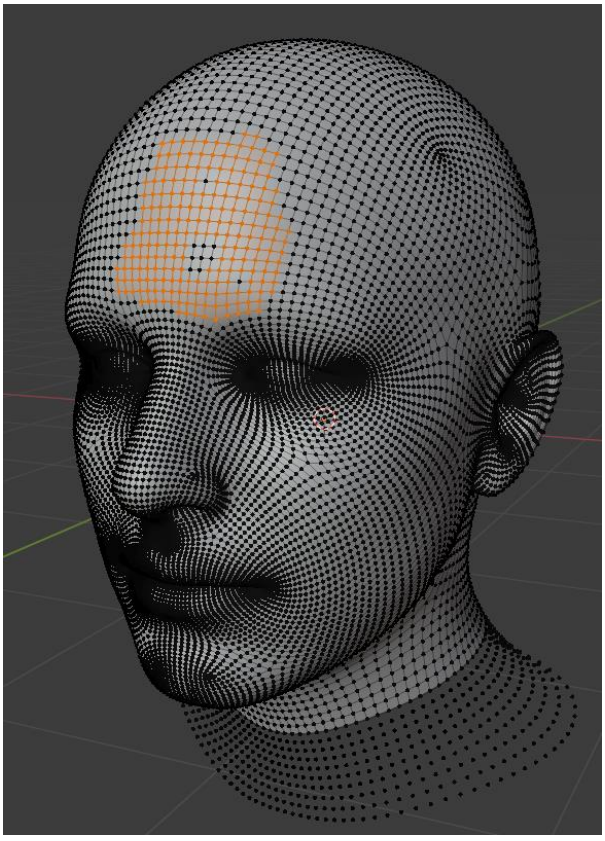

Figura 30: macrozona DEMO_1 seleccionada.

El panel de zonas faciales muestra las opciones de cara a la creación de zonas faciales: recordamos en este apartado que las zonas faciales es otra manera de llamar a los grupos de vértices a nivel teórico, ya que es menos confuso para el usuario final y son fundamentalmente lo mismo en el plano visual. Las funciones AÑADIR ZONA, GUARDAR ZONA y BORRAR ZONA hacen alusión a los grupos de vértices, que serán guardados como autenticas zonas faciales en la función GUARDAR PERFIL.

- AÑADIR ZONA: cambia a modo edición y muestra un tutorial básico para seleccionar zonas.
- GUARDAR ZONA: crea un grupo de vértices con los vértices seleccionados, de nombre el especificado por el usuario.
- BORRAR ZONA: borra el grupo de vértices indicado del escenario de Blender, pero no borra la zona facial asociada del sistema de archivos.

Debajo de estas opciones se encuentra la lista desplegable de las zonas presentes, al seleccionar cualquiera de ellas se selecciona de manera análoga a la macrozona de la **figura 30**. En este caso se muestra en la **figura 31**.

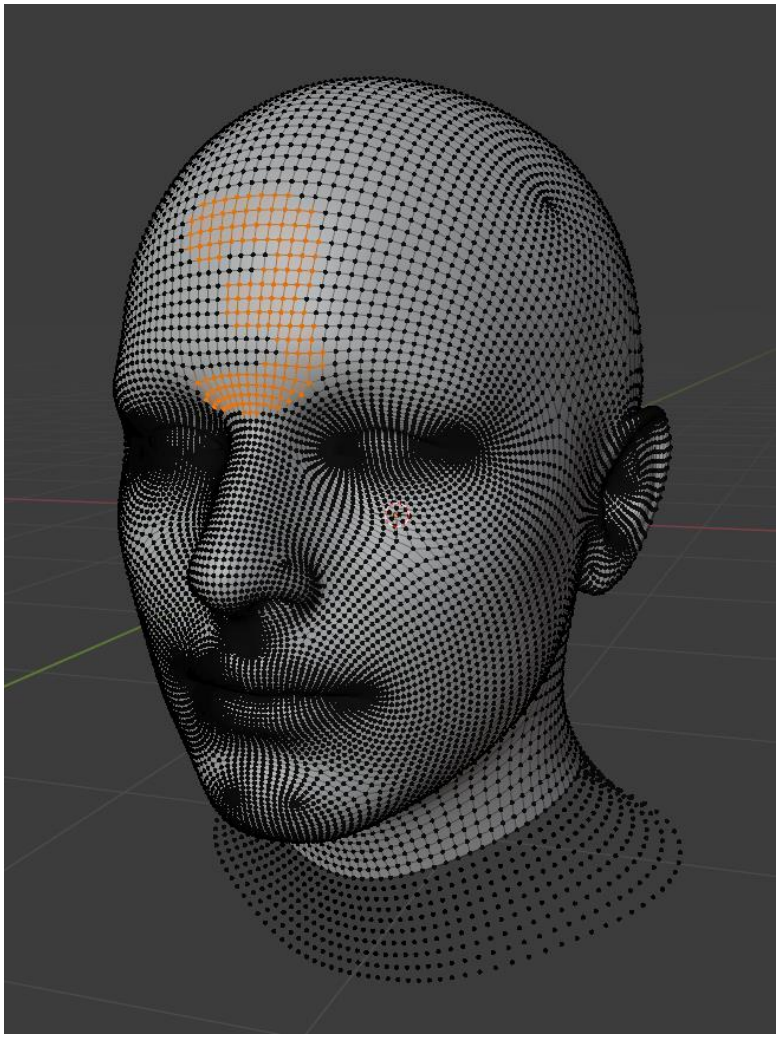

Figura 31: zona "Zona_3" seleccionada.

Al seleccionar una zona, si se pulsan las teclas G, S o R se traslada, escala o rota la zona (grupo de vértices) en cuestión. Si se pulsa previamente la tecla O, se hará una alteración proporcional, que evita cambios abruptos en la *mesh* (**figura 32**).

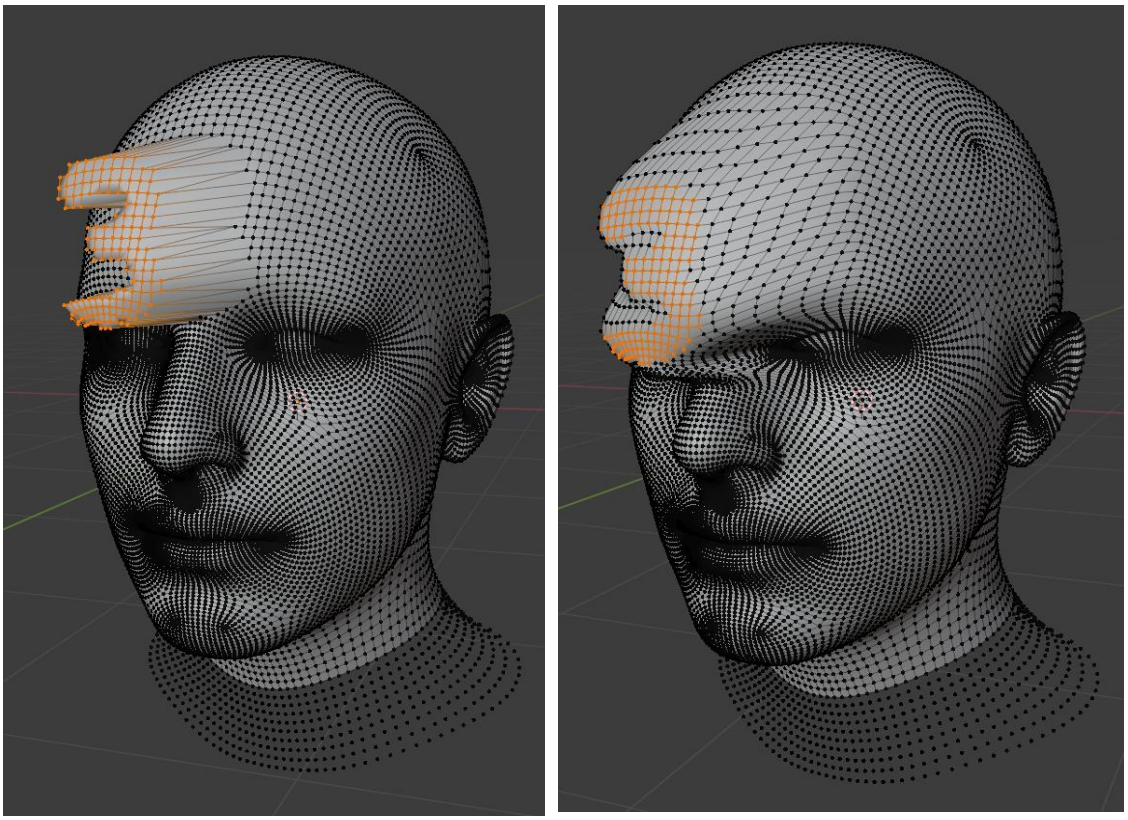

Figura 32: Comparativa de la alteración proporcional: izquierda: desactivada, derecha: activada

Diseñador: procedimiento de creación de perfil de edición

Al desplegar por primera vez la aplicación es probable que no se cuente con un entorno de edición preestablecido para el objeto que tenemos en escena, por lo que procederá de la siguiente manera:

- Paso 1) Se añaden las distintas zonas: esto puede hacerse paso a paso, focalizándose de parte en parte de la cara y guardando las zonas paulatinamente, para ello se usan las funciones AÑADIR ZONA, GUARDAR ZONA Y GUARDAR PERFIL.
- Paso 2) Crear las macrozonas: se crean las distintas macrozonas, formadas por las zonas faciales creadas anteriormente. Para este paso se utilizan las funciones AÑADIR MACROZONA, AÑADIR ZONA A MACROZONA.
- Paso 3) Crear las transformaciones, falta por integrar.

Este procedimiento es sencillo de seguir, pero es muy importante guardar el perfil antes de salir, ya que los grupos de vértices no se pueden exportar como zonas si no se han guardado y tanto las macrozonas como las macrozonas dependen de ellos, haciendo el editor inservible en un escenario nuevo.

Capítulo 5 Conclusiones y pasos futuros.

A raíz de la investigación realizada en el entorno Blender y de las herramientas desarrolladas, se concluye que la elaboración de una aplicación de retrato robot tridimensional en Blender es perfectamente factible: ha sido posible elaborar una infraestructura que permite crear a un editor de personajes personalizado al gusto de cada uno, que en pasos futuros permitirá alterar la geometría de cualquier objeto de manera fácil y eficiente. Además, esta aplicación es transparente al entorno mismo, lo que significa que se podrá beneficiar de todas las herramientas a disposición de Blender y programas de terceros como FaceBuilder. De los objetivos establecidos se han podido completar todos, a excepción del código para implementar las transformaciones, pero si que se ha podido crear la infraestructura para soportarlas.

Animación facial, simulación de una situación con fuentes de luz variables, todo lo que ofrece actualmente Blender podrá aplicarse en los modelos editados sin problemas de compatibilidad.

Los pasos futuros más inmediatos serían el desarrollo del código para registrar y aplicar transformaciones y el código que permita alterar la piel del modelo facial a editar para cambiar la tonalidad de los ojos y demás rasgos faciales. Otros pasos futuros muy interesantes serían la migración del sistema de archivos local a una base de datos, un sistema de animación facial que emule gestos o pronunciamiento de palabras, un sistema de iluminación realista que ayude a establecer la escena donde ocurrió el avistamiento. Una meta algo más relacionada con las telecomunicaciones podría ser la elaboración de una aplicación móvil que permita hacer uso de los editores mediante conexión *ssh* a un ordenador, que podría compatibilizarse muy bien con la aplicación *FaceBuilder*.

Trabajar en este proyecto ha sido una experiencia muy gratificante, no solo gracias al conocimiento adquirido en un entorno completamente desconocido a ojos del grado estudiado como es Blender y a la mejora experimentada en Python, si no también a la experiencia desarrollada en tareas como la elaboración de la arquitectura de un programa, el planteamiento de metas y su ejecución de acuerdo con las posibilidades que ofrece el entorno. También se ha hecho hincapié en desarrollar un código eficiente, siguiendo buenas prácticas de programación, para facilitar el desarrollo posterior a este proyecto.

Espero poder seguir trabajando con este proyecto en el futuro.

Capítulo 6 Referencias y bibliografía.

Referencias.

[1] P. Martínek and I. Kolingerová, "Deformation method for 3D identikit creation," *2014 International Conference on Computer Graphics Theory and Applications (GRAPP)*, 2014, pp. 1-8.

[2] Multiple authors (no date available) "What is a Mesh?" *Blender 3D: Noob to Pro* [Online]. Available: https://en.wikibooks.org/wiki/Blender_3D:_Noob_to_Pro/What_is_a_Mesh%3F#:~:text=A%20mesh%20is%20a%20collection,flat%20surface%20enclosed%20by%20edges

[3] Blender official channel on Youtube (2020, 04, 03) "*Your Own Operator*" [Online]. Available: https://www.youtube.com/watch?v=xscQ9tcN4GI&list=PL1fkRtMmJ4OOrY20bOVlxn_PFYx9ly97j&index=6.

[4] Blender official site (no date available) "Operator(bpy_struct)" [Online]. Available: https://docs.blender.org/api/current/bpy.types.Operator.html.

[5] Blender official site (no date available) "*Gotchas: Help! My script crashes Blender*" [Online]. Available: https://docs.blender.org/api/current/info_gotcha.html#help-my-script-crashes-blender

Bibliografía

[] Blender Stack Exchange user "stacker" (2015, 01, 13) "*Select vertices of mesh in python*" [Online]. Available: https://blender.stackexchange.com/questions/23113/select-vertices-of-mesh-in-python

[] Blender Stack Exchange community, especifically user "TLousky" (2017, 03, 9) "*Finding vertices in a Vertex Group using Blender's Python API*" [Online]. Available: https://blender.stackexchange.com/questions/75223/finding-vertices-in-a-vertex-group-using-blenders-python-api

[] B3D Interplanety user "Nikita" (2020, 02, 12) "*How to create a Vertex Group and add vertices to it with the Blender Python API*" [Online]. Available: https://b3d.interplanety.org/en/how-to-create-a-vertex-group-and-add-vertices-to-it-with-the-blender-python-api/

[] Erik Selin (no date) "*How to use vertex groups in Blender*" [Online]. Available: https://artisticrender.com/how-to-use-vertex-groups-in-blender/

[] Blender official site (no date available) "*Vertex Weights - Locking*" [Online]. Available: https://docs.blender.org/manual/en/latest/modeling/meshes/properties/vertex_groups/vertex_weights.html#locking

[] Blender Stack Exchange user "Gwen" (2013, 06, 27) "*Efficient way to get selected vertices via python (without iterating over the entire mesh)*" [Online]. Available: https://blender.stackexchange.com/questions/1412/efficient-way-to-get-selected-vertices-via-python-without-iterating-over-the-en

[] Blender Scripting User "zeffii" (2012, 12, 26) "*getting index of currently selected vertex (or vertices)*" [Online]. Available: https://blenderscripting.blogspot.com/2011/07/getting-index-of-currently-selected.html

[] programiz.com post, no autor visible (no date) "*Python List Comprehension*" [Online]. Available: https://www.programiz.com/python-programming/list-comprehension

[] Blender Artists user "batFINGER" (2012, 02, 23) "*Deleting vertex groups*" [Online]. Available: https://blenderartists.org/t/deleting-vertex-groups/533627

[] Blender Stack Exchange user “Skylumz” (2018, 06, 5) “*How to popup simple message box from python console?*” [Online]. Available: https://blender.stackexchange.com/questions/109711/how-to-popup-simple-message-box-from-python-console

[] Blender Stack Exchange user “CodeManX” (2015, 01, 9) “*How to export list of prop values as txt to a certain folder? And how to import them back? [closed]*” [Online]. Available: https://blender.stackexchange.com/questions/22921/how-to-export-list-of-prop-values-as-txt-to-a-certain-folder-and-how-to-import

[] kite.com post, no autor visible (no date) “*How to read a text file into a list in Python*” [Online]. Available: https://www.kite.com/python/answers/how-to-read-a-text-file-into-a-list-in-python

[] pythontutorial.net post, no autor visible (no date) “*Python Directory*” [Online]. Available: https://www.pythontutorial.net/python-basics/python-directory/

[] Jacob Valenta (2011, 09, 26) “*File Selection With Python*” [Online]. Available: https://blenderapi.wordpress.com/2011/09/26/file-selection-with-python/

[] Blender Stack Exchange user “fnunnari” (2018, 12, 19) “*Use filemanager to select directory instead of file?*” [Online]. Available: https://blender.stackexchange.com/questions/14738/use-filemanager-to-select-directory-instead-of-file

[] devtalk.blender user “RainerTrummer” (2019, 01, 17) “*Store variable within blender scene?*” [Online]. Available: https://devtalk.blender.org/t/store-variable-within-blender-scene/5070/2

[] Blender Stack Exchange user “Robert Gützkow♦” (2019, 10, 15) “*How do a create a foldout UI panel?*” [Online]. Available: https://blender.stackexchange.com/questions/155515/how-do-a-create-a-foldout-ui-panel

[] Blender Artists user “testure” (2021, 05, 24) “*Storing variables / Classes / property groups in objects / scene / collection*” [Online]. Available: https://blenderartists.org/t/storing-variables-classes-property-groups-in-objects-scene-collection/1306172

Universidad de Alcalá

Escuela Politécnica Superior

ESCUELA POLITECNICA SUPERIOR