Universidad de Alcalá
Escuela Politécnica Superior

GRADO EN INGENIERÍA DE COMPUTADORES

Trabajo Fin de Grado

Contribuciones al desarrollo de una aplicación de retrato robot sobre Blender

Autor: Enrique Díaz-Romeral Marcos

Tutor/es: Bernardo Alarcos Alcázar

2022

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

GRADO EN INGENIERÍA DE COMPUTADORES

Trabajo Fin de Grado

Contribuciones al desarrollo de una aplicación de retrato robot sobre Blender

Autor: Enrique Díaz-Romeral Marcos

Tutor/es: D. Bernardo Alarcos Alcázar

TRIBUNAL:

Presidente: D. Iván Marsá Maestre

Vocal 1º: D. Luis de la Cruz Piris

Vocal 2º: D. Bernardo Alarcos Alcázar

FECHA: 10/09/2022

ÍNDICE

Resumen 3
Summary 4
Resumen extendido 5
Palabras clave 7
Key words 8
Glosario de atajos de teclado 11
1. Introducción 13
1.1 Definición y evolución del retrato robot. 13
1.2 Objetivos 16
1.3 Estructura de la memoria 19
2. Fundamentos sobre el trabajo de fin de grado 18
2.1 Proyecto anterior 18
2.2 Preámbulo de la aplicación Blender. 21
2.2.1 ¿Qué es Blender? 21
2.2.2 Introducción a Blender: Workspace 22
2.2.3 Introducción a Blender: Malla (mesh). 24
2.2.4 Introducción a Blender: Tipos de vista. 25
2.2.5 Introducción a Blender: Modos de interacción con el objeto 27
3. Contribuciones al desarrollo de una aplicación de retrato robot sobre Blender 30
3.1. Análisis general del ojo humano 31
3.2. Creación de la malla de un ojo: Superficie ocular 32
3.3 Creación de la malla de un ojo: Iris 36
3.4 Creación de los materiales. 39
3.5 Creación material: Superficie del ojo 42
3.6 Creación material: Iris. 54
4. Resultado de aplicar los materiales de “Superficie Ojo” y de “Iris”. 58
5. Scripting material para los ojos 58
5.1 Determinar los índices de los vértices de la mesh. 58
5.2 Seleccionar los índices de los materiales “Superficie del ojo” y del “Iris” a través de Python. 62
5.3 Creación de los materiales “Superficie del ojo” e “Iris” y asignar materiales a los vértices estudiados en los puntos anteriores. 65
5.3.1 Tipos de nodos utilizados para crear los materiales 67
5.3.2 Explicación de la implementación de los enlaces entre los nodos en código 85
5.3.3 Asignación de los materiales a la mesh. 88
6. Ampliación de script (Parte I) : “Creación_Materiales_Ojo.py” 90
7. Ampliación de script (Parte II) : “Aplicar_Materiales.py” 95
8. Ejecutar scripts de Python en Blender. 102
9. Enlazando las mallas creadas con herramienta del TFG anterior. 104

10. Problemas encontrados 105
10.1 Selección mal de vértices al guardar perfil y cargarlo. 105
10.2 Conocimiento de los índices que identifican a los vértices. 108
10.3 Variable global en script de Python de interfaz de usuario para cambiar material. 110
11. Scripts involucrados en el proyecto. 111
11.1 “Imprimir_vertices_seleccionados.py” 111
11.2 “Seleccionar_vertices_ojo.py” 113
11.3 “Creacion_Materiales_Ojo.py” 116
12. Conclusión y futuros trabajos. 142
13. Bibliografía 144

FIGURAS

Fig. 1. Ejemplo de boceto 13
Fig. 2. Diagrama del proceso de identificación a través del modo aprendizaje....14
Fig. 3. Diagrama del proceso de identificación a través del modo de pruebas 14
Fig. 4. Comportamiento de herramienta al seleccionar macrozona....19
Fig. 5. Macrozona del ojo con sus respectivas zonas asociadas 20
Fig. 6. UI Panel creado por la herramienta del anterior alumno....21
Fig. 7. Fotograma de la película Robot 7723 creada con Blender....22
Fig. 8. Pantallazo de los espacios de trabajo y tipos de editores que vienen por defecto en Blender 22
Fig. 9. Pantallazo del espacio de trabajo Layout 23
Fig. 10. Workspace Scripting de Blender.....24
Fig. 11. Malla de un ojo formada por vértices y caras 24
Fig. 12. Acceso a distintos tipos de vista.....25
Fig. 13. Vista Wireframe....25
Fig. 14. Vista Sólido 26
Fig. 15. Vista renderizada 26
Fig. 16. Vista previa de los materiales 27
Fig. 17. Combo box de selección de modo de interacción 27
Fig. 18. Modo de objeto 28
Fig. 19. Modo de edición: vértices, aristas y caras. 28
Fig. 20. Modo de esculpir. 29
Fig. 21. Anatomía del ojo.....31
Fig. 22. Creación de malla UV Sphere y aplicación de material metal 32
Fig. 23. Vista de UV Sphere desde un polo 32
Fig. 24. Rotación para que polo quede alineado en el eje Y....33
Fig. 25. Menú de atajo para selección de cursor 33
Fig. 26. Modificación de polo de la mesh para simular superficie de córnea....34
Fig. 27. Aplicación se Shade Smooth a la malla 34
Fig. 28. Aplicación de creación de más vértices de la malla para suavizar 35
Fig. 29. Unión de facetas en la córnea 35
Fig. 30. Flor poligonal de ocho puntas....36
Fig. 31. Selección de arista y creación superpuesta de otra idéntica 36
Fig. 32. Demostración de creación de primera circunferencia interna de la “submalla”. 37
Fig. 33. Creación de las tres circunferencias internas de la “submalla” 37
Fig. 34. Unión de los vértices....38
Fig. 35. Unión de facetas....38
Fig. 36. Resultado de la malla del ojo 39
Fig. 37. Resultado de aplicar “Face Orientation” en superficie....40
Fig. 38. Resultado de aplicar “Face Orientation” en “submalla” 40
Fig. 39. Resultado de aplicar SHIFT + N en la “submalla”....41
Fig. 40. Activando Add ons 41
Fig. 41. Creación de slots de material 42
Fig. 42. Asignación de material iris a la “submalla” 43
Fig. 43. Editor propiedades de renderizado y de material 43
Fig. 44. Blend Mode a Alpha Clip 44

Fig. 45. Blend Mode a Alpha ClipFig 44
Fig. 46. Enlace entre nodos (1) 45
Fig. 47. Enlace entre nodos (2) 45
Fig. 48. ColorRamp, Mapping y Location sobre superficie ocular 46
Fig. 49. ColorRamp, Mapping y Location sobre superficie ocular (2) 46
Fig. 50. Color azul temporal en material Iris 47
Fig. 51. Enlace entre nodos de Superficie Ojo (3) 47
Fig. 52. Degradados de color en los vasos sanguíneos del ojo 48
Fig. 53. Aplicando ruido Voronoi 48
Fig. 54. Disposición de los nodos Frame y sus contenidos 49
Fig. 55. Asignando valores a Frame L 50
Fig. 56. Asignando valores a Frame M 50
Fig. 57. Asignando valores a Frame S 51
Fig. 58. Uso del nodo Mix para juntas colores de vasos sanguíneos. 51
Fig. 59. Uso del nodo Mix de los nodos Mix 52
Fig. 60. Fotografía de ojo 52
Fig. 61. Renderizado y un mapa conceptual de los nodos del material Superficie Ojo 53
Fig. 62. Editor de propiedad de materiales 54
Fig. 63. Enlace entre nodos Iris 55
Fig. 64. Añadiendo colores de color Iris con Color Ramp 55
Fig. 65. Enlace entre nodos Iris (2) 56
Fig. 66. Enlace entre nodos Iris (3) 56
Fig. 67. Enlace entre nodos Iris definitivo 57
Fig. 68. Ojo vista frontal 58
Fig. 69. Ojo vista trasera y latera 58
Fig. 70. Editor 3D Viewport 59
Fig. 71. Panel UI con botón 60
Fig. 72. Proceso para seleccionar nodos “submalla” 60
Fig. 73. Abrir consola Blender y menú surgido de la ejecución. 61
Fig. 74. Resultados índices de vértices 61
Fig. 75. Menú obtenido al ejecutar script seleccionar_vertices_ojo.py 62
Fig. 76. Demostración de la solución de presionar botón Iris 64
Fig. 77. Demostración de la solución de presionar botón Superficie Ojo 64
Fig. 78. Ejemplo gráfico de poner a True la refracción. 66
Fig. 79. Cómo se ve Frame Node una vez ejecutado el script 68
Fig. 80. Cómo se ve Mapping Node ‘Normal’ una vez ejecutado el script 69
Fig. 81. Cómo se ve Mapping Node ‘Point’ una vez ejecutado el script 70
Fig. 82. Cómo se ve Voronoi Texture Node una vez ejecutado el script 72
Fig. 83. Cómo se ve Color Ramp Node una vez ejecutado el script 75
Fig. 84. Cómo se ve Color Mix Node una vez ejecutado el script 77
Fig. 85. Cómo se ve Gradient Texture Node una vez ejecutado el script 78
Fig. 86. Cómo se ve Texture Coordinate Node una vez ejecutado el script 79
Fig. 87. Cómo se ve Noise Texture Node una vez ejecutado el script103 80
Fig. 88. Cómo se ve Bump Node una vez ejecutado el script 81
Fig. 89. Cómo se ve Material Output Node una vez ejecutado el script 82

Fig. 90. Cómo se ve Principled BSDF Node una vez ejecutado el script 84
Fig. 91. Nodo genérico para explicar enlaces 85
Fig. 92. Enlace entre dos nodos 87
Fig. 93. Comparación de iris creado a uno real. 93
Fig. 94. Materiales en editor listado. 93
Fig. 95. Editor de vista 3D tras ejecución de ampliación. 94
Fig. 96. Resultado de clicar botón de cambiar color ojos (1) 100
Fig. 97. Resultado de clicar botón de cambiar color ojos (2) 101
Fig. 98. Resultado de clicar botón de cambiar color ojos (3) 101
Fig. 99. Espacio de trabajo Scripting 102
Fig. 100. Abrir script de una ruta. 103
Fig. 101. Selección de ojo izquierda y de zona creada en dicha malla 104
Fig. 102. Selección mallas restantes y de respectivas zonas 104
Fig. 103. Creación temporal de la malla ocular 105
Fig. 104. Creación temporal de zona de malla ocular temporal 106
Fig. 105. Guardando perfil con ojo auxiliar 106
Fig. 106. Cargando perfil con ojo auxiliar 107
Fig. 107. Habilitar la vista de índices de los vértices 108
Fig. 108. Habilitar modo de búsqueda extendido en Notepad++ 109
Fig. 109. Flujo del funcionamiento open() 110

TABLAS

Tabla. 1. dimensiones de globo ocular 31
Tabla. 2. Lista obtenida de los índices de los vértices Iris 61
Tabla. 3. Lista obtenida de los índices de la superficie del ojo. 63
Tabla. 4. Sección de código de Creacion_Materiales_Ojo.py (Parte 1) 66
Tabla. 5. Parte de código sobre Frame Node 67
Tabla. 6. Parte de código sobre Mapping Node ‘Normal’ 69
Tabla. 7. Parte de código sobre Mapping Node ‘Point’ 70
Tabla. 8. Parte de código sobre Voronoi Texture Node 72
Tabla. 9. Parte de código sobre Color Ramp Node 74
Tabla. 10. Parte de código sobre Color Ramp Node (parada) 75
Tabla. 11. Parte de código sobre Color Ramp Node (borrar blanco y negro en extremos) 76
Tabla. 12. Parte de código sobre Mix Node 77
Tabla. 13. Parte de código sobre Gradient_Texture_Cornea 78
Tabla. 14. Parte de código sobre Texture Coordinate Node 79
Tabla. 15. Parte de código sobre Texture Noise Node 80
Tabla. 16. Parte de código sobre Bump Node 81
Tabla. 17. Parte de código sobre Material Output 82
Tabla. 18. Parte de código sobre Principled BSDF 84
Tabla. 19. Parte de código sobre invención de un nodo 86
Tabla. 20. Parte de código sobre invención de enlace de un nodo 87
Tabla. 21. Parte de código sobre asignación de material 88
Tabla. 22. Parte de código sobre seleccionar vértices 89

Tabla. 23. Parte de código sobre cambiar de modo de vista.....89
Tabla. 24. Código de script Aplicar_Materiales.py.....93
Tabla. 25. Código de script Aplicar_Materiales.py parte función open().....98
Tabla. 27. Asignar valor al contador.....99
Tabla. 28. Flujo condicional de variable cont 100
Tabla. 29. Código imprimir_vertices_seleccionados.py.....112
Tabla. 30. Código de seleccionar_vertices_ojo.py 115
Tabla. 31. Código de Creacion_Materiales_Ojo.py 141

Resumen

El objetivo principal de este trabajo de fin de grado es continuar con la herramienta de realización de retratos robots tridimensionales donde se trabaja con la Policía Científica. Para lograrlo se han creado las mallas de los ojos pasando por: el entendimiento del espacio de trabajo “Shading” que ofrece Blender para la creación de materiales y posteriormente poderlos aplicar en las mallas, saber cómo funcionan los nodos tanto a nivel de usuario, como de desarrollador, como de programador; y desarrollar scripts los cuales crean y aplican los materiales en las mallas oculares para crear el ojo y poder cambiar el color del iris, las cuales además de para la identificación de un sujeto también servirán para en un futuro poderse utilizar en el desarrollo de gestos y transformaciones del mismo.

Summary

The main objective of this final degree project is to continue with the tool for the realization of three-dimensional robot portraits, identikits, where we cooperate with the scientific police. In this project we have created the meshes of the eyes going through the following steps: understanding the workspace "Shading" that Blender offers for the creation of materials and then be able to apply them in the meshes, knowing how the nodes work the levels of user, developer and programmer, and implement scripts which create and apply the materials in the eye meshes to create the eye and to change the color of the iris, which in addition to the identification of a subject will also serve in the future to be used in the development of gestures and transformations of the person who we are trying to find out who it is.

Resumen extendido

Este trabajo de fin de grado parte de otro anterior *Estudio y desarrollo de una aplicación de retrato robot en Blender* por José María Oliet Villalba, el cual consiste en la elaboración y creación de la arquitectura de un programa, a través de Python en la ventana de Scripting, que como función principal tiene la de crear, cargar un perfil que contenga:

- Zonas faciales: hacen alusión directa a la abstracción de los vértices que conforman los sectores
- Macrozonas: conjunto de zonas que contiene una macrozona.

Para que se entiendan mejor estos conceptos plasmaremos un ejemplo:
El ojo derecho de una persona sería la macrozona mientras que el párpado, la pupila, la esclerótica y el iris serían las zonas en las que se divide dicha sección.

Como veremos en este documento las partes realizadas para este trabajo de fin de grado han sido:

- **Familiarizarse con la aplicación Blender**, la cual tiene muchos conceptos siendo necesario aprender los más básicos para seguir con la aplicación de retratos robot tridimensionales. Respecto a la continuación del menú creado por el anterior alumno ha sido necesario indagar sobre cómo crear mallas, como funcionan los vértices, las aristas, las caras que la forman. Como se pueden añadir, eligiendo el espacio de trabajo que más se adapte para cada caso como puede ser crear la mesh, interactuar con ella para añadir vértice y más.

- **Continuar el proyecto de crear una aplicación de retrato robot,** el cual se ha tenido que entender el TFG previo a este, y saber cómo adaptarse para poder proseguir. Para ello la parte que dejó el anterior alumno como posible seguimiento era la de implementar transformaciones y gestos en el programa, pero dicha idea ha sido desestimada ya que para poder aplicar las

transformaciones y ver cómo repercuten deben de existir las mallas, y para gesticular estaríamos en un caso similar. Además, que facilitará el entendimiento de la continuación del desarrollo de la aplicación del retrato robot viendo ejemplos prácticos con dichas mallas.

- **Entendimiento de creación de malla** creando los ojos que se aplicaran en un futuro al busto. Para poder hacer los gestos es necesario tener la cabeza creada, es por ello por lo que se ha comenzado por los ojos, viniendo como primer gesto básico el de poder dirigir la mirada de la recreación del sujeto. Para ello se ha adquirido conocimientos de creación de objetos en Blender.

- **Crear y aplicar materiales a la malla creada a través de Blender** esta parte como la anterior podríamos decir que se ha hecho desde el punto de vista de rol de diseñador, donde ha sido necesario **comprender el funcionamiento de los nodos**, los cuales son esenciales para dar textura al ojo y el cómo pueden aplicarse varios materiales a una misma mesh y ver cómo son utilizadas juntas para una finalidad común.

- **Trasladar los conocimientos del funcionamiento de los nodos a scripting de Python** este caso ya sí que se ha realizado desde el punto de vista de un programador, pues al poder "traducir" la creación de los nodos, los valores de las propiedades que deben tener, los enlaces que hay entre ellos para crear un material se ha podido ver de una forma gráfica y entretenida como se puede programar en Python y cómo repercute en la aplicación de Blender,

- **Aplicar funcionalidades de Blender a través de Python,** como la selección de vértices a través de script, la función de seleccionar objetos, espacios de trabajo, la creación de materiales y la aplicación de ellos sobre un objeto.

Palabras clave

Material, Blender, 3D, Phyton, Malla.

Key words

Material, Blender, 3D, Phyton, Mesh.

Glosario

CDG [1] **:** Siglas de “Centro De Gravedad” en referencia al punto medio de los objetos o figuras que creamos en Blender.

Diagrama de Voronoi [2]**:** Estructura característica en el mundo de la geometría donde principalmente se basa de proximidad entre los puntos que están contenidos dentro de un plano.

Escena [3]**:** *Orientado a Blender*. Contiene colecciones.

Faceta [4]**:** *Orientado a geometría*. Rasgo característico de una estructura geométrica que es la cara que surge de anidar los vértices. También denominado cara.

Linkear [5]**:** *Anglicismo.* Frecuente en el mundo informático, conectar dos elementos como pueden ser los nodos o los hipertextos.

Malla [6]**:** *Orientado a Blender*. Estrechamente unido al diseño tridimensional, estructuras geométricas surgidas de la unión de vértices, aristas y caras, encargados de crear el volumen deseado.

Mesh [7]**:** Malla.

Modo blend [8]**:** Utilizado para fusionar un color con la malla como es el modo opaco.

Multiplataforma [9]**:** *Orientado a Informática*. Característica que hace que una aplicación pueda ser ejecutada en diversos sistemas operativos.

Open-source [10]**:** Término en inglés que significa “Fuente abierta” o “ Código abierto”. Como su propio nombre indica se trata de un software de código abierto que pretende dar licencia a los programadores de modificar, cambiar, redistribuir y mejorar este

código, esto permite poder utilizarlo en aplicaciones desarrolladas por otros programadores externos a Blender. Es gratuito.

Operadores [11]**:** *Orientado a Blender*. Son modificadores que ofrece la aplicación los cuales suelen ser no destructivas al modelo final. Es decir, cuando cierres la aplicación volverá a su estado anterior.

Reflexión [12]**:** Fenómeno físico característico porque el rayo de luz solar que se aplica sobre un material haciendo que rebote dicho rayo.

Refracción [13]**:** Fenómeno físico característico porque el rayo de luz solar desvía su dirección al atravesar un material de distinta densidad.

Shade smooth [14]**:** *Orientado a Blender*. Función que aumenta el número de vértices, aristas y facetas de una malla haciéndolo más definido. Se utiliza sobre todo para superficies curvas.

Tomas VFX [15]**:** Fotogramas donde se aplican efectos visuales.

Vértice [16]**:** *Orientado a Blender*. Elemento que sirve como unión de aristas que forman la geometría de una malla.

Workspace [17]**:** *Inglés.* Espacio de trabajo, es una ventana donde la aplicación ofrece unos editores por defecto.

Glosario de atajos de teclado

ALT + J [1]**:** Combina las facetas contiguas seleccionadas, eliminando el borde o arista que las separa.

CTRL + I [2]**:** Selecciona los vértices, las caras o las aristas inversas; es decir, las que antes no estaban seleccionadas pasarán a estar seleccionadas y viceversa.

CTRL + 3 [3]**:** Subdivide la superficie añadiendo más vértices, caras y aristas a la malla.

CTRL + SHIFT + CLIC DERECHO [4]**:** En el espacio de trabajo de Shading hace que el nodo seleccionado se enlace automáticamente con la salida del material para que se vea la repercusión que tiene sobre el material ese nodo específico y los enlaces anteriores que este pueda tener.

CTRL + T [5]**:** El nodo seleccionado se enlaza automáticamente con la creación de un nodo de tipo Texture Coordinate.

E + S [6]**:** Extiende la malla como es del borde seleccionado y con la tecla S podemos escalar la selección.

G + (X/Y/Z) [7]**:** Permite mover la malla con el puntero hasta clicarle, si se añada cualquiera de las teclas X, Y o Z permite mover la malla por el eje que alude la tecla.

M [8]**:** Une cada uno de los vértices seleccionados en un vértice central.

R + (X/Y/Z) + 90º [9]**:** Rota la malla noventa grados en sentido del eje indicado.

S + (X/Y/Z) [10]**:** Escala la mesh en sentido del eje indicado

SHIFT + A [11]**:** En el espacio de trabajo de Shading sirve para crear un nodo de material utilizando un buscador.

SHIFT + D [12]**:** Hace una copia de los vértices, borde o caras seleccionado.

SHIFT + H [13]**:** Permite ver el teñido de los colores azul o rojo en función de hacia dónde está orientada las facetas de una malla.

SHIFT + N [14]**:** Invierte la orientación de la cara o caras seleccionadas.

SHIFT + S [15]**:** Menú auxiliar para cambiar el modo del puntero mouse.

1. Introducción

1.1 Definición y evolución del retrato robot.

Cuando se necesita identificar a un delincuente y no se tienen evidencias de quién puede ser la persona, ni de ningún documento audiovisual con el que se pueda utilizar para el reconocimiento de éste, la policía hace uso de los retratos robots.

Un retrato robot es la reconstrucción gráfica de un individuo tomando en consideración sus rasgos físicos como alternativa de no contar con una fotografía del ser bajo sospecha. Dichos rasgos físicos se obtienen a través de personas que han presenciado aquello por lo que se haya puesto en investigación el caso o hechos relacionados en espacio y tiempo. Otras veces puede darse la situación de que sí se tenga algún documento audiovisual, pero con muy baja resolución que haga imposible la identificación de la persona. Es por ello por lo que un retrato tenga una importancia a la hora de investigar, pues a falta de un documento real permite identificar al sospechoso, permitiendo a la policía la posibilidad de encontrar algún tipo de rastro que conduzca al infractor. En los principios de esta técnica era habitual que los dibujos los hiciera un retratista basándose en los datos proporcionados por testigos o una víctima directa. Hoy en día esto se procesa a través de un ordenador, un estudio del Instituto de Computadores y Ciencias de Información de la Universidad de Czestochowa, Polonia, ha reflejado los elementos más notables a la hora de investigar a un sospechoso. Como son rasgos de la cara como la nariz, los ojos, la forma del cráneo, las orejas, las cejas, los labios y demás que permiten al programa realizar una recreación facial con las características proporcionadas.

Fig. 1. Ejemplo de boceto

Con un retrato de la persona en cuestión se puede lograr la identificación tras un periodo largo de tiempo pues la base de datos cuenta con muchos tipos de rostros para las comparaciones con el boceto ,como el que podemos apreciar en la *Fig. 1. Ejemplo* de boceto, y sería necesario comparar con cada uno de esos datos, volviéndose un proceso tedioso. Con la ayuda de un sistema informático que compare rostros y los almacene en la base de datos este procedimiento se simplifica bastante. Dicho sistema informático aplica la matemática para comparar el modelo. Este método puede funcionar de dos formas:

- **Aprendizaje:** se utiliza para identificar el patrón de las caras básicas, como podemos apreciar en la *Fig. 2. Diagrama del proceso de identificación a través del modo aprendizaje.*

- **Pruebas:** utilizado para identificar al sospechoso comparando el retrato del autor con las imágenes almacenadas en la base de datos, devolviendo una selección de los resultados más similares, como podemos observar en la tercera figura, *Fig. 3. Diagrama del proceso de identificación a través del modo de pruebas.*

Fig. 2. Diagrama del proceso de identificación a través del modo aprendizaje.

Fig. 3. Diagrama del proceso de identificación a través del modo de pruebas.

Los algoritmos más utilizados para el reconocimiento del sujeto sobre la imagen de un rostro visto de frente son:

- **Transformada Wavelet:**
  Conjunto de herramientas y técnicas las cuales hacen posible la evaluación y el tratamiento de forma robusto y eficiente gran variedad de señales de nuestro entorno, como puede ser sísmicas como los sonidos que emite la voz, pasando por imágenes como sería el caso de este trabajo. Consiste en dividir una señal en versiones desplazadas y escaladas de una "onda pequeña" que se la conoce como wavelet madre.

- **Modelos Ocultos de Markov (MOM):**
  Son autómatas abstractos de estados finitos que hacen posible el modelar ocurrencias de procesos con evolución en el tiempo aleatoria, donde dicha ocurrencia de los estados está asociada con a una distribución de probabilidad y donde las transiciones entre los estados están gobernadas por un conjunto de probabilidades.

A la hora de buscar parecidos en la base de datos al aplicar un dato nuevo en ella, el proceso de identificación en modo aprendizaje tendría que repetir el proceso de aprender todo el sistema desde el principio, no sirviendo todo lo anterior estudiado. Para este trabajo no se van a tener en cuenta la forma de obtener resultados de parecidos; aprenderemos cómo modelar

Partes de lo que podríamos llamar retrato robot "moderno", teniendo este como diferencia principal que no se trataría de dibujar sobre papel, si no con ayuda de la aplicación Open-source [10] Blender , utilizada para diseño 3D. Esto nos daría la libertad de verlos en tres dimensiones, la posibilidad de crearlo más rápido y aplicar cambios inmediatos e incluso, en un fututo, verlos en movimiento (pestañeo, tics, respiración), elementos que quizás podrían ayudar a identificar a individuos de manera más sencilla.

Retrato robot que sería una especie de boceto, pero en vez de ser dibujado y en dos dimensiones dando una vista tridimensional, haciéndolo más semejante a la vida real, por lo que no nos detendremos en la explicación de estos algoritmos y más procesos que componen la parte de identificar a una persona, pero sí es necesario tener en cuenta la continuación de la finalidad de nuestro trabajo para hacerlo lo más compatible posible.

1.2 Objetivos

- **Objetivo general**

Se plantea desarrollar una herramienta para que pueda ser usada por la policía científica, la cual debe de ser intuitiva para que como resultado final obtengamos una especie de entorno gráfico donde sin complejidad el usuario al que va destinado pueda reflejar los rasgos del sospechoso en cuestión.

Para obtener este resultado este trabajo comenzó con el planteamiento y parte de desarrollo de la creación del menú interactivo a través de operadores de Python los cuales pueden ser ejecutados en la aplicación de Blender. Dicho programa será un pilar fundamental de este proyecto ya que es la aplicación Open-Source por excelencia de diseño en tres dimensiones.

- **Objetivos específicos**

Los objetivos específicos para este trabajo de fin de grado para poder seguir con el desarrollo de una herramienta para el reconocimiento de sujetos mediante retratos robot en tres dimensiones a través de la aplicación Blender han sido:

- **Familiarizarse con el entorno que ofrece la aplicación de Blender**: además del entendimiento de los principios básicos de dicho software, como es el saber qué entornos de trabajo hemos de usar en cada momento, cual es la vista más afín según el tipo de interacción que deseamos con la malla o con los elementos que forman dicha mesh como son los vértices, las aristas y las facetas.

- **Comprender el trabajo de fin de grado anterior:** para poder avanzar con la herramienta previamente pensada para que todo lo que se haga en este proyecto sea compatible y tenga un sentido del porqué se ha optado por realizar así los avances.

- **Desarrollar y plantear modificaciones y gestos:** este paso que estaba estipulado en el anterior proyecto ha sido importante tener en cuenta la herramienta pensada por el anterior alumno y sobre todo las especificaciones transmitidas por la policía científica. Como indicación importante dada es la de poder realizar modificaciones en la mesh sin generar un mal impacto en el resto de la malla, es por ello por lo que se ha optado por en vez de comenzar directamente con modificaciones y ojos se ha tenido que crear una malla que haga referencia a los ojos, para poder realizar las modificaciones como son el cambiar el color del iris de los ojos y en futuros pasos que estos puedan realizar gestos como son mover los ojos sin tener impacto en el resto de la cara.

- **Entender anatómicamente de manera general un ojo humano e implementar la malla tridimensional en Blender:** a través de varias búsquedas por internet y entrando en documentaciones se ha tenido un conocimiento de cómo es anatómicamente un ojo, sobre todo en la parte externa que es la que podemos observar y una vez entendido a través de aprender el funcionamiento de Blender, con nuevos conocimientos para este proyecto se ha creado a través de la unión de vértices, aristas y facetas colocadas específicamente en los lugares determinados y con sus respectivos tamaños se ha obtenido la mesh que permitirá observar en un futuro los ojos que contendrá dicho retrato robot.

- **Creación de materiales a través del entorno gráfico de Blender:** esta parte del proyecto ha sido compleja ya que podríamos decir que es la que más se aleja de los conocimientos que poseemos, pues ha sido necesario

comprender cómo funcionan los nodos que existen en el espacio de trabajo Shading y saber cuáles se deben de aplicar, con qué propiedades, qué parámetros y qué enlaces se deben de crear para poder ver una textura de la malla que se aprecie como un ojo lo más cercano de la realidad.

- **Programar scripts "auxiliares" de Python:** estos scripts como son los de imprimir los índices de los vértices, los de seleccionar los vértices pasados en una lista de enteros y el de creación y aplicación de material han sido necesarios para el entendimiento de Blender a nivel de programación para poder aplicar los scripts principales de este proyecto.

- **Creación y aplicación de materiales a través del scripts de Python:** previamente habiendo comprendido el funcionamiento de los nodos a través del entorno gráfico de Blender, y con ayuda del modo desarrollador que ofrece Python para poder ver el nombre de los tipos de los nodos en cómo se utilizarían a través de Python ha sido posible crear los materiales usando scripts de Python. En este apartado cabe destacar el entendimiento de los nodos a nivel de código pues en rasgos generales podemos decir que está formado por las propiedades y valores que contiene cada tipo de nodo la cual son distintas entre nodos y es importante la función que desempeñan, las entradas que sirven para enlazar el nodo con otro nodo creado previamente y las salidas las cuales sirven para en un futuro enlazar este nodo con otro que se cree, cabe destacar que es habitual ver que no todas las entradas y salidas se utilizan.

- **Modificaciones y aplicación de materiales a través del scripts de Python:** Al haber programado el script que crea dos materiales el de superficie del ojo y el del iris se ha podido implementar las modificaciones que cambiarán los colores del iris cambiando valores de propiedades en los nodos color ramp, teniendo estos una función de degradado de colores y son los que en el material del iris reflejan el color que hemos determinado a

través de los valores de los colores RGB para obtener ojos azules, marrones y verdes. Una vez creados estos nuevos materiales del iris se ha podido crear una herramienta para poder aplicar los materiales en los ojos a través de un botón para que sea sencillo de cambiar.

- **Propuestas para futuros trabajos:** este objetivo intenta facilitar el proceso de pensar en cómo seguir el proyecto, obviamente son propuestas y toda nueva ocurrencia es bienvenida, donde que más se recomienda es seguir con la creación de las partes básicas que tiene una cara para poder ser tratado de forma independiente pero que juntos forman la cara del sujeto en cuestión, se recomienda que se puedan tratar de forma independiente para que al modificar uno de ellos no implique un aspecto negativo en cualquier otra parte que forma la cara o en la cara en sí, posteriormente una vez se tenga el resultado deseado se podrá seguir con el desarrollo de la herramienta planteada por el anterior alumno, para poder controlar dichas modificaciones y gestos de la manera más sencilla posible.

1.3 Estructura de la memoria

Para conseguir estos objetivos planteados el resto de la memoria se estructura de la siguiente manera: teniendo en cuenta de que es necesario conocer el funcionamiento de Blender y haber planteado un desarrollo previo del funcionamiento de la herramienta que se quiere llevar a cabo donde ambas se podrán ver reflejadas en el punto *"2. Fundamentos sobre el trabajo de fin de grado"* de este documento. Una vez comprendido podremos pasar a progresar con el proyecto general, el cual en este trabajo se ha elegido por comenzar con las modificaciones sobre mallas, las cuales pretenden poder implementar versatilidad para el buen funcionamiento de la herramienta, ya que no queremos que al modificar una parte específica del rostro conlleve una consecuencia negativa sobre las demás partes de dicho rostro, es por ello que se ha optado por implementar el órgano del ojo humano para poder en un futuro al implementar gestos o modificaciones cumpla con lo estipulado, para ello en el punto *"3. Contribuciones al desarrollo de una aplicación de retrato robot sobre Blender"* se expone la

investigación llevada a cabo y la implementación de esta en Blender a través de la parte gráfica que nos ofrece la aplicación, además de explicar el procedimiento a seguir para la implementación, además de reflejar en la aplicación de los materiales sobre la malla en el punto *"4. Resultado de aplicar los materiales de "Superficie Ojo" y de "Iris"."* Tras este entendimiento del entorno gráfico con el apartado *"5. Scripting material para los ojos"* se ha optado por entender a nivel de código el funcionamiento de los nodos, los cuales se encargan de asignar los valores de las propiedades y enlazarlos entre ellos con conexiones específicas para poder aplicar esa capa de "pintura" sobre la superficie obteniendo una aproximación a la realidad. Posteriormente se ha ampliado el script encargado para crear más materiales que influyan en el color de los ojos en el apartado "*6. Ampliación de script (Parte I) : "Creación_Materiales_Ojo.py""* y poderlos aplicar en el punto "*7. Ampliación de script (Parte II) : "Aplicar_Materiales.py""*. Seguidamente en el *"8. Ejecutar scripts de Python en Blender."* se demuestra el proceso a seguir para cargar los archivos expuestos a lo largo del proyecto y poderlos procesar en el entorno de trabajo de Scripting. Posteriormente se refleja como la implementación desarrollada en este trabajo se aplica con el anterior a través del apartado *"9. Enlazando las mallas creadas con herramienta del TFG anterior."* En el desarrollo de este proyecto nos hemos encontrado una serie de problemas expuestos en el punto*"10. Problemas encontrados"* donde se demuestra cómo hemos podido resolver con éxito. Como antepenúltimo *"11. Scripts involucrados en el proyecto"* apartado tenemos el código de los scripts que forman parte de nuestra contribución al desarrollo para la aplicación del retrato robot. Y para finalizar en el apartado *"12. Conclusión y futuros trabajos"* se expone la conclusión de la contribución implementada en este proyecto y planteamiento de pasos futuros que serían destacables contemplar.

2. Fundamentos sobre el trabajo de fin de grado

2.1 Proyecto anterior

Antes de comenzar a explicar el contenido del proyecto es necesario saber de dónde partimos, pues dicho proyecto es una continuación del trabajo de fin de grado anterior, ***Estudio y desarrollo de una aplicación de retrato robot en Blender*** realizado por el alumno Jose Maria Oliet Villalba. Dicho proyecto precedente consistió principalmente en el diseño de un menú auxiliar dentro del programa de Blender como herramienta para poder dividir una cara, sobre la que se trabajará el retrato tridimensional, para dividirla en distintas partes como ojos, nariz y boca y poder así hacer transformaciones sobre ellos. Dicho programa puede ser utilizado , de forma general, por tres tipos de usuarios.

- **Programador:** rol que implica el conocimiento de la arquitectura interna, las bases teóricas, la interacción con la aplicación de Blender. Estos pueden probar su correcto funcionamiento, rectificar de no ser así y de interactuar con comandos que ofrece Blender a través del lenguaje Python por el entorno.
- **Diseñador:** papel que será llevado a cabo por un perfil orientado a la edición, el cual no necesita necesariamente comprender la arquitectura a nivel de código, pero sí la base teórica del funcionamiento de las zonas, las macrozonas y transformaciones.
- **Usuario básico:** grupo de usuarios, como podría ser la persona que le toca retratar de manera tridimensional a un sujeto, el cuál le es transparente la parte teórica y la de código dichas anteriormente. Sí que sería necesario conocer el funcionamiento de la herramienta diseñada, la cual será intuitiva para hacer más simple el proceso de aprendizaje.

La arquitectura de la herramienta creada por el anterior alumno consiste en utilizar los vértices [16] que componen las mallas [6], pudiéndolos agrupar a través de código el cual se enlace con la interfaz de usuario, que a su vez debe representar de manera sencilla los conceptos dichos para que la persona pueda interactuar de la manera más cómoda posible.

Esta interacción entre los datos de almacenamiento y el código de enlace trata principalmente en una articulación de gestión de ficheros. A continuación, en la *Figura 4*, se plasma el funcionamiento de la herramienta al seleccionar una macrozona.

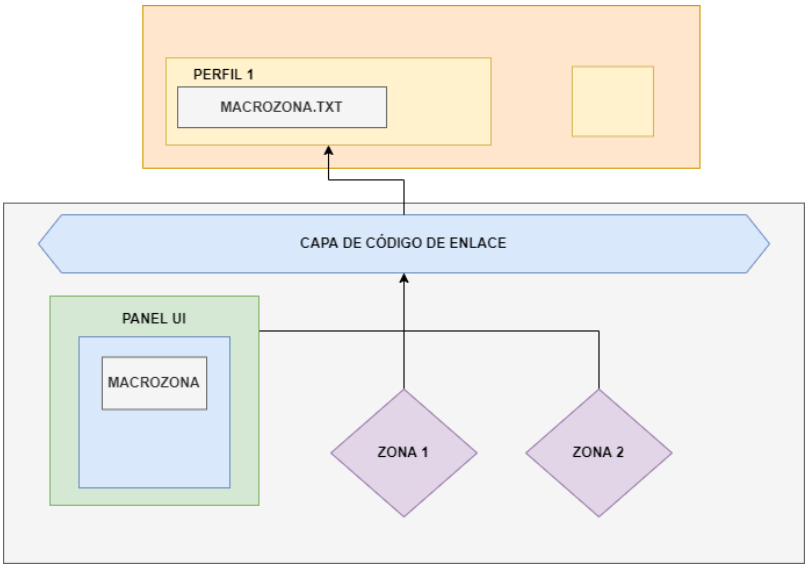

Fig. 4. Comportamiento de herramienta al seleccionar macrozona

Esta figura podría estar tratándose de la selección de la zona ojo la cual tendría asociada las zonas faciales párpado, iris, pupila, esclerótica y córnea. La capa de código de enlace es donde se encuentran los operadores [11] que interactúan con los archivos almacenados. Cada perfil que se guarde se almacenará como un fichero distinto el cual lo hará único y pudiendo ser identificado sobre los demás. Dicho perfil almacenado tendrá registrado las zonas, las macrozonas y las transformaciones de la mesh[7] implicada. Estas macrozonas no se pueden ver reflejadas en la aplicación de Blender tal cual es por ello por lo que se optó por acumularse en un fichero de texto (.txt) como podría ser el ojo del que hablábamos antes, éste se registraría de la siguiente manera:

Ojo.txt → [5,‘párpado’,‘iris’,‘pupila’,‘esclerótica’,‘córnea’,0]

Donde como la posición inicial de dicha lista hace alusión a un número entero el cual representa el número de zonas que hay asociadas a dicha macrozona ,la cual para

mejorar su entendimiento hemos reflejado en la *Fig. 5. Macrozona del ojo con sus respectivas zonas asociadas*, posteriormente tendremos tantas posiciones en la lista como número de zonas devuelva el primer elemento. Como último elemento tendremos un número entero el cual hace alusión al número de transformaciones asociada.

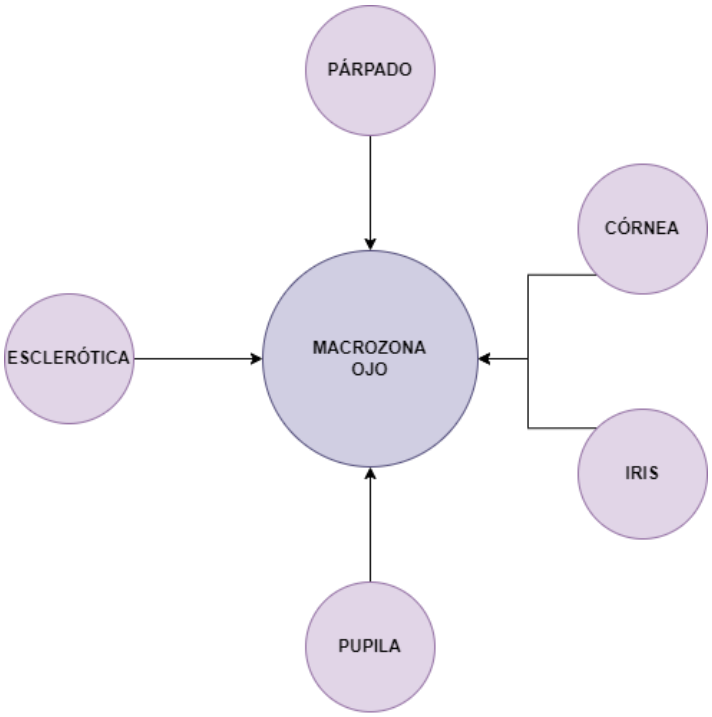

Fig. 5. Macrozona del ojo con sus respectivas zonas asociadas

Las transformaciones son cualquier ejecución del programa que implique el cambio de localización de los vértices, pues eso en función de cómo se actúe implica una transformación en la malla, por ejemplo, si tenemos la mesh de un ojo y todos los vértices pertenecientes a ella misma los cambiamos de posición a 5 centímetros más de su posición inicial estaremos agrandando dicha malla.

A continuación, en la *figura 6* dejaremos una captura de pantalla de la aplicación Blender de la herramienta creada por el anterior alumno, en la cual podemos apreciar los ajustes de perfil, pudiendo crear, guardar o eliminar un perfil. Posteriormente las macrozonas que tiene asociadas dicho perfil cargado. Seguidamente un panel de zonas faciales las cuales nos permite añadir, guardar o eliminar una zona de las que estarán

asociadas a la macrozona. En cada una de ellas podemos ver un desplegable el cual nos indicará los elementos que hay almacenados en ellas.

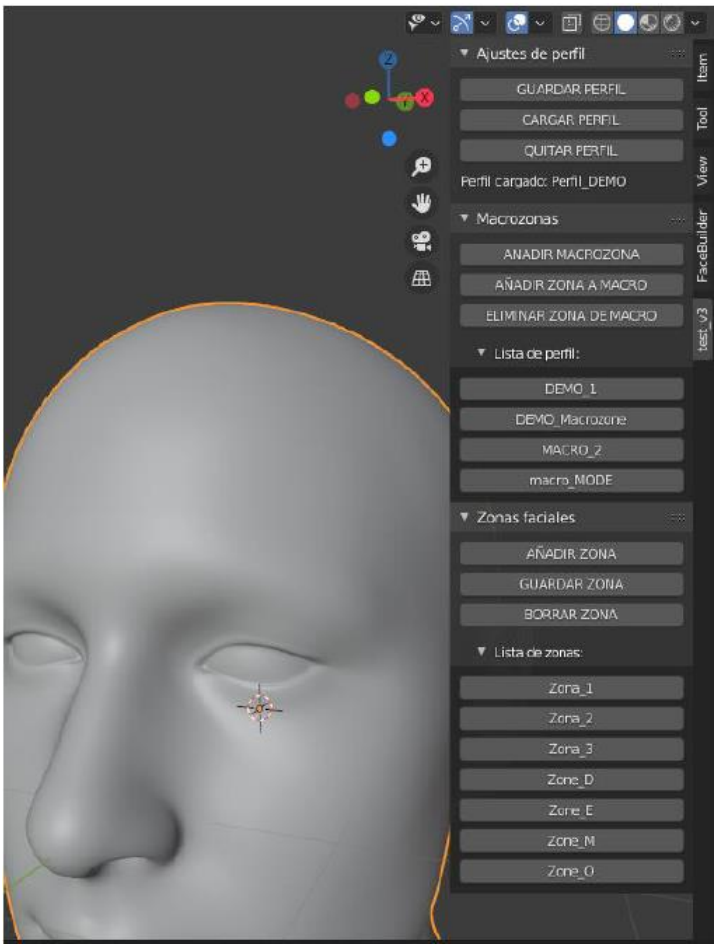

Fig. 6. UI Panel creado por la herramienta del anterior alumno

2.2 Preámbulo de la aplicación Blender.

2.2.1 ¿Qué es Blender?

Blender es una aplicación multiplataforma [9] orientado a usuarios que se dedican al mundo artístico, del diseño y de la multimedia. Esta herramienta admite la creación de modelados tridimensionales y vídeos de cualquier resolución, pero no sólo está limitado a los vídeos, si no que también se pueden crear imágenes fijas, animaciones tridimensionales, editar vídeo y tomas VFX [15]. La parte más importante por la que este proyecto se ha decidido trabajar con Blender es porque es un programa gratuito y Open-

source, por lo que nos permitirá trabajar de forma libre sin limitarnos por falta de licencia o de derechos.

Como curiosidad Blender ha sido utilizado en películas, por lo que da un carácter más profesional a la aplicación gratuita con herramientas tan potentes como para poder hacer trabajos de tal magnitud, haciendo constancia de que es una herramienta usada por muchos usuarios y no deja nada que desear comparándola con softwares de modelaje 3D que necesiten licencias. Un ejemplo de películas realizadas con este útil es *Robot 7723*, la cual podemos apreciar en *Fig. 7. Fotograma de la película Robot 7723 creada con Blender* conocida en inglés como *Next Gen*. La cual se estrenó en la plataforma Netflix el siete de septiembre del año dos mil ocho.

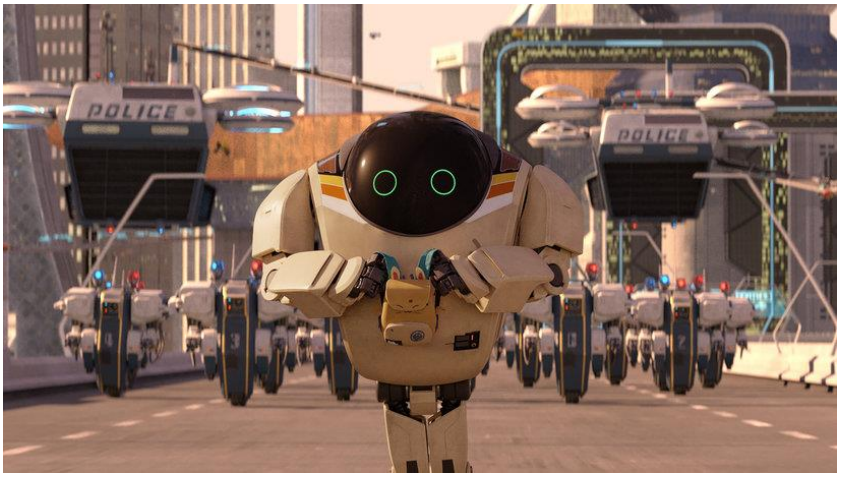

Fig. 7. Fotograma de la película Robot 7723 creada con Blender

2.2.2 Introducción a Blender: Workspace

En esta sección se va a tratar los espacios de trabajo que vienen por defecto disponibles en Blender resaltado en amarillo en la *figura 8* y los tipos de editores contenidos, lo que está en rojo, y para qué se utilizan de una forma genérica.

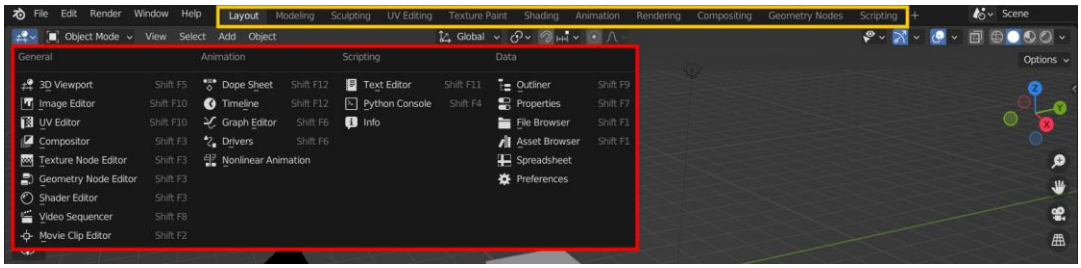

Fig. 8. Pantallazo de los espacios de trabajo y tipos de editores que vienen por defecto en Blender.

- **Layout:** es el workspace [17] que viene por defecto cuando se abre la herramienta. Es de ámbito general para poder previsualizar la escena [3] y los objetos que forman parte de esta. En dicho espacio podemos observar cuatro tipos de los editores disponibles: Vista 3D (resaltado en rojo en la *Fig. 9. Pantallazo del espacio de trabajo Layout*, listado (resaltado en verde) el cual nos muestra una lista organizada para poder ver los datos que hay en la escena que estamos trabajando, propiedades (resaltada en amarillo) y la línea de tiempo que es la azul.

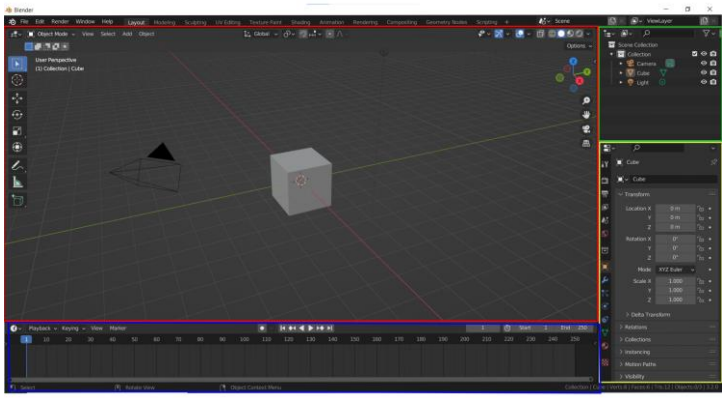

Fig. 9. Pantallazo del espacio de trabajo Layout

- **Modeling:** espacio para transformar la geometría con útiles desarrollados para el modelado.
- **Sculpting:** entorno de trabajo para esculpir la geometría con herramientas especificadas en escultura.
- **UV Editing:** ambiente dedicado al mapeo de coordenadas de las texturas de una imagen a una superficie tridimensional.
- **Texture Paint:** workspace dedicado a poder dibujar sobre el tipo de editor vista 3D.
- **Shading:** distribución dedicada a determinar propiedades de los materiales a la hora de su renderizado.
- **Animation:** espacio de trabajo en el que se involucra una línea temporal en la que cada fotograma quedará almacenado con los vértices, posiciones y modificaciones de manera única para poder hacer animaciones.

- **Rendering:** es el workspace que permite visualizar y analizar de una forma más precisa los resultados del renderizado.
- **Scripting:** es el espacio de trabajo junto al de Shading que más hablaremos en este trabajo, ya que es el que se utiliza para programar scripts en Python. A continuación, se plasma una captura de pantalla de dicho espacio de trabajo en la *Fig. 10. Workspace Scripting de Blender.*

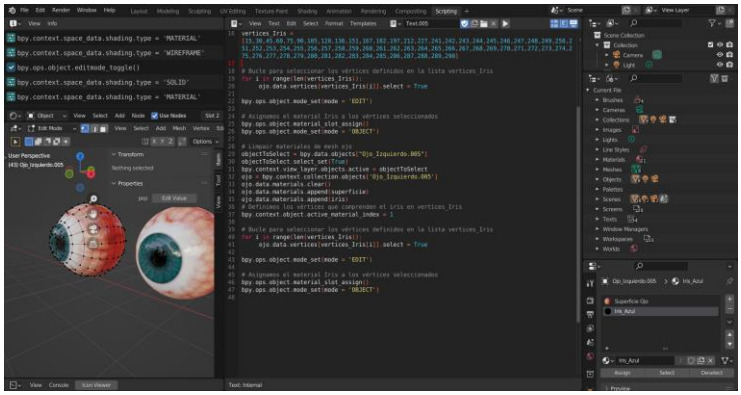

Fig. 10. Workspace Scripting de Blender.

2.2.3 Introducción a Blender: Malla (mesh).

Tras haber dado una ojeada por los espacios de trabajos de Blender, ahora veremos la parte en la que Blender crea y trabaja con las mallas. Una **malla** ,también conocida como mesh, es una estructura basada principalmente en caras poligonales las cuales se van editando para construir una figura geométrica deseada, como podría ser una cara. Para que dicha cara no se vea de manera plana, y con un efecto más definido se pueden crear nuevos **vértices** que formen dicha cara, añadir más caras, eliminar otras… Como podemos apreciar en la *Fig. 11. Malla de un ojo formada por vértices y caras* para que un modelo en 3D se vea bien definido debe tener cuantas más caras/vértices mejor. Donde los vértices son cada punto que apreciamos en dicha imagen y las caras los huecos que quedan entre medias que forma parte de la superficie de la mesh.

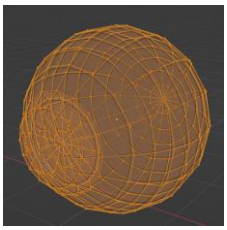

Fig. 11. Malla de un ojo formada por vértices y caras.

2.2.4 Introducción a Blender: Tipos de vista.

Para poder sacar el máximo partido a Blender es necesario conocer los tipos de vista existentes en el sombreado de Blender. Hay dos formas de seleccionar la que más convenga,. La primera sería manteniendo la tecla Z y clicando en la vista deseada, la cual hace alusión a la que apreciamos en la *Fig. 12. Acceso a distintos tipos de vista.* a la izquierda la segunda forma es en la propia barra de tareas superior sobre el entorno de trabajo podremos elegirla con un simple clic, la primera forma suele ser la más utilizada pues una vez aprendido el atajo es más cómodo no moverte de la parte que estás observando en la vista 3D.

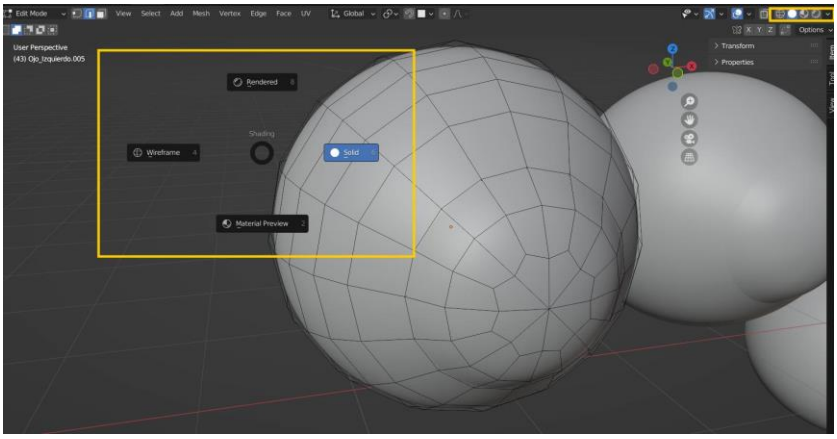

Fig. 12. Acceso a distintos tipos de vista.

- **Estructura de alambre (Wireframe):** Es una referencia directa a cómo vemos la malla, pues parece una especie de alambre interconectada por los vértices. Como se puede apreciar en la *Fig. 13. Vista Wireframe* siguiente las caras nos son visibles, aunque siguen estando presente, dando un efecto de radiografía para poder ver todo lo posible de la malla en sí.

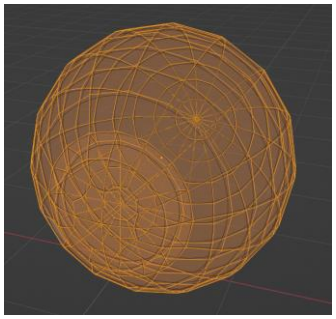

Fig. 13. Vista Wireframe

- **Sólido:** Muestra la escena usando el motor de renderizado Workbench Engine, el cual es eficaz para poder renderizar al mismo tiempo que trabajamos con la escena. Dicho motor no está preparado para el renderizado final de un proyecto de Blender, pero aun así lo podemos seleccionar en ajustes para que así sea. Podemos ver que, en este caso, *Fig. 14. Vista Sólido* , sólo vemos la superficie de la malla pues las caras no tienen transparencia como era el caso anterior.

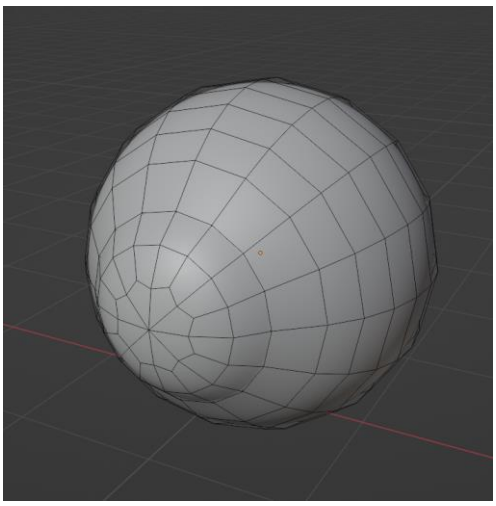

Fig. 14. Vista Sólido

- **Renderizado:** Como su propio nombre indica renderiza con el motor Render Engine teniendo en cuenta la iluminación creada en la propia escena en sí, podemos ver dicho funcionamiento en la *figura 15.*

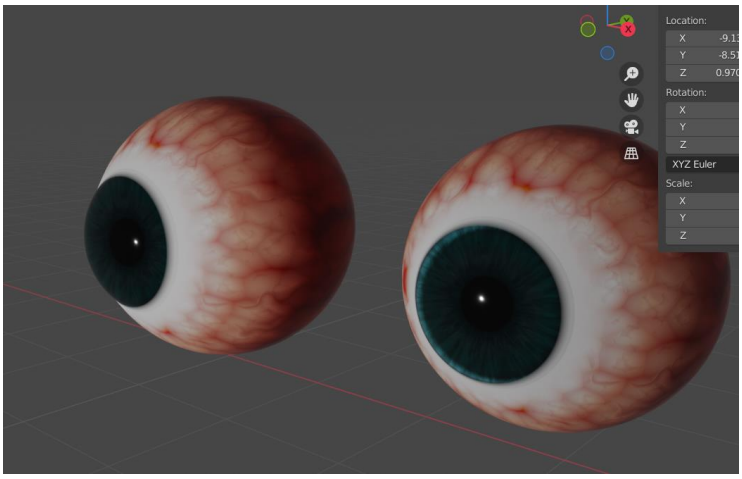

Fig. 15. Vista renderizada

- **Vista previa del material:** Esta vista junto a la de Wireframe es la más utilizada en este proyecto, pues dicha vista nos permite ver aplicado el material que hemos creado y asignado a la malla como se aprecia en la *Fig. 16. Vista previa de los materiales*. El motor de renderizado que utiliza es el Eevee y es el más adecuado para trabajar con materiales.

Fig. 16. Vista previa de los materiales.

2.2.5 Introducción a Blender: Modos de interacción con el objeto

Es importante destacar que para poder modificar a tu antojo una malla, es necesario saber que modos de interacción existen, pues no vale solamente con elegir un tipo de vista adecuado. Estos modos en la aplicación de Blender se pueden seleccionar en la parte superior izquierda del propio tipo de editor, en el caso de la *Fig. 17. Combo box de selección de modo de interacción* está justo al lado de la vista tridimensional.

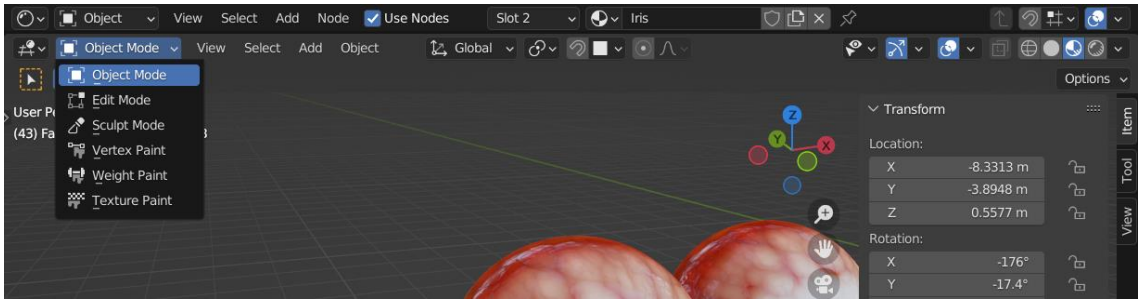

Fig. 17. Combo box de selección de modo de interacción.

Los modos de interacción utilizados en este trabajo de fin de grado son: modo de objeto, modo de edición y modo de esculpir.

- **Modo de objeto:** Es el modo que viene por defecto seleccionado al abrir la aplicación de Blender (*Figura 18*). Disponible para los objetos que están en la escena y está destinado a editar bloques de datos de objetos como puede ser moverlo por el entorno, escalarlo para tener la medida que deseamos, rotar el objeto …

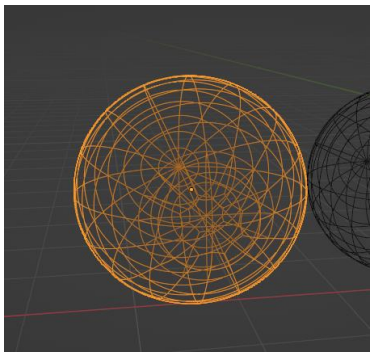

Fig. 18. Modo de objeto.

- **Modo de edición:** Es utilizado para todos los tipos de objetos que puedan ser renderizados, ya que está destinado más a la forma de los objetos; es decir, como por ejemplo seleccionar , eliminar o añadir vértices dentro de una malla. En vez de vértices también aplica para aristas, caras e incluso puntos de control de curvas. Para seleccionar vértices, aristas o caras nos aparecerán unos iconos justo al lado del botón donde podremos seleccionar el que más convenga. En la siguiente figura *Fig. 19. Modo de edición: vértices, aristas y caras* podemos ver vértices, aristas y caras respectivamente.

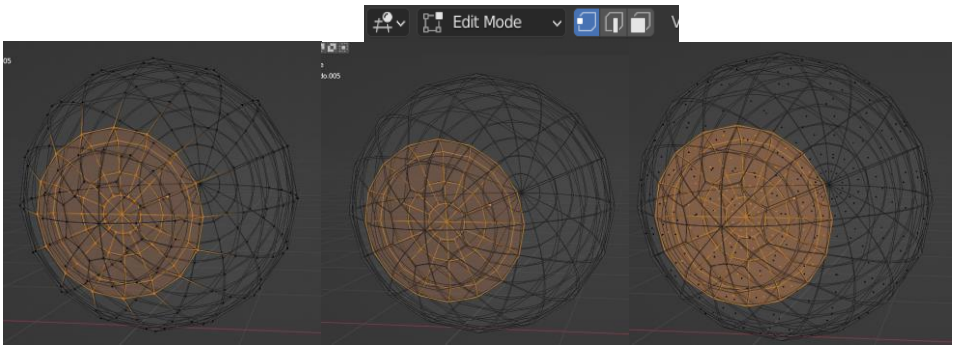

Fig. 19. Modo de edición: vértices, aristas y caras.

- **Modo de esculpir:** modo utilizado sobre mesh a las que se les quiere modificar la superficie sobre la malla eligiendo el pincel que más se adapte a nuestra necesidad. Por ejemplo, podemos ver como en la *Fig. 20. Modo de esculpir*. existe una especie de grabado en la propia mesh.

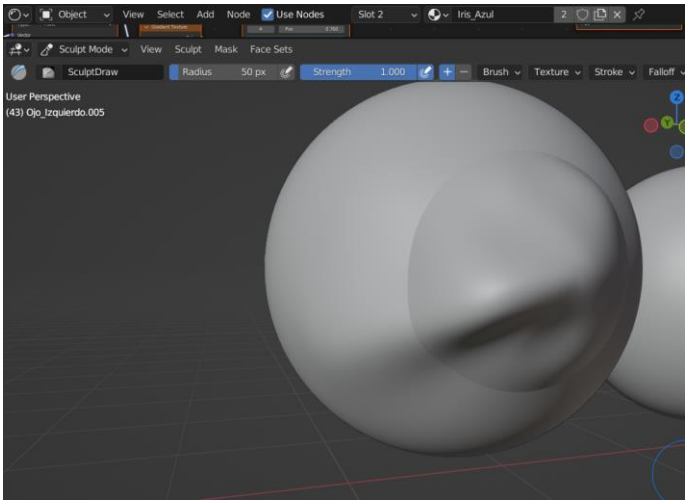

Fig. 20. Modo de esculpir.

Por ejemplo, para seleccionar una malla es necesario estar en el "modo de objeto" pues en cualquier otro modo no puedes seleccionar la mesh y si quieres seleccionar un vértice de la malla previamente seleccionada se necesitará estar en el "modo de edición". Esto es muy importante a la hora de programar en la parte de Scripting de Python, pues por ejemplo al querer aplicar los materiales al ojo primero se tendrá que elegir el objeto en "modo objeto" y posteriormente crearemos los nodos de los materiales y posteriormente para asignar el material de Iris a los vértices deseados se tendrán que seleccionar en el "modo edición".

3. Contribuciones al desarrollo de una aplicación de retrato robot sobre Blender

Como inicio de la realización de este proyecto se estableció una reunión con el tutor donde se comunicó que el proyecto está pensado para que cuando esté completo pueda ser usado por la policía. El cometido principal es el de otorgar una solución para la reconstrucción facial de un sujeto y así poder ser identificado. Para tener en cuenta a la hora de las modificaciones se requiere que al escalar las zonas faciales no implique deformar el resto de la cara, pues los softwares que han probado es un inconveniente que tienen a la hora de intentar recrear la cara del sujeto.

Este proyecto tiene un desarrollo previo, por lo que no se comienza desde cero, pues el anterior alumno comenzó una herramienta para poder hacer una especie de menú la cual permite cargar, guardar o eliminar un perfil. Dicho perfil se almacena en un .txt único. Además de también hacer una sección en la que se pueda seleccionar las zonas faciales para poderlas juntar en macrozonas. Como continuación de esta herramienta anteriormente se estimó que se podría realizar las transformaciones y los gestos, pero para este paso hace falta uno intermedio el cual sería la creación de los objetos necesario. Es por ello por lo que como primer gesto y modificación se estimó el de poder mover los ojos y la de cambiar los colores del iris de los ojos.

Para esta parte podríamos decir que se ha trabajado desde el punto de vista de un usuario diseñador, pero tras reflexionar es una parte necesaria para poder avanzar como nivel de usuario de programador. Por lo que a continuación ha sido necesario el crear una malla que represente un ojo a través de las utilidades que ofrece la aplicación Blender en sí, pero antes de crear nada es necesario entender de manera básica la anatomía de un ojo, *Fig. 21. Anatomía del ojo* , para poder ser aplicada de la manera más correcta posible .

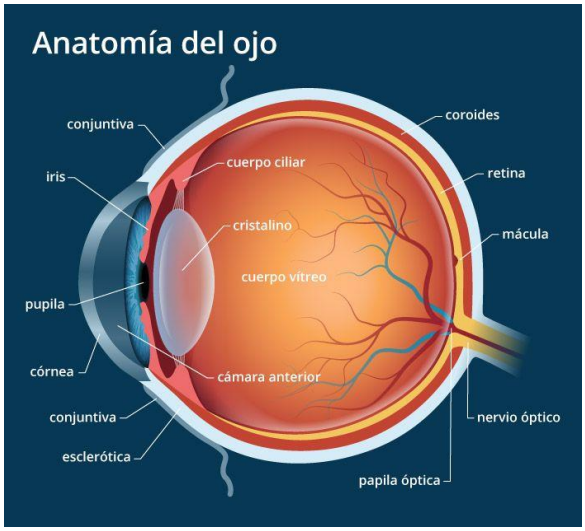

Fig. 21. Anatomía del ojo.

3.1. Análisis general del ojo humano

La información plasmada en la *Tabla 1. dimensiones de globo ocular* la cual contiene valores de cómo es anatómicamente el ojo humano desde un punto de vista generalizado se ha obtenido principalmente en este blog para poderlo aplicar en este proyecto.

DIMENSIONES DE GLOBO OCULAR	
Diámetro vertical	23 milímetros
Diámetro transversal	23,5 milímetros
Diámetro anteroposterior	De 22 a 24 milímetros

Tabla 1. dimensiones de globo ocular.

Durante la creación de la mesh de un ojo, se ha encontrado con un problema. Dicho problema consiste en que al crear una esfera y agregar una zona en ella, al cerrar y abrir Blender sin guardar la esfera no se volvía a crear al cargar el perfil, trataremos este problema en la sección de problemas. Por lo que se ha optado por crear una malla y guardarla, una vez guardado si se cierra la aplicación, se abre y se carga el perfil se resuelve el problema.

3.2. Creación de la malla de un ojo: Superficie ocular

Una vez resuelto dicho contratiempo ya se ha podido crear la malla a través de la aplicación Blender. Para ello se ha optado por seleccionar el espacio de trabajo Layout y en el modo editar se ha creado una UV Sphere la cual hará de ojo. Para hacerlo de la forma más realista posible en Viewport Shading, se ha seleccionado MatCap y le hemos puesto la textura de metal como se aprecia en la *Fig. 22. Creación de malla UV Sphere y aplicación de material metal*.

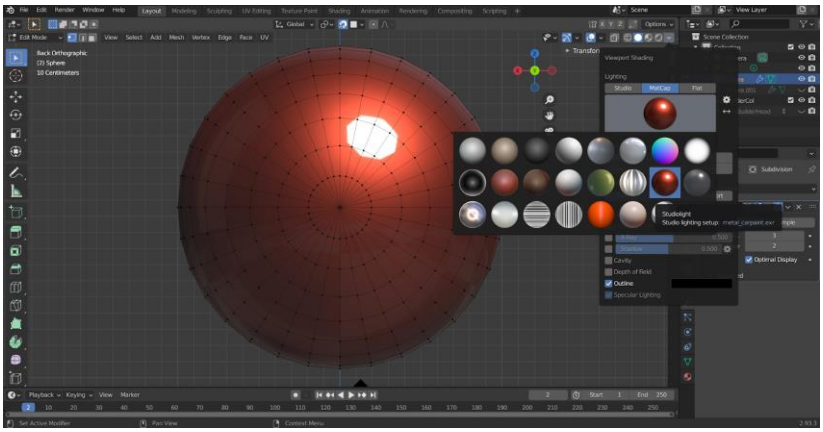

Fig. 22. Creación de malla UV Sphere y aplicación de material metal

Cuando creamos un UV Sphere, a través de la *Fig. 23. Vista de UV Sphere desde un polo* podemos apreciar como en los polos de dicha mesh quedan imperfecciones que no deberían de darse, ya que estamos tratando de simular un ojo humano.

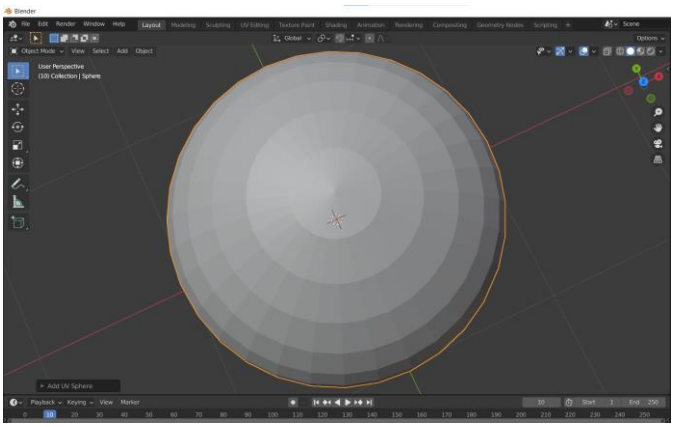

Fig. 23. Vista de UV Sphere desde un polo

Para mitigar este problema lo que haremos es rotar la mesh a través de los atajos de "Rotar al en el eje X 90º" , para ello mientras tenemos seleccionada la mesh, mantendremos pulsadas las teclas R + X [9] y además hemos de teclear un 9 y posteriormente un 0. Obteniendo así el resultado deseado plasmado en la *figura 24*.

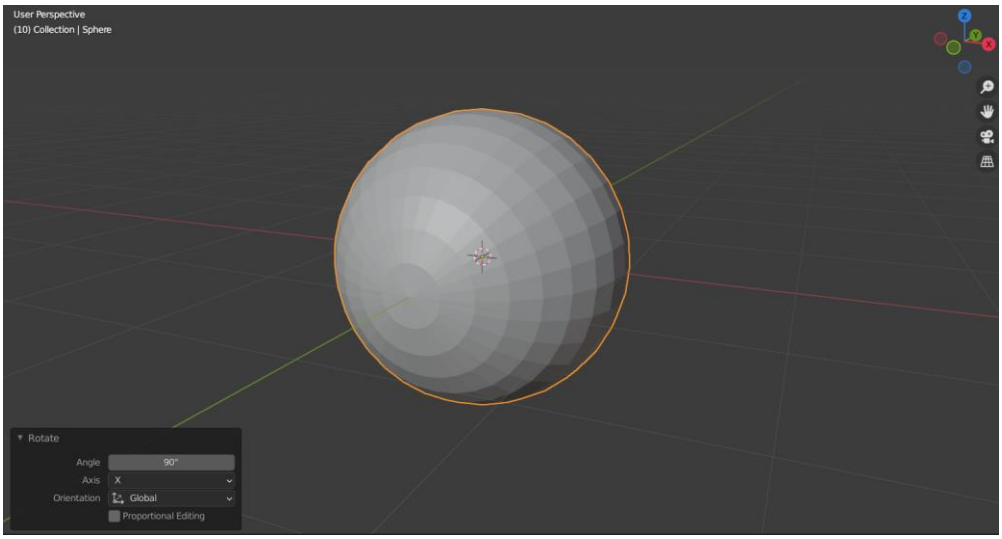

Fig. 24. Rotación para que polo quede alineado en el eje Y.

Como podemos apreciar en la figura veinticuatro el polo atraviesa la línea verde, eje Y, quedando perfectamente centrado. Una vez rotado debemos seleccionar el cursor 3D y marcar un eje central en la circunferencia de mayor radio, de las que vamos a deformar. Para ello haremos doble clic (en mi caso en la tercera) y con el atajo SHIFT + S [15] seleccionamos el cursor "to selected" como queda reflejado en la *figura 25*.

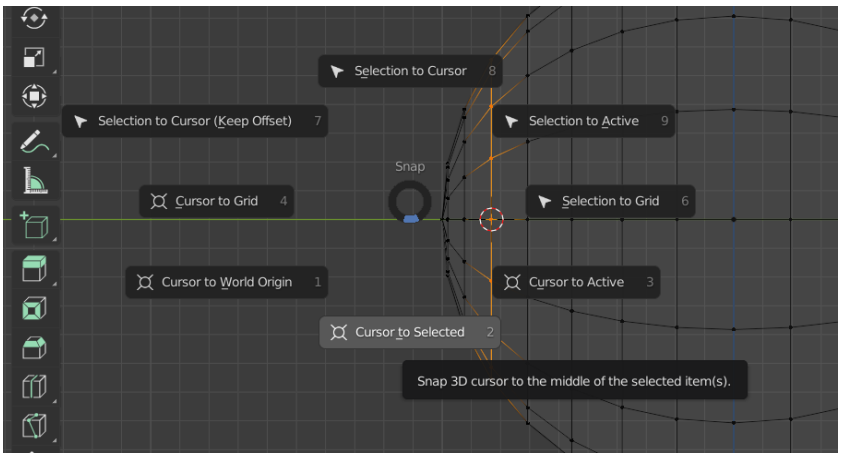

Fig. 25. Menú de atajo para selección de cursor.

Una vez rotada la mesh, nos iremos al modo de edición Wireframe, manteniendo a la Z y clicando sobre él, para poder seleccionar la parte frontal del ojo donde haremos la córnea. Mientras presionamos las teclas S + Y [10] teclearemos 1.5, para poder deformar dicha parte tal y como encontramos en la *Fig. 26. Modificación de polo de la mesh para simular superficie de córnea.*

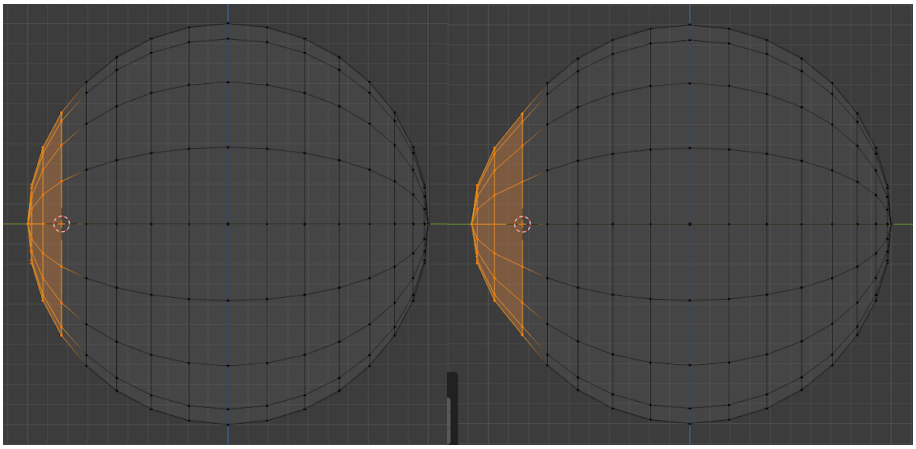

Fig. 26. Modificación de polo de la mesh para simular superficie de córnea.

Una vez hecho, vemos que la mesh de la parte izquierda de la *Fig. 27. Aplicación se Shade Smooth a la malla.* se aprecian los vértices por lo que para que parezca una superficie más lisa lo que haremos será seleccionarla y hacer clic derecho y a la opción "Shade smooth" [14], esto fuerza la asignación del atributo de "suavizado" a cada cara de la malla, incluso cuando agrega o elimina geometría.

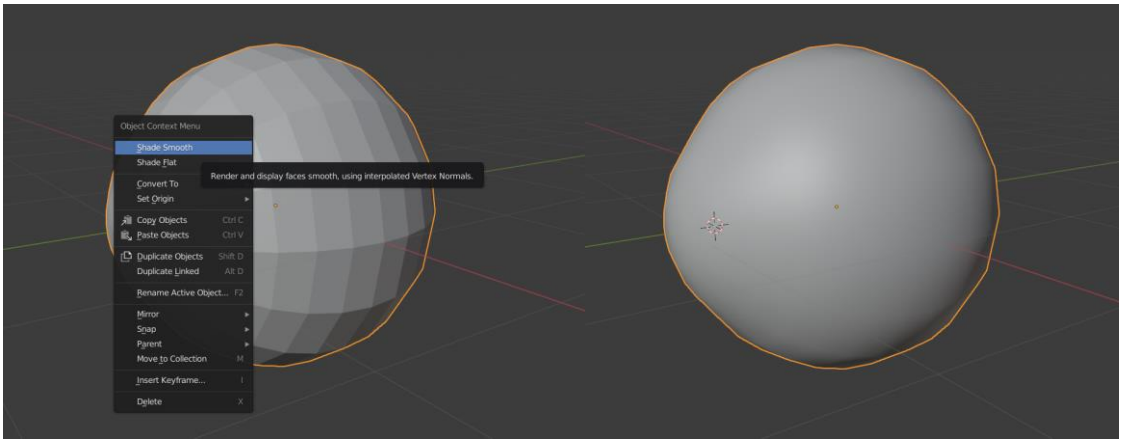

Fig. 27. Aplicación se Shade Smooth a la malla.

Y para más suavidad haremos CTRL + 3 [3], el cual subdivide, esto agregará el modificador "Subdivision Surface" con las subdivisiones de Viewport establecidas en 3 pudiéndolo ver reflejao en la *Fig. 28. Aplicación de creación de más vértices de la malla para suavizar*.

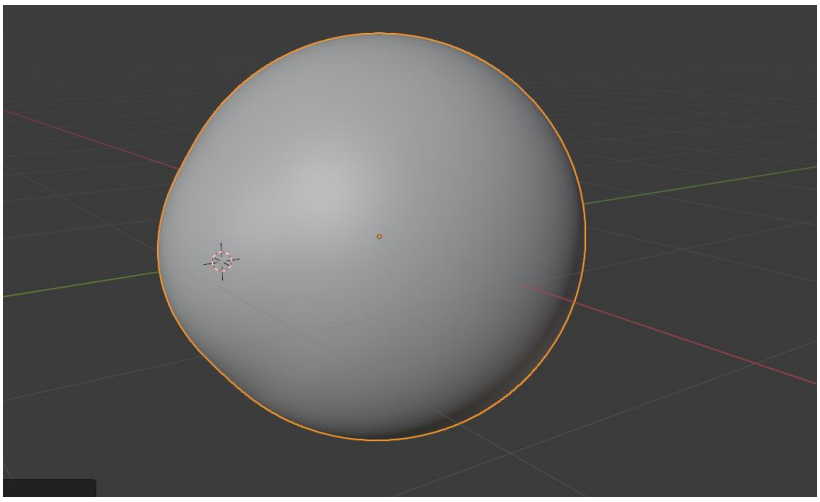

Fig. 28. Aplicación de creación de más vértices de la malla para suavizar

A continuación, como se observa en la *figura 29*, haremos zoom al polo de la esfera destinada para tener la parte frontal del ojo y juntaremos de dos en dos secciones con el atajo ALT + J [1] para poder unir los espacios, quedando más liso. Esto hará más fácil que al aplicar el material en esa parte que será la que contenga el iris ocular se pueda poner transparente con reflejo quedando el efecto de córnea que buscamos.

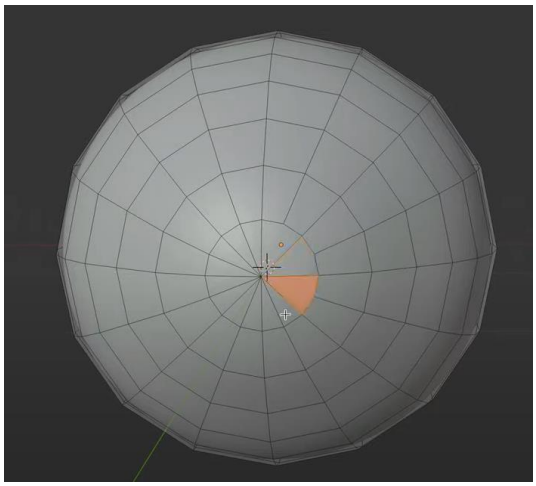

Fig. 29. Unión de facetas en la córnea

Después seleccionaremos los vértices que no tienen arista hacia el centro y pulsaremos la tecla G dos veces y 0.200 para obtener el resultado de la *Fig. 30. Flor poligonal de ocho puntas* como una especie de flor poligonal de ocho puntas.

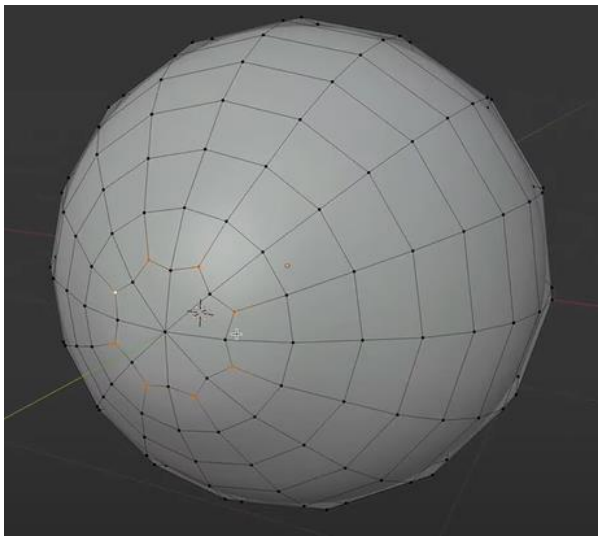

Fig. 30. Flor poligonal de ocho puntas

3.3 Creación de la malla de un ojo: Iris

Para crear el iris seleccionaremos la circunferencia hablada anteriormente de mayor radio de las que hemos seleccionado para deformar la mesh y crearemos una igual mediante SHIFT + D [12] como podemos apreciar en la *Figura 31*, esto nos permitirá crear dicha circunferencia para posteriormente agregar vértices quedando una "submalla" interna la cual nos servirá para aplicar el material del iris, pues la superficie del ojo será transparente, pero la malla interna tendrá aplicado el color del iris deseado

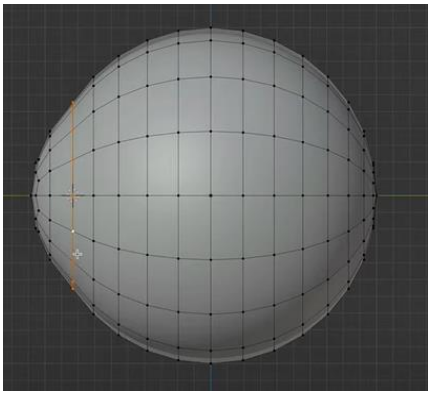

Fig. 31. Selección de arista y creación superpuesta de otra idéntica.

Siguiendo con el proceso teclearemos ALT + H, para obtener únicamente dicha circunferencia, la cual como se aprecia en la *Fig. 32. Demostración de creación de primera circunferencia interna de la "submalla"* uniremos con otra de menor radio seleccionando previamente el modo de objeto, punto medio. Si asumimos que cada elemento (objeto, cara, vértice, etc.), de la selección tiene la misma masa, el punto medio se asentará en el punto de equilibrio de la selección (el CDG [1]). Para crear dicha circunferencia menor lo haremos con E + S [6] y 0.854, y seguidamente le daremos profundidad con G + Y [7] y 0.049856, pues es queremos concatenar una serie de circunferencias internas para crear dicha "submalla" que hará la vez del iris del ojo.

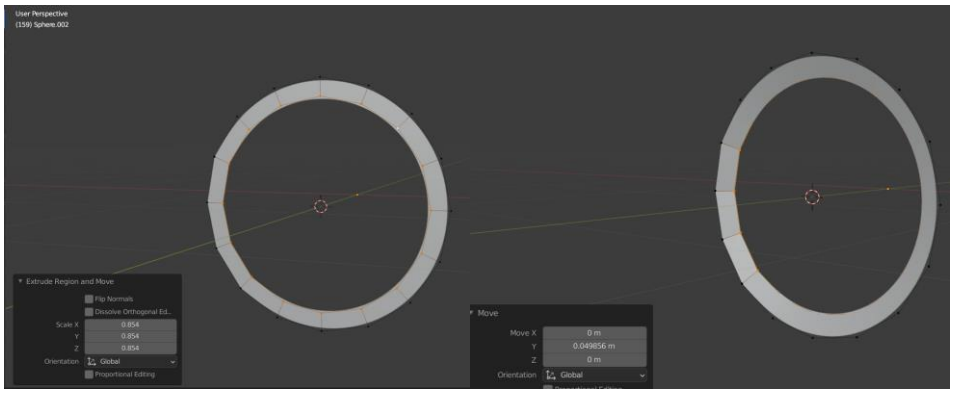

Fig. 32. Demostración de creación de primera circunferencia interna de la "submalla"

Del mismo modo como vemos en *Fig. 33. Creación de las tres circunferencias internas de la "submalla"* haremos otras tres circunferencias internas de 0.583, de 0.459 y de 0.412 respectivamente sin darles profundidad, ya que queremos crear esta malla interna para posteriormente que se vea reflejada en la superficie del ojo, más concretamente en la zona de la córnea.

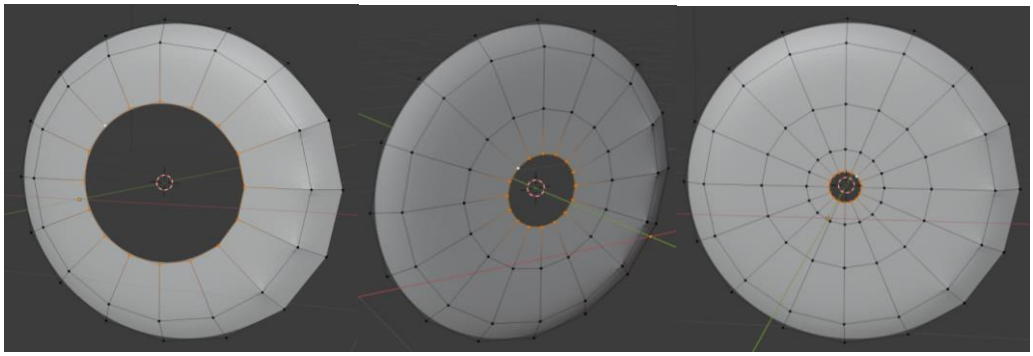

Fig. 33. Creación de las tres circunferencias internas de la "submalla"

Y por último uniremos la mesh creada, mediante “merge” que es la tecla M [8] y seleccionaremos la opción en el centro la cual colocará el vértice restante en el centro de la selección. Servirá como unión de los vértices de la circunferencia con menor radio para formar un origen central como el que se aprecia en la *Fig. 34. Unión de los vértices* , ya que es más correcto el cerrar esos vértices que crear más circunferencias de menor radio, pues, aunque sea un fallo de escala pequeña, las mesh interna no estaría cerrada completamente como apreciamos en la siguiente figura.

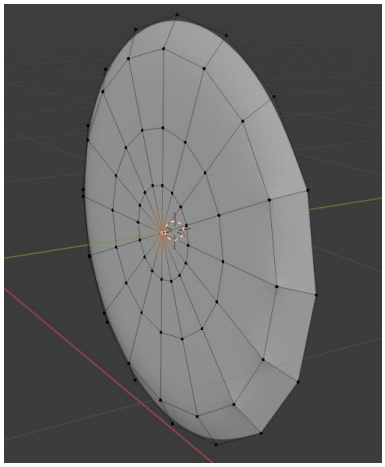

Fig. 34. Unión de los vértices.

Y al igual que hemos hecho anteriormente con el polo de la esfera juntaremos de dos en dos secciones con el atajo ALT + J, para ello seleccionaremos la opción “Face Select” y juntaremos las caras obteniendo un resultado como el que se aprecia en la *Fig. 35. Unión de facetas.*

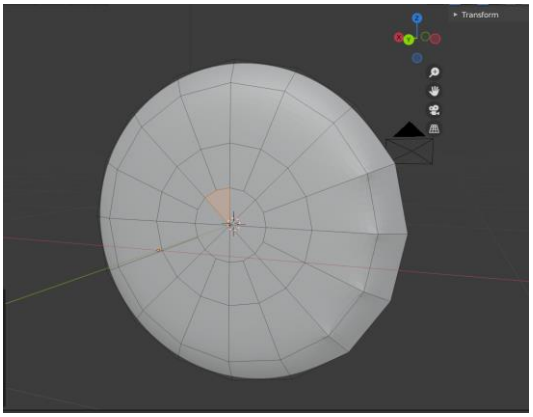

Fig. 35. Unión de facetas.

Para volver a la mesh que teníamos anteriormente volveremos a teclear ALT + H y poder ver el objeto de la escena en su totalidad, como podemos observar en la *Fig. 36. Resultado de la malla del ojo.* ya tenemos creada la mesh que formará tanto el ojo izquierdo como el derecho.

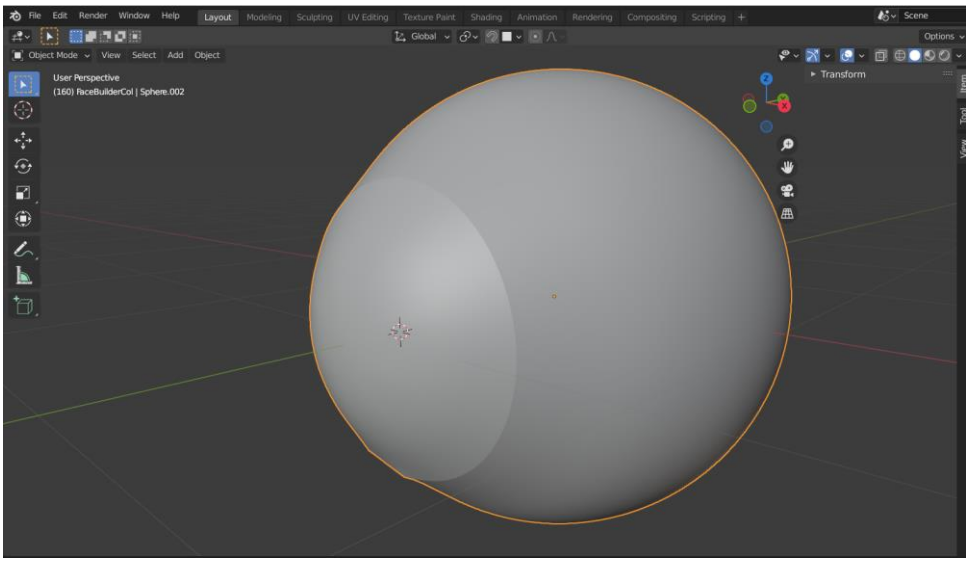

Fig. 36. Resultado de la malla del ojo.

3.4 Creación de los materiales.

A continuación, se va a explicar el proceso seguido para la creación de dos materiales que se aplicarán a la mesh obrada en el apartado anterior, uno para la superficie del ojo, los vértices que se aprecian desde el punto de vista sólido, es decir los que están en la parte externa de la malla y por otra parte el material del iris, el cual se aplicará en la "submalla" interna.

Antes de crear el material es necesario saber la orientación que tienen las facetas [4]. Para ello seccionaremos la opción "Face orientation", la cual como su propio nombre indica, nos informará mostrando en la consola los colores de las caras que forman la malla de color rojo o color azul. El color rojo indicará que la superficie apunta hacia adentro mientras que el azul indicará que las superficies apuntan hacia afuera. Este paso es importante para que cuando se aplique el material sobre la malla no se vea ningún

error, de lo contrario podría repercutir en la mal visualización del material en el ojo. Para ello primero estudiaremos la orientación de las facetas de la malla en su superficie, como podemos ver en la siguiente figura en la consola de Blender tendremos que seleccionar en el editor de vista tridimensional, más concretamente en el menú desplegable de superposiciones y deberemos tener activado la orientación de caras ("face orientation"). Como podemos ver en *Fig. 37. Resultado de aplicar "Face Orientation" en superficie* nos aparecen con las caras teñidas de azul, esto quiere decir que apuntan hacia afuera. Es importante tener en cuenta que para que esto se vea reflejado tendremos que estar en un método de display distinto a Wireframe, ya que en dicha vista no se aprecian las facetas de la malla.

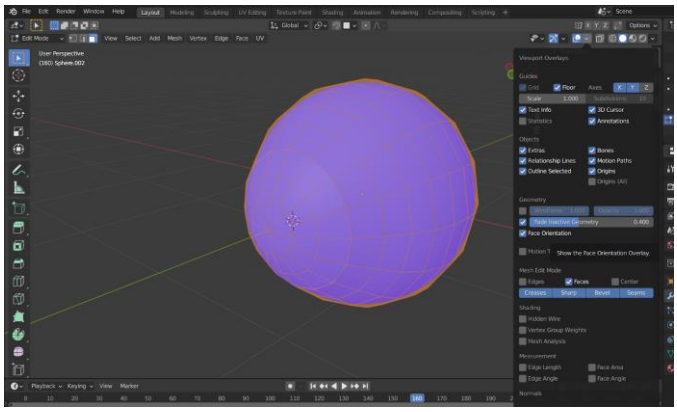

Fig. 37. Resultado de aplicar "Face Orientation" en superficie

Una vez investigado la orientación de la superficie de la malla tendremos que ver la "submalla" que será en la que se refleje el material del iris que vamos a crear posteriormente, es por ello que observaremos la malla interna a parte de la superficie, para poder apreciar bien las orientaciones de las caras, para seleccionar la mesh interna usaremos el atajo SHIFT + H [13] tal y como se plasma en la siguiente figura (*figura 38*).

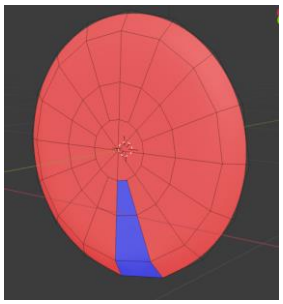

Fig. 38. Resultado de aplicar "Face Orientation" en "submalla"

A través de la vista de malla Solid, en la *Fig. 39. Resultado de aplicar SHIFT + N en la "submalla"* comprobamos que hay tres caras teñidas de azul; es decir, las facetas apuntan hacia afuera teniendo una orientación opuesta. Para corregir dicho problema, las seleccionaremos y usaremos el atajo de SHIFT + N [14], para cambiar hacia donde apuntan.

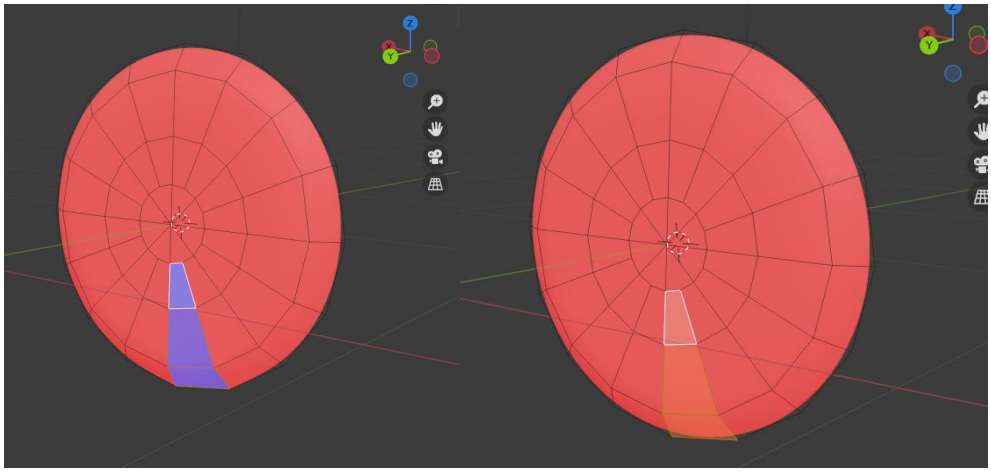

Fig. 39. Resultado de aplicar SHIFT + N en la "submalla"

Antes de determinar el material del ojo, tendremos que activar unos Add-ons esenciales para que se aprecie bien el material que vamos a crear. Para ello tenemos que ir al menú de edición, preferencias, Add-ons y activaremos 3D View: Stored Views y además Node Wrangler tal y como vemos en la *Fig. 40. Activando Add ons.*

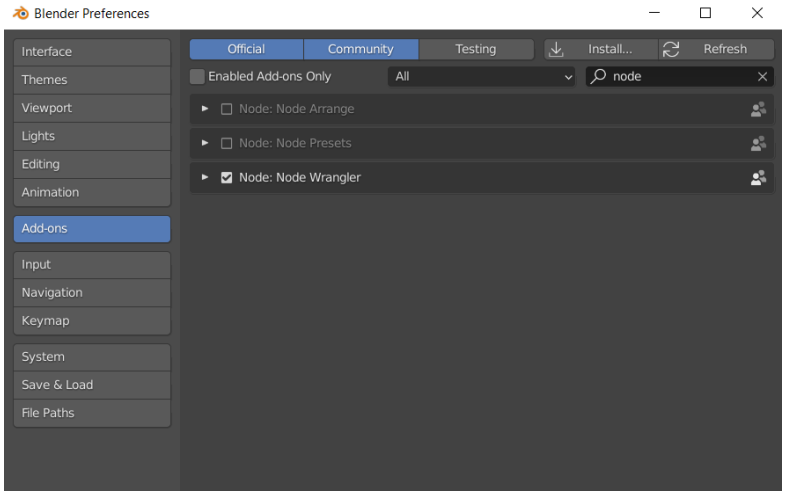

Fig. 40. Activando Add ons.

3.5 Creación material: Superficie del ojo

Para la textura del ojo nos iremos al workspace de Shading y añadiremos dos nuevas ventanas de materiales tal y como vemos en la *Fig. 41. Creación de slots de material.* , una para el material de la superficie del ojo y otro para el del iris.

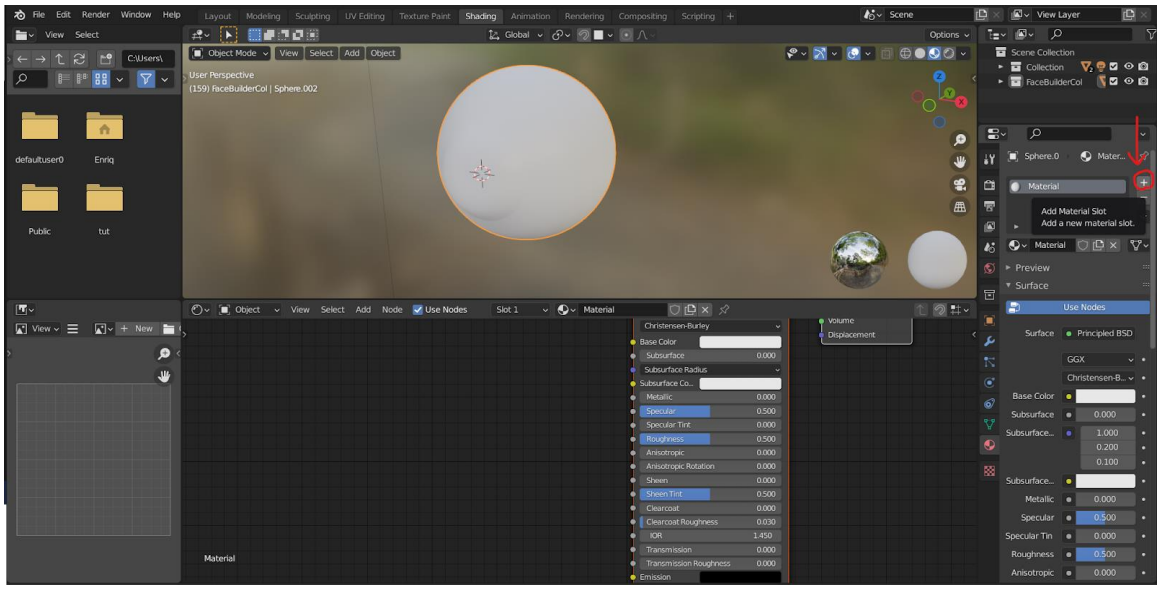

Fig. 41. Creación de slots de material.

Para ver como influyen sobre la mesh los materiales que vamos a ir creando es importante aplicarlos desde un principio. Con la creación del primer material en este caso "Superficie Ojo" por defecto se aplica automáticamente en todos los vértices, facetas o bordes de la mesh. Es por ello por lo que el material del iris lo tendremos que asignar a la "submalla" creada en el anterior apartado. Para ello seleccionaremos los vértices superficiales a través de una vista distinta a Wireframe y posteriormente con el atajo de invertir selección CTRL + I [2] podremos tener marcados los vértices a los que queremos aplicar el material, a través del modo Assign en modo de edición como vemos en la *Fig. 42. Asignación de material iris a la "submalla"*. Para la creación de materiales nos situaremos en el workspace de Shading, el cual dentro de un material se trabaja con nodos, hay una amplia gama de nodos los cuales cada uno tiene una función específica, como puede ser dar un gradiente de color a la superficie, otra dar efecto

metálico a la superficie, otra añadir ruido para que parezca una superficie rugosa y muchos más. En función de cómo enlacemos los nodos y los valores que demos a las propiedades individuales de cada uno obtendremos un resultado u otro.

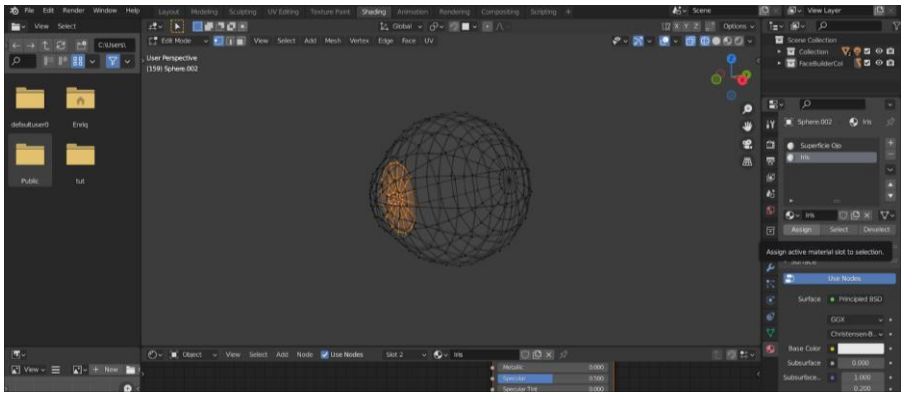

Fig. 42. Asignación de material iris a la "submalla"

Para que funcione correctamente la reflexión [12] y la refracción [13] tendremos que ir al apartado de propiedades de renderizado y seleccionar la pestaña de Screen Space Reflection. Además, debemos de aplicarlo en las propiedades de material haciendo clic en la pestaña de Screen Space Refraction como se plasma en la *figura 43*.

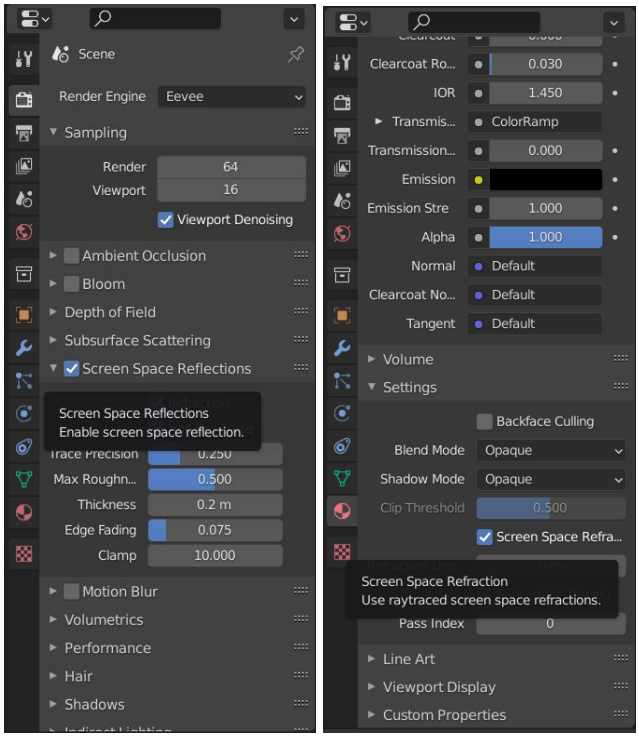

Fig. 43. Editor propiedades de renderizado y de material.

Esto nos permitirá poder ver el color del Iris, como se aprecia en la *figura 44* el color es blanco, el cual se encuentra en otro slot de material. Además, para poder simular la transparencia del ojo tendremos que en configuración de la superficie del ojo cambiar el modo blend [8] y el modo sombra a Alpha Clip.

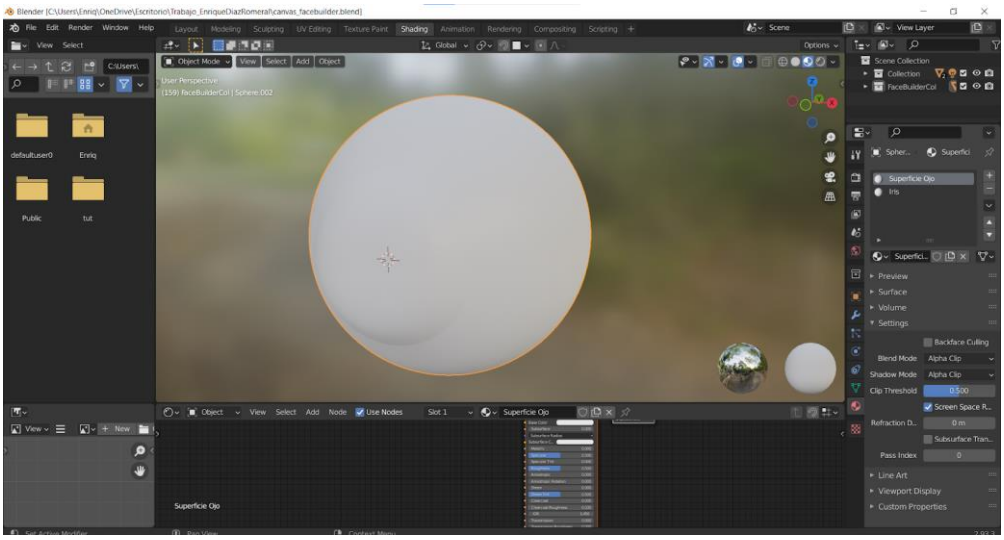

Fig. 44. Blend Mode a Alpha Clip

Iremos a las propiedades BSDF del espacop de trabajo Shading, en la ventana de los nodos que se encuentra en la parte inferior y en Transmission pondremos el valor a 1.0. Para quitar ese efecto de ventana con efecto de condensación, en Roughness lo bajaremos de 0.5 a 0.05 obteniendo un resultado óptimo como el reflejado en *Fig. 45. Blend Mode a Alpha Clip.*

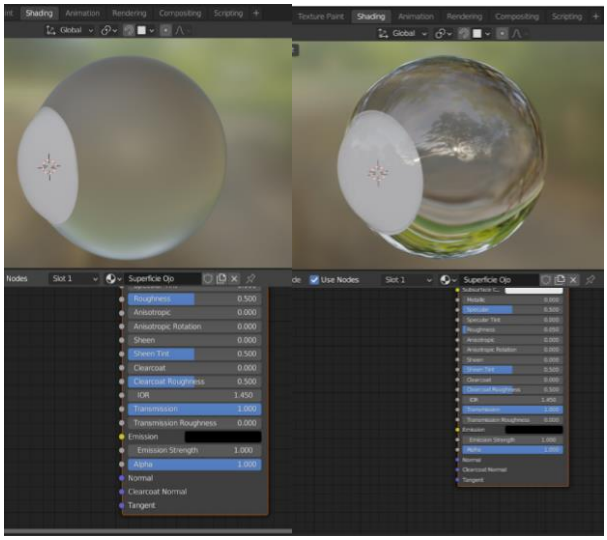

Fig. 45. Blend Mode a Alpha Clip

A continuación, añadiremos un gradiente de textura con el comando Shift + A [11]. linkeamos del nodo coordenadas de texturas el objeto con el vector del nodo Mapping, y como gradiente de textura de linear pasará a esférico. Atajo representado en *Figura 46.*

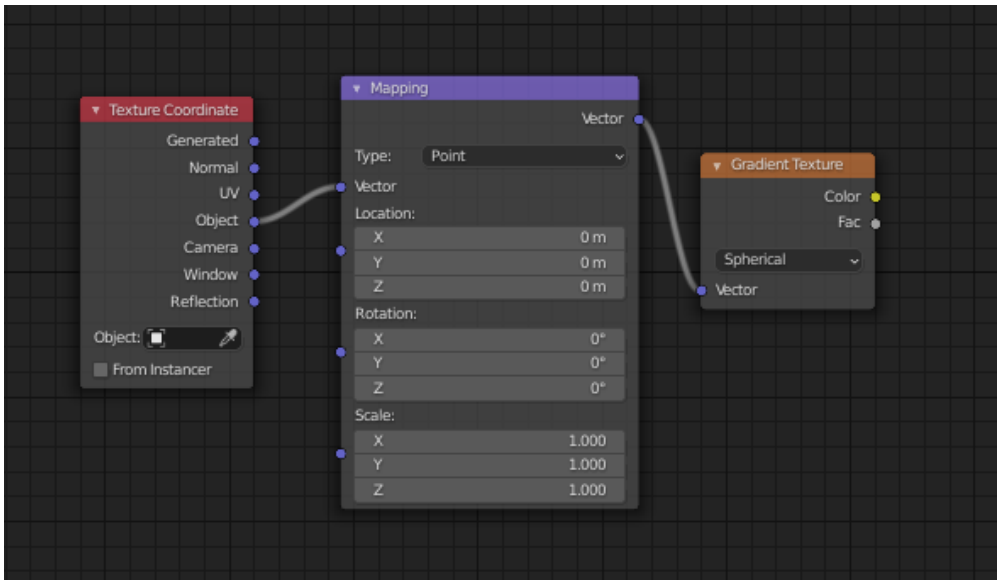

Fig. 46. Enlace entre nodos de Superficie Ojo (1)

Una vez aplicada dicha configuración en los nodos crearemos del mismo modo un nodo ColorRamp y en Color le daremos Click Izquierdo sobre color mientras mantenemos CTRL + SHIFT, creando un viewer linkeado al nodo Material Salida, tal como el de la *Fig. 47. Enlace entre nodos de Superficie Ojo (2).*

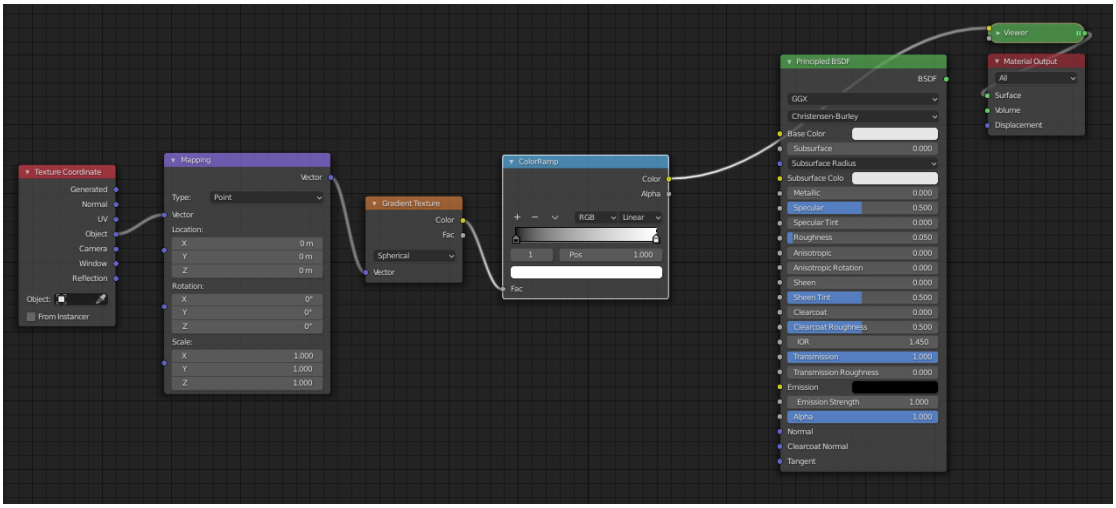

Fig. 47. Enlace entre nodos de Superficie Ojo (2)

Ajustaremos el Color Ramp cambiando el color blanco a la posición 0.607 y el color negro a la 0.579, esto generará un degradado de blanco a negro, además de en el nodo Mapping el Location de Y será de 0.8, T la escala de X, Y y Z será de 0.5 , para poder aplicar el blanco en la zona de la córnea del ojo como se aprecia en la *Fig. 48. ColorRamp, Mapping y Location sobre superficie ocular (1)* . Donde posteriormente esa parte será transparente para permitir ver el color del iris.

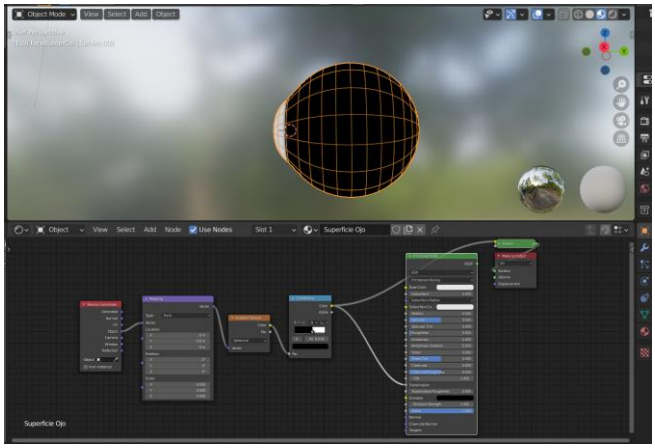

Fig. 48. ColorRamp, Mapping y Location sobre superficie ocular (1)

Sobre el nodo principed BSDF haremos CTRL + SHIFT + Clic Izquierdo y linkearemos[5] Color del nodo ColorRamp a Transmission de Principled BSDF, para poder cambiar el Material Output, pues antes estaba en el nodo ColorRamp, por lo que se veía sólo blanco y negro, si la salida de dicho nodo lo cambiamos a que apunte a la entrada de Transmission de Principled BSDF, para que se aplique como transparencia en la malla como se aprecia en la *figura 49.*

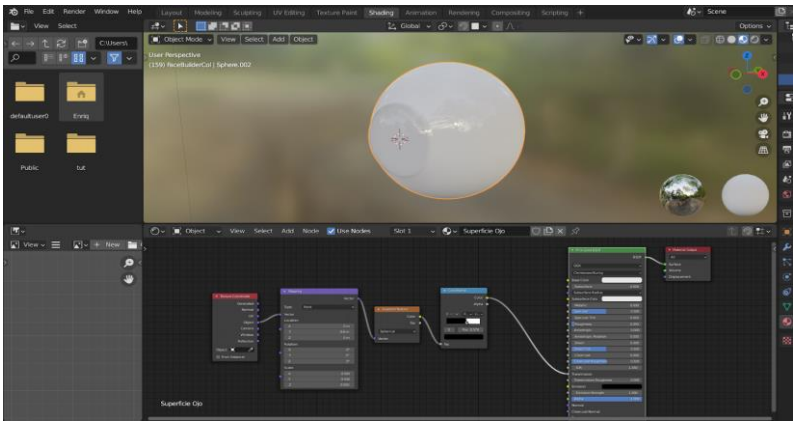

Fig. 49. ColorRamp, Mapping y Location sobre superficie ocular (2)

A continuación, seleccionaremos puntualmente al material del Iris aunque estemos en la sección de cómo se ha creado la superficie del ojo, pues como ya se ha dicho anteriormente, se ha aplicado los nodos anteriores para obtener la parte transparente que caracteriza a la córnea y para ver que realmente podemos ver el iris pondremos un color temporal al iris como es el azul base que vemos en la *figura 50*.

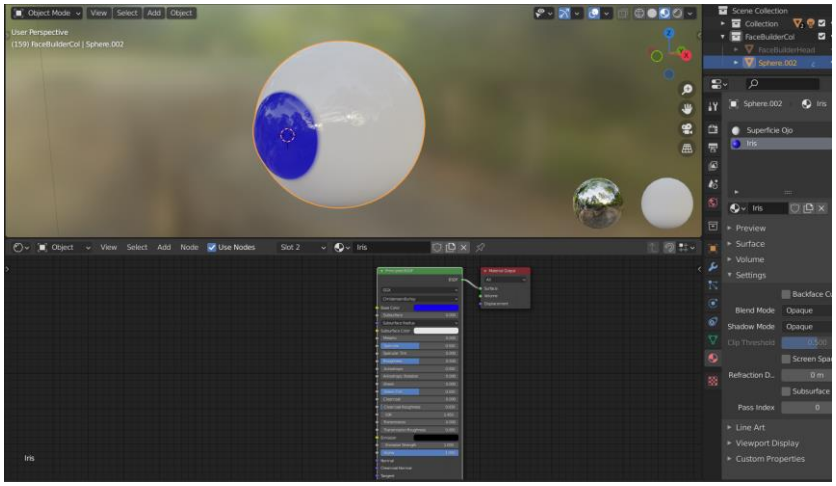

Fig. 50. Color azul temporal en material Iris

A continuación, volveremos nuevamente al material superficie del ojo para simular los vasos sanguíneos. Partiendo de que en la zona trasera del ojo hay más, lo que haremos será crear un nodo ColorRamp el cual lo vamos a vincular con el color base del material del nodo Principed BSDF como se refleja en *Fig. 51. Enlace entre nodos de Superficie Ojo (3)*.

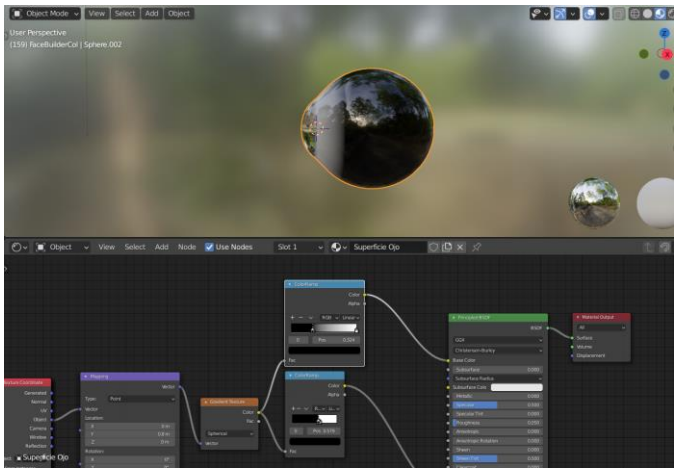

Fig. 51. Enlace entre nodos de Superficie Ojo (3)

Como podemos apreciar en la figura anterior, aún no tenemos un resultado parecido a lo que deseamos, para ello lo que haremos será graduar que el color llegue más o menos a la mitad del ojo y cambiaremos el color negro a uno rojo ayudándonos de otro tercer color rojo más claro, dando una degradación de color. Creando así un degradado de rojos para simular un color más realista y que no sea únicamente un color plano como se ve en la *figura 52*. En la figura siguiente vemos como repercute en la salida de la malla aplicando el primer color y aplicando un color auxiliar entre los dos anteriores para crear ese efecto de degradado de color buscado.

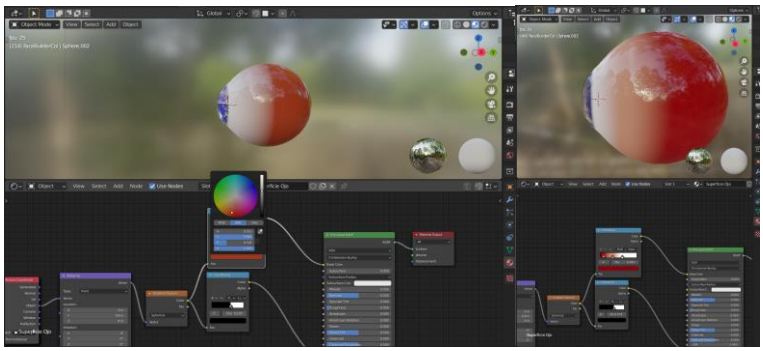

Fig. 52. Degradados de color en los vasos sanguíneos del ojo.

Una vez que obtenemos el color deseado debemos determinar los vasos sanguíneos a través de otro nodo del tipo textura el cual hace alusión al diagrama de Voronoi [2]. Para ello con SHIFT + A buscaremos en la sección de texturas la textura Voronoi. Sobre el nodo haremos como anteriormente CTRL + T [5] para linkearlo con Texture Coordinate y Mapping asociado a Voronoi. Crearemos un nodo ColorRamp sustituyendo el color negro por el color rojo que teníamos anteriormente. Sobre el nodo ColorRamp daremos al atajo CTRL + SHIFT + CLIC DERECHO [4] viendo cómo influye en el renderizado de la mesh en la *Fig. 53. Aplicando ruido Voronoi.*.

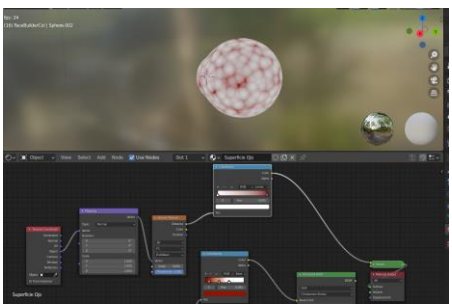

Fig. 53. Aplicando ruido Voronoi.

Crearemos el nodo Frame, tipo Layout, y en él meteremos el nodo Mapping, Voronoi Texture y ColorRamp que hemos creado en esta última tanda. El nodo Frame solamente sirve para que a la hora de ver en la aplicación Blender el resultado final de todos los nodos creados y sus enlaces quede de una forma más ordenada posibilitando la sencillez y el entendimiento. Y así lo copiamos dos veces más y linkeamos el objeto de Texture a cada vector del Mapping correspondiente, donde cada Frame hace alusión a los vasos sanguíneos pequeños, medianos y largos. A continuación, copiaremos dos veces más el Frame con los nodos que contiene pues con ellos vamos a dar tamaños distintos a los vasos sanguíneos tal y como se refleja en la *Fig. 54. Disposición de los nodos Frame y sus contenidos.*

- **Frame L:** Se aplicarán unos valores a las propiedades de los nodos que contiene para crear vasos sanguíneos de una longitud larga.

- **Frame M:** Se aplicarán unos valores a las propiedades de los nodos que contiene para crear vasos sanguíneos de una longitud media.

- **Frame S:** Se aplicarán unos valores a las propiedades de los nodos que contiene para crear vasos sanguíneos de una longitud corta.

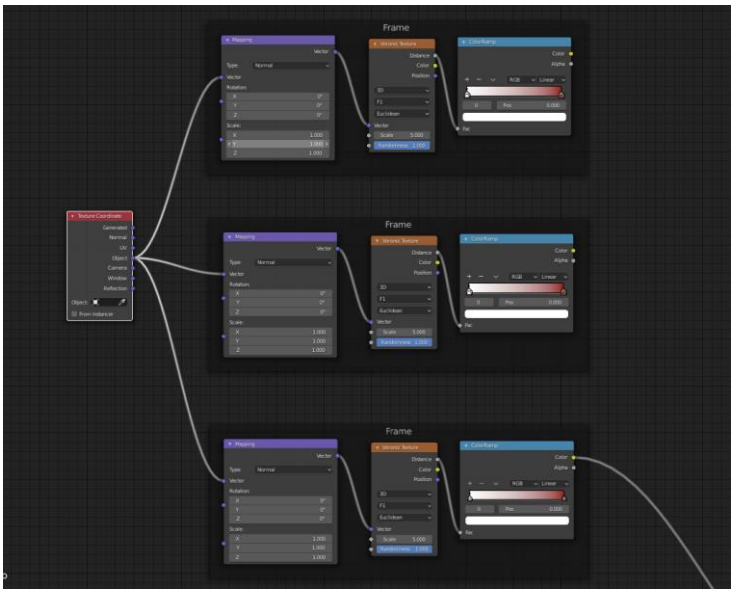

Fig. 54. Disposición de los nodos Frame y sus contenidos.

Comenzaremos con los largos (Frame L), para ello la escala del vector que estamos generando necesitamos que en el eje Y, que indica la largura, deberá de posicionarse a 4, para deformarse y dar así una especie de venas estiradas. Y la Escala del nodo del ruido Voronoi la actualizaremos a 3 para dar un efecto más realista como se plasma en la *figura 55*.

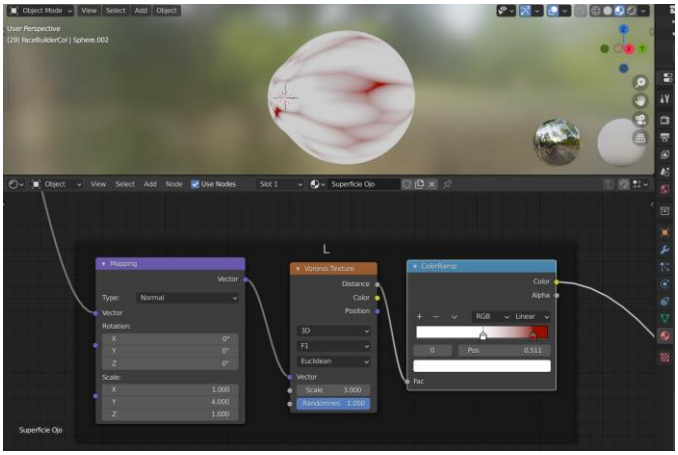

Fig. 55. Asignando valores a Frame L

Seguiremos con los vasos sanguíneos medianos, Frame M, para ello sobre este ColorRamp haremos CTRL+ SHIFT +CLIC DERECHO, esto hará que sobre la superficie sólo podamos ver cómo influye el frame actual que estamos modificando, pero al final los enlazaremos para que se vean todos superpuestos. Cambiaremos el parámetro de Y en el nodo Mapping a un valor 2, en el nodo de la textura de Voronoi le daremos una escala de 6 tal y como se refleja en la *figura 56*.

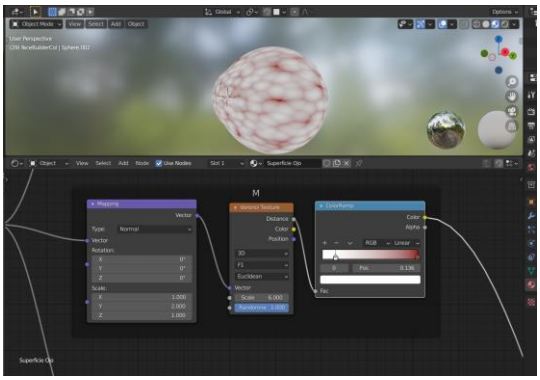

Fig. 56. Asignando valores a Frame M

Seguidamente iremos al nodo Frame S de los vasos sanguíneos pequeños, donde en el nodo Mapping Y ahora tomará un valor de 2 y la textura de Voronoi una escala de 20, viéndose unos vasos sanguíneos más abundantes y de menor tamaño, como en la *Fig. 57. Asignando valores a Frame S*

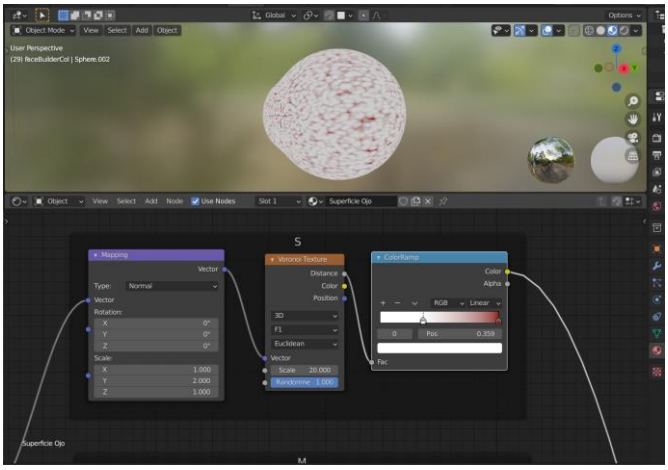

Fig. 57. Asignando valores a Frame S

Para poder aplicar a la superficie las tres casuísticas de Voronoi simulando los vasos sanguíneos usaremos el atajo CTRL + SHIFT + CLIC DERECHO del ColorRamp del frame de los vasos sanguíneos medianos a los largos, obteniendo este nodo donde seleccionaremos en el combo box de Mix a Darken, y para unir los vasos sanguíneos más pequeños procedemos a hacer el mismo atajo anterior del ColorRamp del frame S al nodo auxiliar que ha surgido antes pudiendo observa el resultado en la *Fig. 58. Uso del nodo Mix para juntas colores de vasos sanguíneos.*. Obteniendo otro nodo más y le pondremos también Darken.

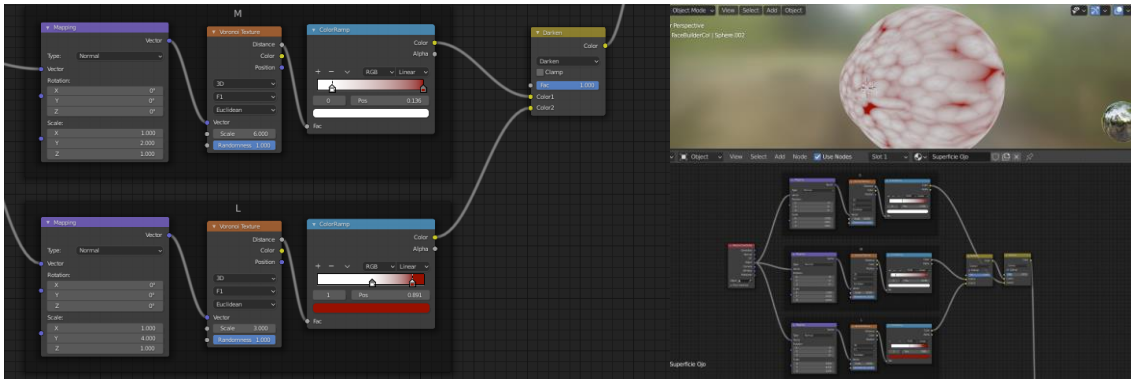

Fig. 58. Uso del nodo Mix para juntas colores de vasos sanguíneos.

Y para terminar haremos el mismo atajo, pero esta vez con el nodo ColorRamp principal con este nodo auxiliar fina y el color lo enlazaremos con el color base de la superficie del nodo principal, obteniendo un resultado en la malla bastante más cercano al deseado como se observa en la *Fig. 59. Uso del nodo Mix de los nodos Mix.*.

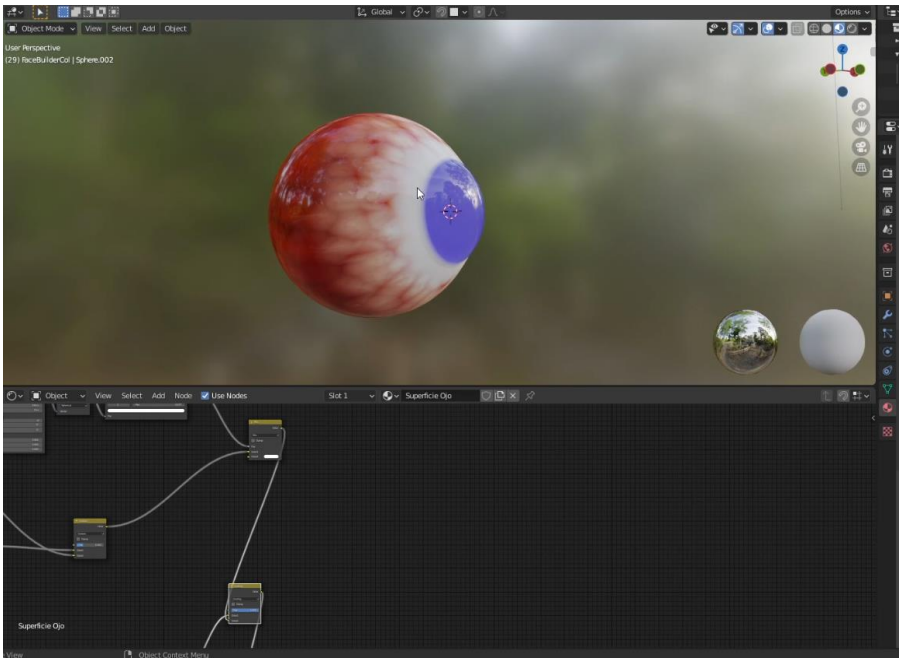

Fig. 59. Uso del nodo Mix de los nodos Mix.

Como se puede observar en la imagen los vasos sanguíneos que se acercan a la pupila tienen una forma con más curvatura; para ello necesitaremos añadir ruido en el frame de los vasos sanguíneos largos y una luz lineal con los nodos textura de ruido y luz linear (Mix RGB) y lo enlazaremos con el vector de Mapping de los vasos sanguíneos largos tal y como se demuestra en la *Fig. 60. Fotografía de ojo*.

Fig. 60. Fotografía de ojo

Para finalizar con el material que se está aplicando a la superficie del ojo añadiremos sobre el tercer nodo de mezcla de color (Color Mix) un nodo ColorRamp y a éste el nodo de textura Bump. Este nodo dará una rugosidad y un reflejo en la superficie del ojo sobre los vasos sanguíneos con un efecto de profundidad de las venas. Obteniendo como resultado del renderizado y un mapa conceptual de los nodos como el reflejado en la siguiente *figura 61*.

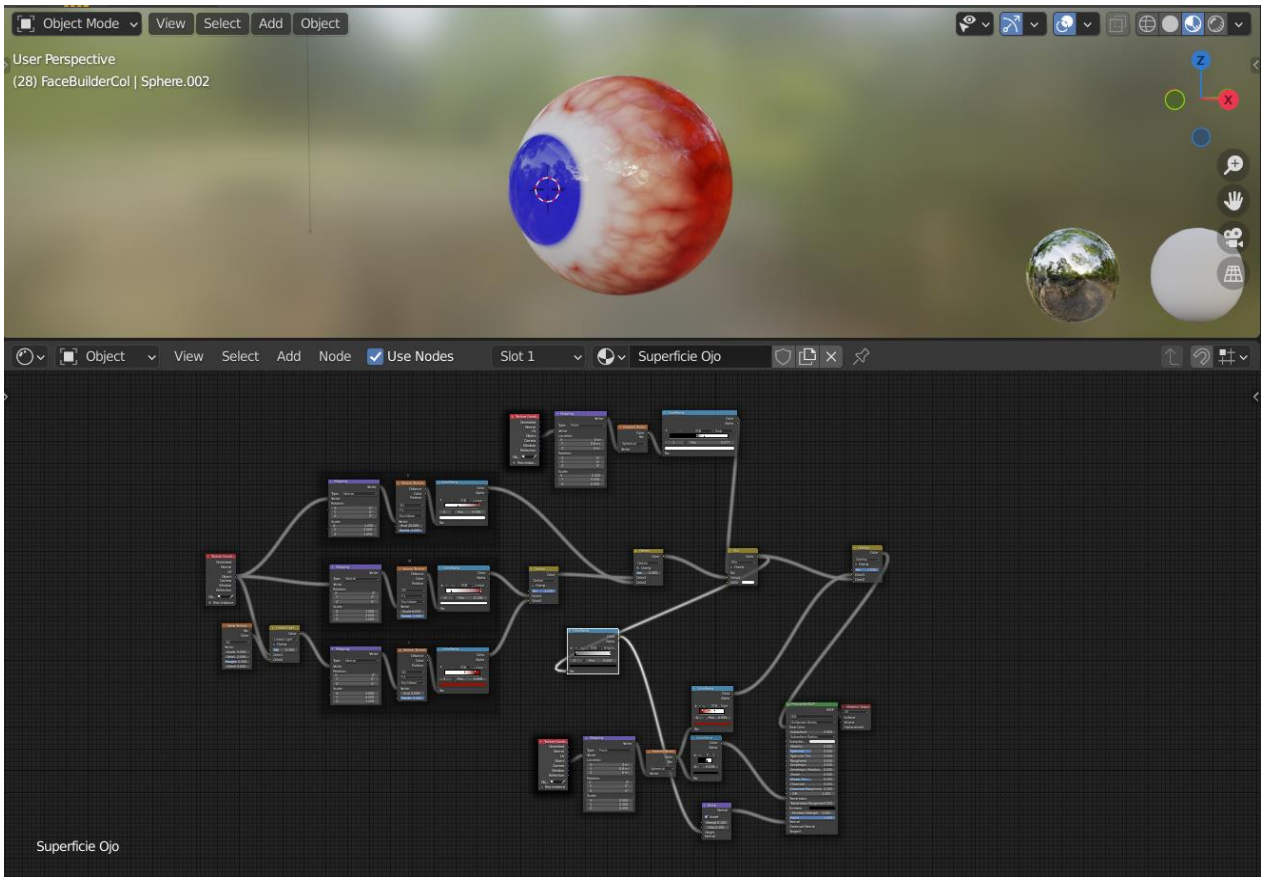

Fig. 61. Renderizado y un mapa conceptual de los nodos del material Superficie Ojo

3.6 Creación material: Iris.

A continuación, dentro del espacio de trabajo de Shading, nos iremos al editor de propiedades. Dentro de la sección de materiales, icono de una esfera tridimensional, el penúltimo seleccionable de la lista (*Fig. 62. Editor de propiedad de materiales*), podremos cambiar de material de Superficie Ojo al material Iris. Como ya hemos asignado los vértices a los que queremos que se aplique dicho material no haría falta volverlo a hacer.

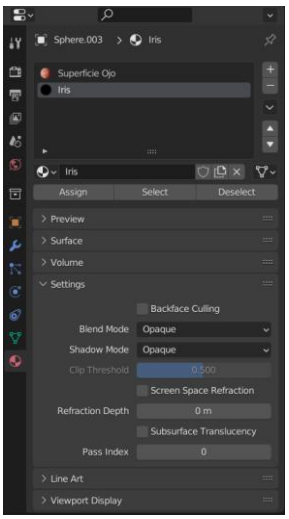

Fig. 62. Editor de propiedad de materiales

Es importante destacar que el primer material creado se aplica en la totalidad de la malla, para poder asignar materiales creados posteriormente será necesario entrar en el modo de edición y automáticamente te aparecerá un botón de deseleccionar, seleccionar y asignar.

- **Asignar:** Aplica el material sobre los vértices, las aristas o las caras, dependiendo en que vista de edición se seleccione, que se hayan seleccionado previamente.

- **Seleccionar:** Es necesario que se haya asignado previamente el material, esto marcará los vértices que están involucrados directamente en la aplicación del material sobre la mesh.

- **Deseleccionar:** Para ver su comportamiento es necesario haber dado previamente al botón de Seleccionar, pues esto lo que hará es dejar de mostrar los vértices involucrados en el material.

Siguiendo en la ventana de nodos crearemos el nodo Gradiente de Textura, la determinaremos como esférica y para crear su respectivo Mapping y su Texture Coordinate a través del atajo de CTRL + T. Sobre la localización del Mapping en el eje Y le daremos un valor de 0.65 quedando tal y como se refleja en la *figura 63.*

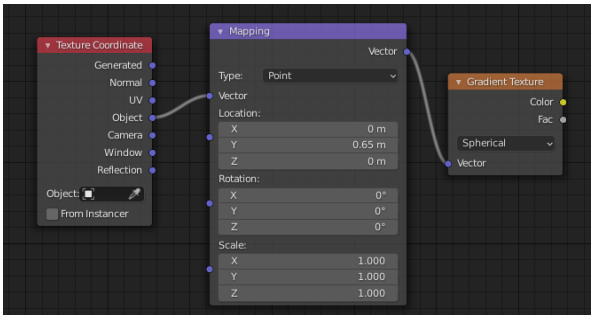

Fig. 63. Enlace entre nodos Iris

Para determinar el color del iris sobre este gradiente de textura se añade un nodo ColorRamp. En este primer caso se optará por hacer el color del iris azul, para ello ajustaremos los nodos de colores negros como extremos y añadiremos un color azul claro entre medias de ellos dos y crear dos colores auxiliares a partir de ese mismo azul y tantear los colores para que quede un degradado de color parecido al del ojo humano. Obteniendo el color general de las partes del iris del centro a la periferia como en la *figura 64*.

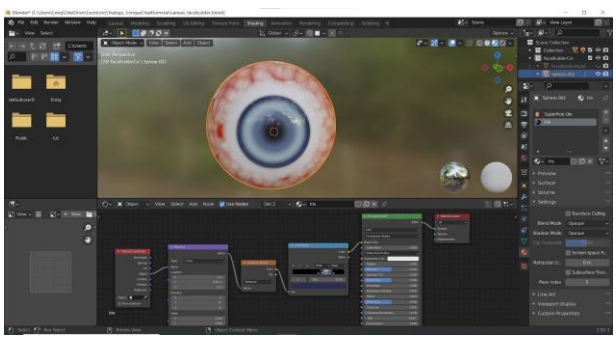

Fig. 64. Añadiendo colores de color Iris con Color Ramp

Una vez teniendo la base del iris ahora debemos de obtener esa textura tan característica que tiene, la cual como curiosidad permite realizar el reconocimiento ocular a través del sistema biométrico, crearemos el nodo de Textura con Ruido y haremos el atajo CTRL + T para crear los nodos que controlen su textura de coordenadas y su Mapping. Enlazaremos Fac del nodo de textura a otro nodo ColorRamp creado para determinar dicho ruido y veremos cómo repercute solamente el ruido. Además, asignaremos a este color dos azules parecidos a los que habíamos puesto anteriormente para poder simular lo dicho y al igual que se ha hecho en los vasos sanguíneos podremos retocar la escala del nodo Mapping para que quede una forma más realista. Además de cambiar el tipo del nodo Mapping a normal obteniendo así una aproximación más cercana a la realidad como la que podemos apreciar en la *figura 65*, como vemos aún le queda unos pasos.

Fig. 65. Enlace entre nodos Iris (2)

Con el atajo CTRL + SHIFT + CLICK DERECHO arrastramos de un ColorRamp a otro, obteniendo la mezcla de ellos dos, la cual pondremos en modo Overlay para superponerlos, el Color 1 corresponderá al primer ColorRamp creado, mientras que el 2 al segundo, obteniendo así el resultado de la *Fig. 66. Enlace entre nodos Iris (3)*.

Fig. 66. Enlace entre nodos Iris (3)

Como remate para la parte del ojo se creará un tercer nodo de ColorRamp el cual meteremos el color de la mezcla de los nodos ColorRamp anteriores lo meteremos en Fac de este nuevo, el color obtenido lo sacaremos al peso del nodo Bump que crearemos. Bajaremos la fuerza y la distancia para que no sea muy abrupto y lo linkearemos con la variable "Normal" del nodo principal del iris Principled BSDF, obteniendo el resultado de la disposición de los nodos reflejado en la *Fig. 67. Enlace entre nodos Iris definitivo*.

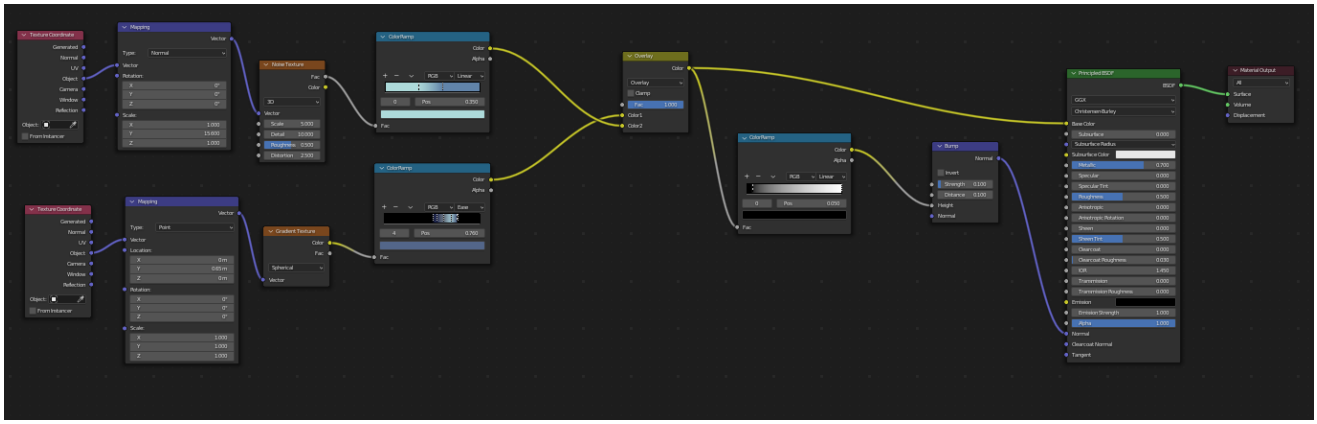

Fig. 67. Enlace entre nodos Iris definitivo

4. Resultado de aplicar los materiales de "Superficie Ojo" y de "Iris".

En este apartado es breve pero necesario para percibir todos los detalles que tiene el enlazar los nodos que hemos visto previamente y los valores que hemos dado a las propiedades de estos, los cuales percibimos en la *figura 68* de manera frontal *y* en la *figura 69* de manera lateral.

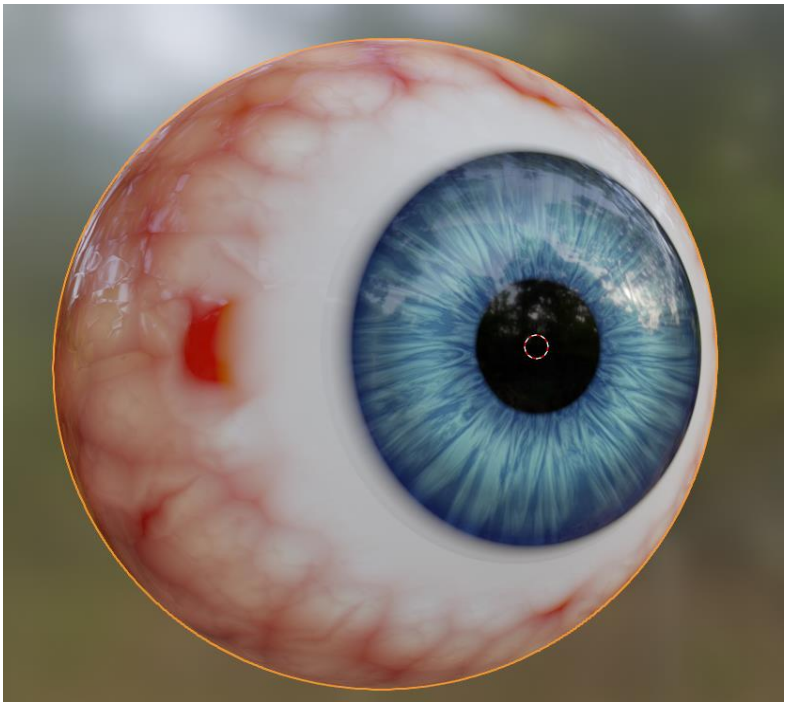

Fig. 68. Ojo vista frontal

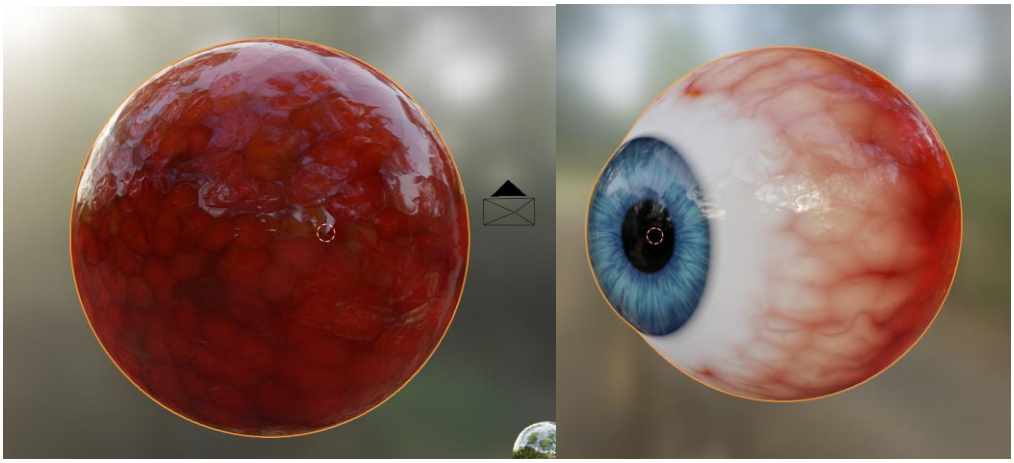

Fig. 69. Ojo vista trasera y lateral

5. Scripting material para los ojos

Una vez adquirido el conocimiento necesario para poder haber realizado el ojo a través de la aplicación de Blender se puede abstraer dichos conocimientos para poderlos aplicar a Scripting de Python. Clasificaremos la forma en la que se desarrolla esta parte de "traducir" a código lo aplicado en el espacio de trabajo Shading a través de Blender en tres partes:

1. Determinar los índices de los vértices de la mesh que serán para la superficie del ojo.
2. Seleccionar los índices de los materiales "Superficie del ojo" y de "Iris" a través de Python.
3. Creación de los materiales "Superficie del ojo" e "Iris" y asignar materiales a los vértices estudiados en los puntos anteriores.

5.1 Determinar los índices de los vértices de la mesh.

Para ello crearemos a través del Scripting que nos ofrece Python, un menú básico con un botón el cual cuando se clique en él nos imprimirá en la consola de Blender los índices de los vértices seleccionados. El fichero se ha reflejado en el apartado de los scripts involucrados *11.1 "Imprimir_vertices_seleccionados.py"*

En la sección ***class ADDONNAME_PT_main_panel(Panel)**:* la cual podemos apreciar en el código inferior es la parte en la cual se definirá el nombre del panel, y cómo se verá reflejado en la vista de la aplicación.

La parte bl_space_type = 'VIEW_3D' hace referencia a que el editor del panel aparecerá en la consola en la vista 3D para aplicarlo por interfaz sería como en la *figura 70*, la cual si cambiamos por otro tipo de vista no se verá reflejado en esta.

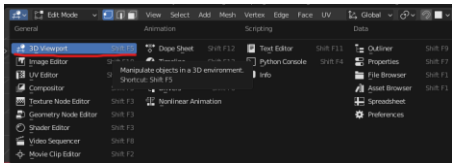

Fig. 70. Editor 3D Viewport

En la sección **class ADDONNAME_OT_my_op(Operator):** es la parte en la cual se refleja el botón y el nombre que tendrá, como se refleja en *Fig. 71. Panel UI con botón.*

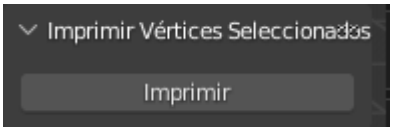

Fig. 71. Panel UI con botón

Dentro de esta misma clase **def execute(self, context):** es la parte del código en la cual se determina cómo actuará el script al clicar sobre dicho botón, en dicha parte tendremos el modo de control con el que se ha ejecutado el script, se define un array numpy. Un array numpy define una red de valores del mismo tipo, en nuestro caso números enteros, y se utiliza para poder incluir en el array un dato nuevo y redimensionarse, haciendo más fácil el control de los índices.

Asignaremos los vértices que conforman la mesh y de estos se imprimirán por la consola los que tengan un valor de seleccionado igual a True, pues quiere decir que son los que están en activo en ese momento.

A continuación, veremos cómo usar correctamente el funcionamiento de este script auxiliar creado para obtener los índices de los vértices. Dichos vértices serán los que se deben de asignar en el material del Iris seleccionaremos en el modo edición. Para seleccionar todos los vértices se puede usar el atajo “Tecla A” y posteriormente deseleccionar aquellos que forman el iris para con el atajo “CTRL + I” al aplicar la inversa se seleccionen. El flujo de este proceso se vería en la aplicación como el de *Fig. 72. Proceso para seleccionar nodos “submalla”.*

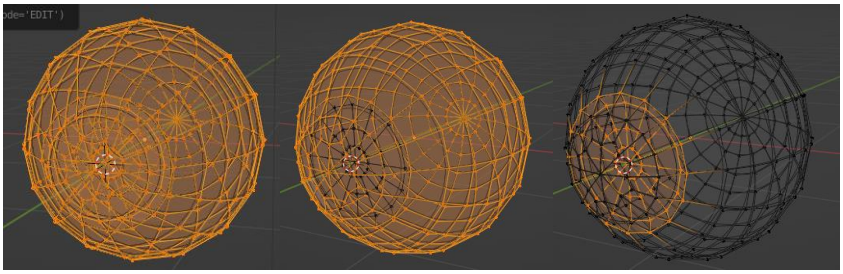

Fig. 72. Proceso para seleccionar nodos “submalla”

Una vez seleccionados los vértices antes de dar al botón para imprimirlos abriremos la ventana de consola en Window → Toggle System Console tal y como en la *Fig. 73. Abrir consola Blender y menú surgido de la ejecución.*

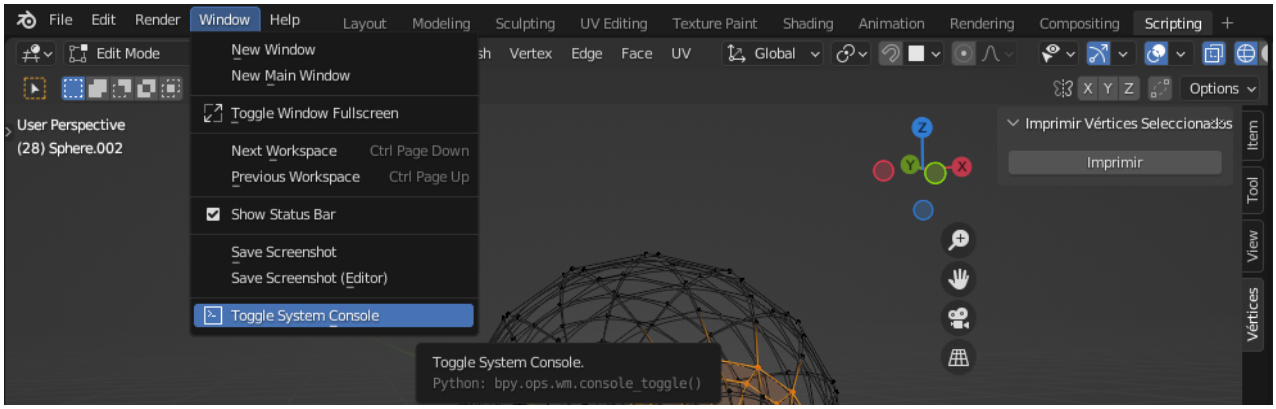

Fig. 73. Abrir consola Blender y menú surgido de la ejecución.

Al clicar sobre el botón "Imprimir" que se ha creado al ejecutar dicho script, podremos copiar y pegar en un bloc de notas, en nuestro caso se ha usado Notepad++, o también se podría realizar con otra aplicación similar para dejarlo en formato de lista de Python. En la *Fig. 74. Resultados índices de vértices.* se ve la ejecución del script y en la *Tabla 2. Lista obtenida de los índices de los vértices Iris* la lista obtenida.

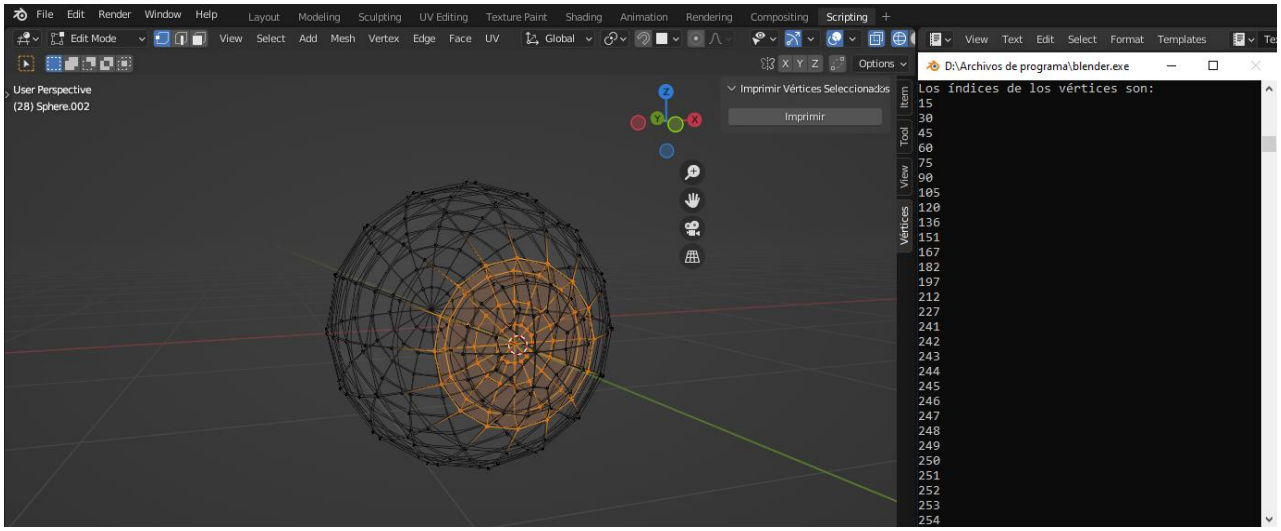

Fig. 74. Resultados índices de vértices.

```
vertices_Iris =
[15,30,45,60,75,90,105,120,136,151,167,182,197,212,227,241,242,243,
244,245,246,247,248,249,250,251,252,253,254,255,256,257,258,259,260,
261,262,263,264,265,266,267,268,269,270,271,272,273,274,275,276,277,
278,279,280,281,282,283,284,285,286,287,288,289,290]
```

Tabla 2. Lista obtenida de los índices de los vértices Iris.

5.2 Seleccionar los índices de los materiales “Superficie del ojo” y del “Iris” a través de Python.

En este segundo paso al previamente haber obtenido la lista de los índices con los vértices para poder seleccionarlos posteriormente y aplicar el material he optado por separar esta parte de investigar cómo seleccionar a través de scripting los vértices deseados. Para esta sección se ha creado el script *“12.2 seleccionar_vertices_ojo.py”*, el cual al igual que con *“imprimir_vertices_seleccionados.py”* se creará un panel lateral en la vista 3D con el nombre Seleccionar Vértices como se puede ver en la *figura 75*.

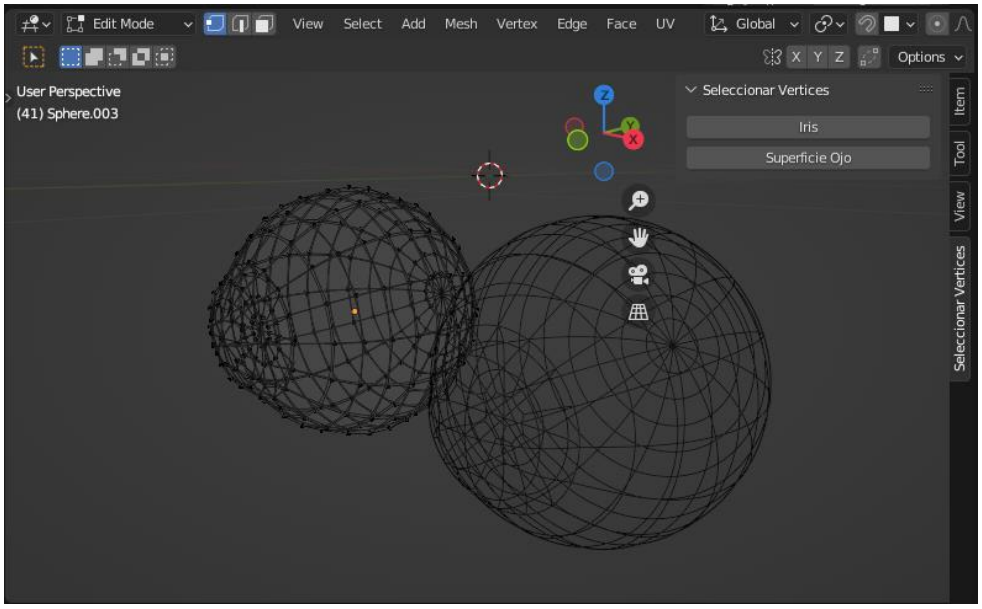

Fig. 75. Menú obtenido alejecutar script seleccionar_vertices_ojo.py

En este panel debemos diferenciar dos clases principales class ***ADDONNAME_OT_my_op(Operator):*** es la encargada de seleccionar los índices de los vértices almacenados en el array vertices_Iris, dichos elementos han sido obtenidos con el script *“imprimir_vertices_seleccionados.py”* como veremos en *Tabla 29.*

Para el correcto funcionamiento de este script, se debe de clicar sobre la mesh del ojo a la que queremos que se aplique, es por ello por lo que en la aplicación en el modo objeto se debe seleccionar.

La parte del código dentro de la clase ***class ADDONNAME_OT_my_op(Operator)*** con obj = bpy.context.active_object indicamos que la mesh seleccionada será en la que se aplicará la selección de los índices de los vértices, para ello se debe cambiar al modo edit e indicar que trabajaremos con vértices bpy.ops.object.mode_set(mode = 'EDIT') , bpy.ops.mesh.select_mode(type="VERT").

Para seleccionar los vértices tendremos que poner a valor "True", para ello se crea un bucle for con la longitud del array vertices_Iris y obj.data.vertices[vertices_Iris[i]].select = True se encargará de poner a dicho valor cada uno de los índices que coincidan con el número entero que hay en cada posición del array.

La parte del código dentro de la clase ***class ADDONNAME_OT_my_op2(Operator)*** y dentro de la definición de "execute" la cual indica lo qué debe de hacer al clicar sobre el botón trabaja de forma similar donde lo que cambia es el array pues en este caso tendrá los índices de los vértices que hacen alusión a la superficie del ojo en la *tabla 3*.

```
vertices_Superficie =
[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,
20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,
44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,64,65,66,67,
68,69,70,71,72,73,74,75,76,77,78,79,80,81,82,83,84,85,86,87,88,89,90,91,
92,93,94,95,96,97,98,99,100,101,102,103,104,105,106,107,108,109,110,111,
112,113,114,115,116,117,118,119,120,121,122,123,124,125,126,127,128,129,
130,131,132,133,134,135,136,137,138,139,140,141,142,143,144,145,146,147,
148,149,150,151,152,153,154,155,156,157,158,159,160,161,162,163,164,165,
166,167,168,169,170,171,172,173,174,175,176,177,178,179,180,181,182,183,
184,185,186,187,188,189,190,191,192,193,194,195,196,197,198,199,200,201,
202,203,204,205,206,207,208,209,210,211,212,213,214,215,216,217,218,219,
220,221,222,223,224,225,226,227,228,229,230,231,232,233,234,235,236,237,
238,239,240,241]
```

Tabla 3. Lista obtenida de los índices de la superficie del ojo.

Probando dicho script en Python podemos observar cómo selecciona lo indicado previamente al hacer clic sobre los botones que presenta el panel. En la *Fig. 76. Demostración de la solución de presionar botón Iris* podremos ver el resultado de pulsar sobre el botón de seleccionar los vértices del iris y en la *Fig. 77. Demostración de la solución de presionar botón Superficie Ojo* el resultado de pulsar sobre el botón de seleccionar los vértices de la superficie del ojo

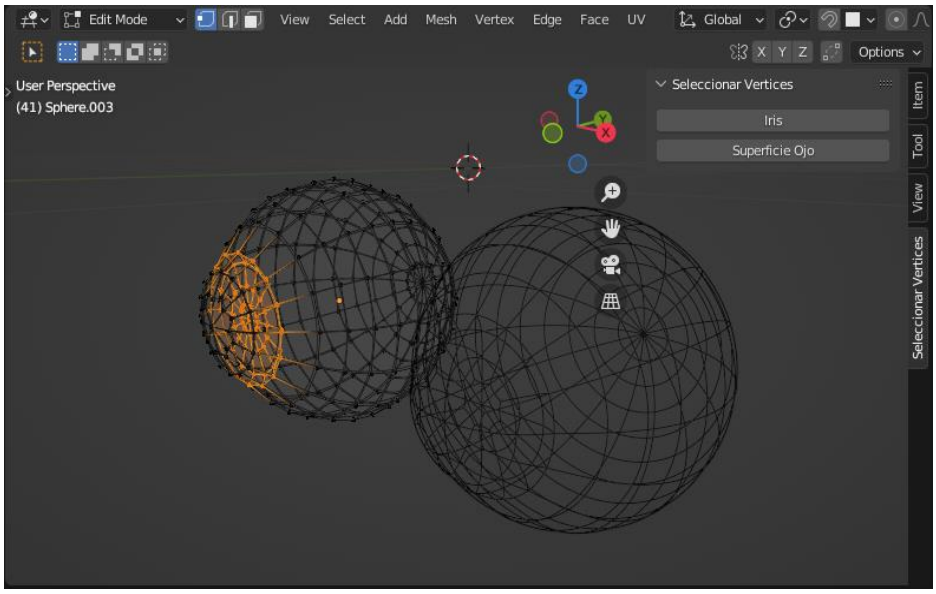

Fig. 76. Demostración de la solución de presionar botón Iris

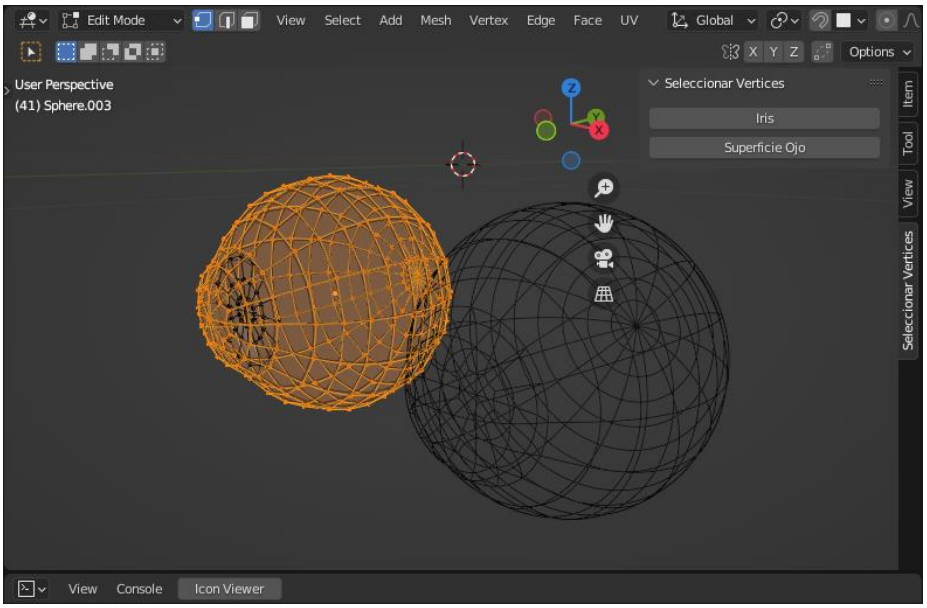

Fig. 77. Demostración de la solución de presionar botón Superficie Ojo

5.3 Creación de los materiales “Superficie del ojo” e “Iris” y asignar materiales a los vértices estudiados en los puntos anteriores.

Esta parte del scripting es la más crucial, ya que es la encarga de crear los materiales “Superficie Ojo” e “Iris” con sus respectivos nodos, los cuales se les tiene que asignar valores y enlazarlos para que el material se vea reflejado tal y como se desea. En este caso no se creará un panel en la interfaz para presionar un botón que cree todo lo expuesto, pues seleccionando la mesh en la que queremos aplicar los materiales y ejecutando este script ya sería suficiente. Dicho script tiene el nombre de *“11.3 Creacion_Materiales_Ojo.py”*. A diferencia del apartado anterior el cual podíamos ver cómo avanzaban los materiales a medida que se creaban y relacionaban los nodos paso a paso, en este caso lo tendremos más complicado pues al ejecutarse se crea al instante y vemos el cambio de cero a cien , es decir; de no tener los materiales creados a tenerlos creados por completo. A continuación, voy a seccionar el fichero *“Creacion_Materiales_Ojo.py”*, para explicar las partes más importantes que de este script plasmado en la sexta tabla. En esta primera parte del código (*Tabla 4*) se asigna en obj la mesh seleccionada, se crea el material Superficie Ojo, se asigna al objeto dicho material y se comprueba que se haya creado correctamente, de lo contrario se creará.

```
################################## SUPERFICIE OJO ###############
obj = bpy.context.object  # Asignación de la mesh seleccionada con el
puntero
mat = bpy.data.materials.new(name='Superficie Ojo')  # Creación del material
# Configuración del material
obj.data.materials.append(mat)  # Asignación del material
obj = bpy.context.active_object

new_mat = bpy.data.materials.get('Superficie Ojo')
if not new_mat:
    new_mat = bpy.data.materials.new('Superficie Ojo')

# Activamos las refracción para poder ver la pupila
bpy.context.object.active_material.use_screen_refraction = True
```

```
# habilitaremos el uso de nodos, ya que sin ello no podríamos crear ninguno
new_mat.use_nodes = True

# Definimos la estructura que tendrán los nodos y los enlaces para su
creación
node_tree = new_mat.node_tree
nodos = node_tree.nodes
nodos.clear()

enlaces = node_tree.links
enlaces.clear()
```

Tabla 4. Sección de código de Creacion_Materiales_Ojo.py (Parte 1)

Pondremos en modo “True” la refracción, el cual equivaldría a activar a través de la aplicación Blender en las propiedades de renderizado esta zona. Sin este valor activo no se podría apreciar el material Iris el cual se aplica a los nodos creados dentro de la esfera de la mesh que forma el ojo. Si lo hiciésemos en la aplicación sería como en la *Fig. 78. Ejemplo gráfico de poner a True la refracción.* Y por último en esta sección del código se define la estructura del árbol de nodos el cuál contendrá a los propios nodos en sí y a la estructura de los enlaces para poder linkear los nodos como se desea.

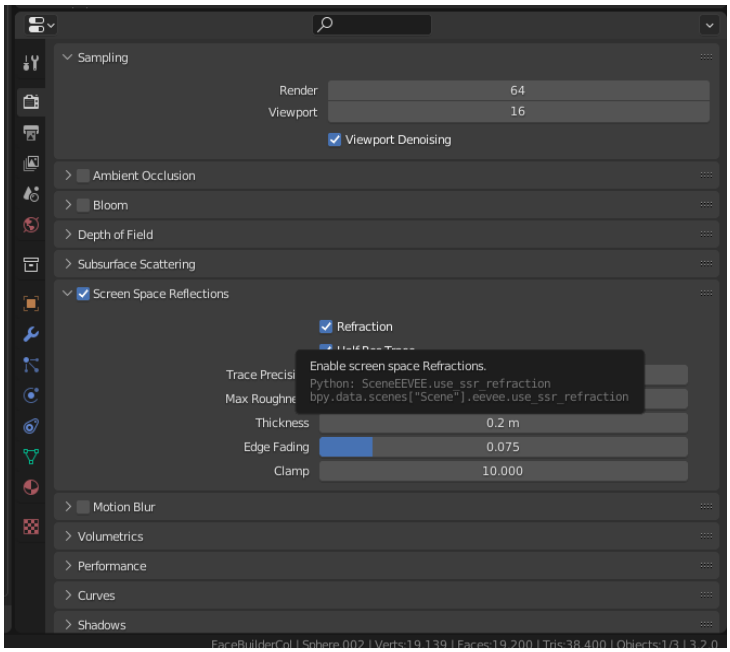

Fig. 78. Ejemplo gráfico de poner a True la refracción.

Sin este valor activo no se podría apreciar el material Iris el cual se aplica a los nodos creados dentro de la esfera de la mesh que forma el ojo. Y por último en esta sección del código se define la estructura del árbol de nodos el cuál contendrá a los propios nodos en sí y a la estructura de los enlaces para poder linkear los nodos como se desea.

5.3.1 Tipos de nodos utilizados para crear los materiales

A continuación, se va a explicar los tipos de nodos que son utilizados en el script de la creación de ambos materiales.

- **Frame Node (Frame_S, Frame_M y Frame_L)**

  Son nodos de tipo "Frame", este tipo sirve para estructurar de una forma más intuitiva los nodos y de una apariencia ordenada, para ser intuitiva y sencilla de entender gráficamente en la aplicación dicho código se muestra en la *tabla 5*.

```
# Frame_S será el nodo que englobe el nodo Mapping, el nodo Voronoi
Texture que da textura voronoi a los vasos sanguíneos de menor
longitud y el nodo ColorRamp
nodo_actual = nodos.new(type='NodeFrame')
nodo_actual.color = (0.608, 0.608, 0.608)
nodo_actual.label = 'S'
nodo_actual.label_size = 20
nodo_actual.location = (-1756.5, 1353.1)
nodo_actual.name = 'Frame_S'
nodo_actual.select = False
nodo_actual.width = 800
```

Tabla. 5. Parte de código sobre Frame Node

Se crea un nuevo nodo con la estructura previamente definida y el tipo NodeFrame es una palabra reservada de Blender para crear el nodo Frame. Le asignaremos un color negro neutral, la etiqueta "S" la cual aparecerá como título del Frame, el tamaño de dicho título, la posición en la cual aparecerá en la ventana de "Shader Editor" y su ancho.

En este caso el nodo "Frame_S" se utilizará para englobar al nodo Mapping_S, ColorRamp_S y Voronoi_Texture_S. Frame_M y Frame_L funcionan de manera similar. Una vez ejecutado se verá como en la *figura 79.*

Fig. 79. Cómo se ve Frame Node una vez ejecutado el script

- **Mapping Node 'Normal' (Mapping_S, Mapping_M, Mapping_L, Mapping_Iris)**

El nodo Mapping se utilizará para transformar la textura de los vasos sanguíneos, S hace referencia a el tamaño de los vasos sanguíneos de menor tamaño (S=Small, M=Medium y L=Large).

```
# Mapping_S el nodo mapping de los vasos sanguíneos de corta distancia
es de tipo 'NORMAL' y la escala es de (1,2,1) haciendo referencia a
(X,Y,Z)
nodo_actual = nodos.new(type='ShaderNodeMapping')
nodo_actual.color = (0.608, 0.608, 0.608)
nodo_actual.location = (-33.8, -25.8)
nodo_actual.name = 'Mapping_S'
# Asignamos el nodo padre, ya que Mapping_S es un nodo que se
encuentra en el interior de Frame_S
parent = nodos.get('Frame_S')
if parent:
    nodo_actual.parent = parent
    while True:
        nodo_actual.location += parent.location
        if parent.parent:
```

```
            parent = parent.parent
        else:
            break
nodo_actual.select = False
nodo_actual.vector_type = 'NORMAL'
nodo_actual.width = 240.0
nodo_actual.inputs[0].default_value = [0.0, 0.0, 0.0]
nodo_actual.inputs[1].default_value = [0.0, 0.0, 0.0]
nodo_actual.inputs[2].default_value = [0.0, 0.0, 0.0]
nodo_actual.inputs[3].default_value = [1.0, 2.0, 1.0]
nodo_actual.outputs[0].default_value = [0.0, 0.0, 0.0]
```

Tabla. 6. Parte de código sobre Mapping Node 'Normal'

Como vemos en la *Tabla. 6. Parte de código sobre Mapping Node 'Normal'* para obtener un nodo como el de la figura 80, en el que se crea un nuevo nodo con la estructura previamente definida y el tipo ShaderNodeMapping es una palabra reservada de Blender para crear el nodo Mapping. Dicho nodo queremos que esté dentro del Frame S, por lo que asignaremos a 'Frame_S' el nodo padre. Al ser el vector de tipo normal se transformará el vector con longitud unitaria y como salida tiene un vector la cual generalmente hace de entrada de un nodo de textura. A destacar nodo_actual.inputs[3].default_value = [1.0, 2.0, 1.0] hace referencia a Scale del nodo donde hace referencua a [X,Y,Z] para transformar la forma, longitud y profundidad de los vasos sanguíneos. Como entrada en la posición 0 vemos que tiene el campo vector, el cual por lo general viene de una salida de nodo de coordenadas de textura. En Blender se verá como en la *Fig. 80. Cómo se ve Mapping Node 'Normal' una vez ejecutado el script.*

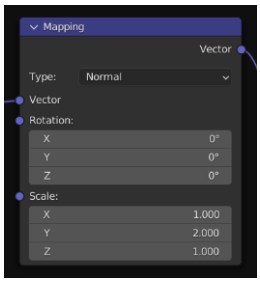

Fig. 80. Cómo se ve Mapping Node 'Normal' una vez ejecutado el script

- **Mapping Node 'Point' (Mapping_Cornea, Mapping_Iris_Superficie, Mapping_Pupila)**

Optamos por explicar este nodo ya que seguimos con el tipo Mapping, donde como distinto a los anteriores resaltaría el tipo de vector, en este caso en la *Tabla. 7. Parte de código sobre Mapping Node 'Point'* el tipo es 'POINT' la forma en que funcionan las matemáticas es mediante dividir los valores de escala. El nodo se reflejará en la aplicación como en la *Fig. 81. Cómo se ve Mapping Node 'Point' una vez ejecutado el script.*

```
nodo_actual = nodos.new(type='ShaderNodeMapping')
nodo_actual.color = (0.608, 0.608, 0.608)
nodo_actual.location = (-610.645, 158.537)
nodo_actual.name = 'Mapping_Cornea'
nodo_actual.select = False
nodo_actual.vector_type = 'POINT'
nodo_actual.width = 240.0
nodo_actual.inputs[0].default_value = [0.0, 0.0, 0.0]
nodo_actual.inputs[1].default_value = [0.0, 0.800, 0.0]
nodo_actual.inputs[2].default_value = [0.0, 0.0, 0.0]
nodo_actual.inputs[3].default_value = [0.5, 0.5, 0.5]
nodo_actual.outputs[0].default_value = [0.0, 0.0, 0.0]
```

Tabla. 7. Parte de código sobre Mapping Node 'Point'

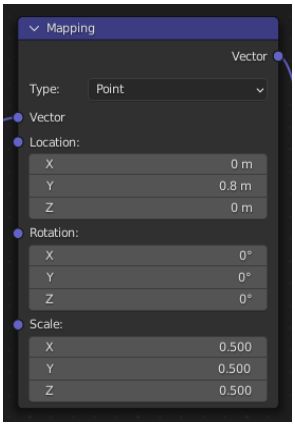

Fig. 81. Cómo se ve Mapping Node 'Point' una vez ejecutado el script

- **Voronoi Texture Node (Voronoi_Texture_S, Voronoi_Texture_M, Voronoi_Texture_L)**

Este tipo de nodo que crearemos en la *Tabla. 8. Parte de código sobre Voronoi Texture Node* se utiliza para estimar ruido Worley, que es un algoritmo que elige puntos aleatorios en el espacio y con esa distancia entre puntos crea combinaciones controlando el color de manera más precisa.

```
# Voronoi_Texture_S es el encargado de como vemos los vasos sanguíneos
de menor longitud
nodo_actual = nodos.new(type='ShaderNodeTexVoronoi')
nodo_actual.color = (0.608, 0.608, 0.608)
nodo_actual.distance = 'EUCLIDEAN'
nodo_actual.feature = 'F1'
nodo_actual.location = (277.432, -33.755)
nodo_actual.name = 'Voronoi_Texture_S'
# Asignamos el nodo padre, ya que Mapping_L es un nodo que se
encuentra en el interior del nodo de Frame_S
parent = nodos.get('Frame_S')
if parent:
    nodo_actual.parent = parent
    while True:
        nodo_actual.location += parent.location
        if parent.parent:
            parent = parent.parent
        else:
            break
nodo_actual.select = False
nodo_actual.voronoi_dimensions = '3D'
nodo_actual.width = 140.0
nodo_actual.inputs[0].default_value = [0.0, 0.0, 0.0]
nodo_actual.inputs[1].default_value = 0.0
```

```
nodo_actual.inputs[2].default_value = 20.0
nodo_actual.inputs[3].default_value = 1.0
nodo_actual.inputs[4].default_value = 0.5
nodo_actual.inputs[5].default_value = 1.0
nodo_actual.outputs[0].default_value = 0.0
nodo_actual.outputs[1].default_value = [0.0, 0.0, 0.0, 0.0]
nodo_actual.outputs[2].default_value = [0.0, 0.0, 0.0]
nodo_actual.outputs[3].default_value = 0.0
nodo_actual.outputs[4].default_value = 0.0
```

Tabla. 8. Parte de código sobre Voronoi Texture Node

Al determinar valores 3D el ruido se aplica en el vector de entrada, el rasgo se marca como F1 para que se aplique en la distancia al punto característico más cercano, así como también a su color y localización. Como métrica de distancia le indicamos que sea euclidiana para cumplir la geometría de Euclides con sus respectivos axiomas y teoremas. Una vez ejecutado el código el nodo que aparecerá en la aplicación será como el de la *Fig. 82. Cómo se ve Voronoi Texture Node una vez ejecutado el script.*

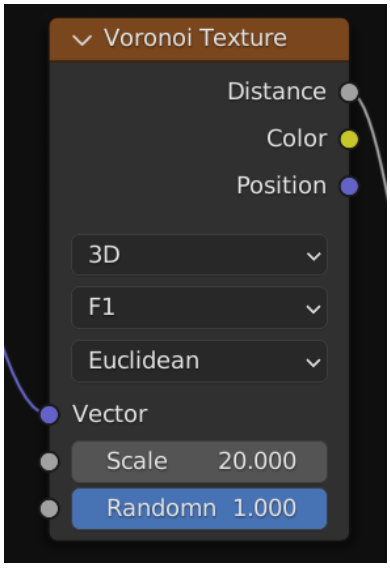

Fig. 82. Cómo se ve Voronoi Texture Node una vez ejecutado el script

- **Color Ramp Node (ColorRamp_S, ColorRamp_M, ColorRamp_L, ColorRamp_Cornea, ColorRamp_Vasos_Sanguíneos, ColorRamp_Rojo_Superficie, ColorRamp_Bump, ColorRamp_Iris_Interior, ColorRamp_Iris_Circunferencia, ColorRamp_Iris_Textura)**

  Este nodo es el encargado de asignar valores de colores con el uso de un degradado. Para explicar este tipo de nodo he optado por plasmar el **ColorRamp_Iris_Circunferencia** pues es el que tiene más paradas y colores. Para crearlo se utilizará el código como de la *Tabla. 9. Parte de código sobre Color Ramp Node*

```
# ColorRamp_Iris_Circunferencia es el nodo encargado de las tonalidades de azul en el interior del iris
nodo_actual = nodos.new(type='ShaderNodeValToRGB')
nodo_actual.color = (0.608, 0.608, 0.608)
cr = nodo_actual.color_ramp
cr.color_mode = 'RGB'
cr.hue_interpolation = 'NEAR'
cr.interpolation = 'EASE'
stop_actual = cr.elements.new(0.5)
# para crear el color negro queremos en RGB los valores 0.0, 0.0, 0.0 y el 1.0 hace alusión a la opacidad siendo 1.0 completamente opaco y 0.0 sería transparente
stop_actual.color = [0.0, 0.0, 0.0, 1.0]
stop_actual = cr.elements.new(0.550)
stop_actual.color = [0.020, 0.028, 0.100, 1.0]
stop_actual = cr.elements.new(0.620)
# para crear el color azul oscuro pálido queremos en RGB los valores 0.120, 0.152, 0.200 y el 1.0 hace alusión a la opacidad siendo 1.0 completamente opaco y 0.0 sería transparente
stop_actual.color = [0.120, 0.152, 0.200, 1.0]
stop_actual = cr.elements.new(0.700)
# para crear el color azul claro queremos en RGB los valores 0.020, 0.028, 0.100 y el 1.0 hace alusión a la opacidad siendo 1.0 completamente opaco y
```

```
0.0 sería transparente
stop_actual.color = [0.302, 0.476, 0.550, 1.0]
stop_actual = cr.elements.new(0.760)
# para crear el color azul oscuro pálido queremos en RGB los valores 0.120, 0.152, 0.200 y el 1.0 hace alusión a la opacidad siendo 1.0 completamente opaco y 0.0 sería transparente
stop_actual.color = [0.087, 0.133, 0.25, 1.0]
stop_actual = cr.elements.new(0.764)
# para crear el color negro queremos en RGB los valores 0.0, 0.0, 0.0 y el 1.0 hace alusión a la opacidad siendo 1.0 completamente opaco y 0.0 sería transparente
stop_actual.color = [0.0, 0.0, 0.0, 1.0]
removed_black = removed_white = False
# Esta parte de código es necesaria para que no se creen los puntos de parada de los extremos (color negro puro y blanco puro)
for i in range(len(cr.elements) - 1, -1, -1):
    stop = cr.elements[i]
    if not removed_black and stop.position == 0 and all([stop.color[i] == (0, 0, 0, 1)[i] for i in range(4)]):
        cr.elements.remove(stop)
        removed_black = True
    if not removed_white and stop.position == 1 and all([stop.color[i] == (1, 1, 1, 1)[i] for i in range(4)]):
        cr.elements.remove(stop)
        removed_white = True
nodo_actual.location = (-237.990, 246.606)
nodo_actual.name = 'ColorRamp_Iris_Circunferencia'
nodo_actual.width = 244.526
nodo_actual.inputs[0].default_value = 0.5
nodo_actual.outputs[0].default_value = [0.0, 0.0, 0.0, 0.0]
nodo_actual.outputs[1].default_value = 0.0
```

Tabla. 9. Parte de código sobre Color Ramp Node

Se define el tipo *ShaderNodeValToRGB*, y la estructura de Color Ramp ”cr” para poder definir el modo en el que se tratarán los colores “RGB” como podemos apreciar en la *Fig. 83. Cómo se ve Color Ramp Node una vez ejecutado el script*.

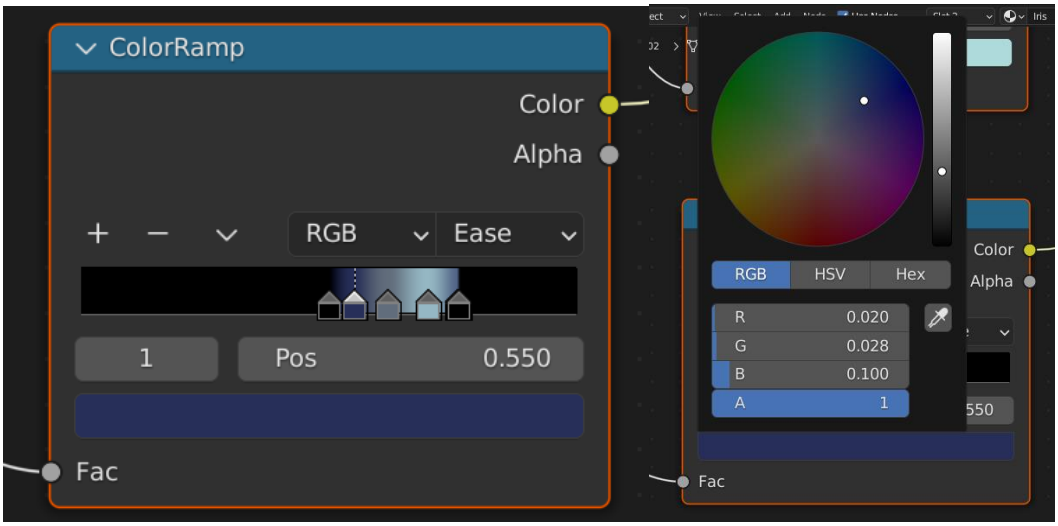

Fig. 83. Cómo se ve Color Ramp Node una vez ejecutado el script

Si nos centramos en esta parte de código (*Tabla 10*), nosotros pondremos el límite del degradado en la posición 0.550, y habiendo asignado el valor de parada a “stop_actual” entonces podremos meter el valor del color en RGB, el último número decimal debe tener un valor entre 0 a 1 siendo la opacidad del color en el material.

```
stop_actual = cr.elements.new(0.550)
# para crear el color azul oscuro queremos en RGB los valores 0.020,
0.028, 0.100 y el 1.0 hace alusión a la opacidad siendo 1.0
completamente opaco y 0.0 sería transparente
stop_actual.color = [0.020, 0.028, 0.100, 1.0]
```

Tabla. 10. Parte de código sobre Color Ramp Node (parada)

Y en esta otra parte, *Tabla. 11. Parte de código sobre Color Ramp Node (borrar blanco y negro en extremos)* , se utiliza para borrar los límites de degradado que

se crean por defecto por el propio nodo en sí, y a través del bucle for eliminaremos los colores por defecto sabiendo que se posicionan en los extremos 0 el color negro (0,0,0) y 1 el color blanco (1,1,1). Es necesario eliminar para que el degradado se limite a la parte que se ha programado.

```
# Esta parte de código es necesaria para que no se creen los puntos de
parada de los extremos (color negro puro y blanco puro)
for i in range(len(cr.elements) - 1, -1, -1):
    stop = cr.elements[i]
    if not removed_black and stop.position == 0 and all([stop.color[i]
== (0, 0, 0, 1)[i] for i in range(4)]):
        cr.elements.remove(stop)
        removed_black = True
    if not removed_white and stop.position == 1 and all([stop.color[i]
== (1, 1, 1, 1)[i] for i in range(4)]):
        cr.elements.remove(stop)
        removed_white = True
```

Tabla. 11. Parte de código sobre Color Ramp Node (borrar blanco y negro en extremos)

- **Mix Node (Mix_L, Mix_M_L, Mix_S_ML, Mix_Total, Mix_Vasos_Sanguineos, Mix)**

Este nodo que creamos en la *Tabla. 12. Parte de código sobre Mix Node* mezcla los colores, con la propiedad de mezclar que se establezca Darken (vasos sanguíneos), Mix (Mix_Total) y Overlay. Una vez ejecutemos el script en la aplicación de Blender se verá reflejado como en la *Fig. 84. Cómo se ve Color Mix Node una vez ejecutado el script.*

```
# Mix_Total es el nodo que mezcla de modo superpuesto todos los
colorRamp relacionados con el rojo sanguíneo para mandarlo como color
base al nodo principal BSDF
nodo_actual = nodos.new(type='ShaderNodeMixRGB')
nodo_actual.blend_type = 'OVERLAY'
nodo_actual.color = (0.608, 0.608, 0.608)
nodo_actual.location = (629.845, 1025.724)
nodo_actual.name = 'Mix_Total'
nodo_actual.select = False
nodo_actual.use_alpha = False
nodo_actual.use_clamp = False
nodo_actual.width = 140.0
nodo_actual.inputs[0].default_value = 1.0
nodo_actual.inputs[1].default_value = [0.5, 0.5, 0.5, 1.0]
nodo_actual.inputs[2].default_value = [0.5, 0.5, 0.5, 1.0]
nodo_actual.outputs[0].default_value = [0.0, 0.0, 0.0, 0.0]
```

Tabla. 12. Parte de código sobre Mix Node

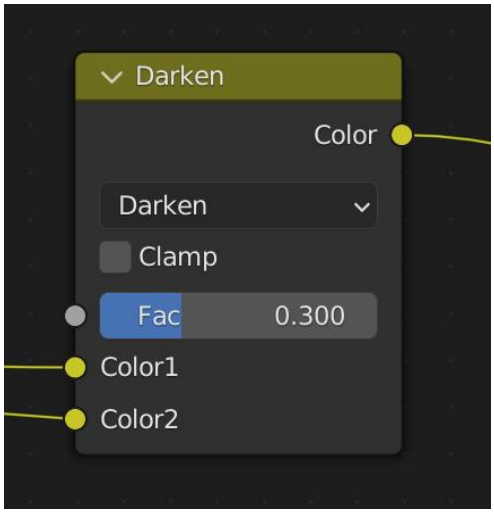

Fig. 84. Cómo se ve Color Mix Node una vez ejecutado el script

- **Gradient_Texture_Cornea, Gradient_Texture_Iris_Superficie, Gradient_Texture_Iris_Medida**

  Este nodo se utiliza para generar valores de colores e intensidad en función de la entrada que es un vector. Al usar el tipo de gradiente esférico se crea un degradado opuesto utilizando la longitud del vector de entrada. Y como salida utilizaremos el color de textura. Al aplicar el código reflejado en la *Tabla. 13. Parte de código sobre Gradient_Texture_Cornea* el nodo se verá en Blender como el de la *Fig. 85. Cómo se ve Gradient Texture Node una vez ejecutado el script*

```
# Gradient_Texture_Cornea sirve para dar esa transparencia necesaria
para poder aprecias el iris/pupila del ojo, pues de estar en otro modo
la transparencia no se situaría con esas dimensiones en la parte
frontal
nodo_actual = nodos.new(type='ShaderNodeTexGradient')
nodo_actual.color = (0.608, 0.608, 0.608)
nodo_actual.gradient_type = 'SPHERICAL'
nodo_actual.location = (-320.645, 96.098)
nodo_actual.name = 'Gradient_Texture_Cornea'
nodo_actual.select = False
nodo_actual.width = 140.0
```

Tabla. 13. Parte de código sobre Gradient_Texture_Cornea

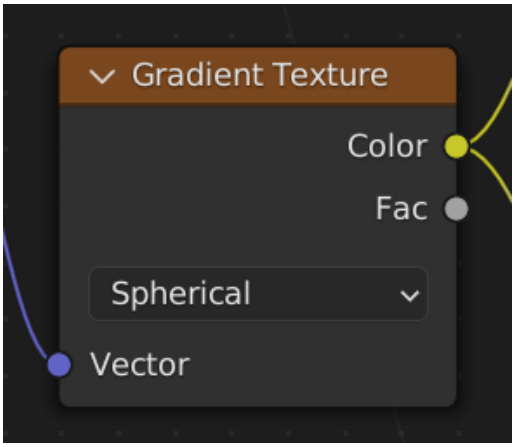

Fig. 85. Cómo se ve Gradient Texture Node una vez ejecutado el script

- **Texture Coordinate Node (Texture_Coordinate_Cornea, Texture_Coordinate_Iris_Superficie, Texture_Coordinate_S_M_L, Texture_Coordinate_Iris_Base, Texture_Coordinate_Iris_Interior).**

  El nodo de coordenadas de texturizado, el cual se crea con el código que veremos en *Tabla. 17. Parte de código sobre Texture Coordinate Node* ofrece un conjunto de coordenadas para formar una textura y asignarla a un vector. Como peculiaridad de este nodo no posee entradas, por lo que podríamos decir que es un punto de inicio en el árbol de nodos.

```
# Gradient_Texture_Cornea sirve para dar esa transparencia necesaria
para poder aprecias el iris/pupila del ojo,
nodo_actual = nodos.new(type='ShaderNodeTexCoord')
nodo_actual.color = (0.608, 0.608, 0.608)
nodo_actual.from_instancer = False
nodo_actual.location = (-820.645, 141.537)
nodo_actual.name = 'Texture_Coordinate_Cornea'
nodo_actual.object = None
nodo_actual.select = False
nodo_actual.width = 140.0
```

Tabla. 14. Parte de código sobre Texture Coordinate Node

La salida del nodo que usaremos principalmente es el objeto, que se puede animar para mover una textura alrededor o a través de una superficie. El nodo obtenido será como el que vemos en la *figura 86*.

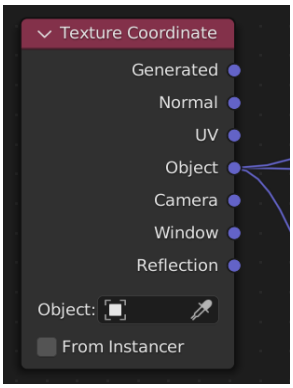

Fig. 86. Cómo se ve Texture Coordinate Node una vez ejecutado el script

- **Noise Texture Node (Noise_Texture_L, Noise_Texture_Iris)**

  El nodo creado en *Tabla. 15. Parte de código sobre Texture Noise Node* genera un ruido para crear texturas procedurales de algunos efectos generados por ordenador.

```
# Noise_Texture_L es un ruido 3D el cual le da ese
 efecto a los vasos sanguíneos
 nodo_actual = nodos.new(type='ShaderNodeTexNoise')
 nodo_actual.color = (0.608, 0.608, 0.608)
 nodo_actual.location = (-2281.211, 671.236)
 nodo_actual.name = 'Noise_Texture_L'
 nodo_actual.noise_dimensions = '3D'
 nodo_actual.select = False
 nodo_actual.width = 140.0
 nodo_actual.inputs[0].default_value = [0.0, 0.0, 0.0]
 nodo_actual.inputs[1].default_value = 0.0
 nodo_actual.inputs[2].default_value = 5.0
 nodo_actual.inputs[3].default_value = 2.0
 nodo_actual.inputs[4].default_value = 0.5
 nodo_actual.inputs[5].default_value = 0.0
 nodo_actual.outputs[0].default_value = 0.0
 nodo_actual.outputs[1].default_value = [0.0, 0.0, 0.0, 0.0]
```

Tabla. 15. Parte de código sobre Texture Noise Node

Al determinar valores 3D el ruido se aplica en el vector de entrada, el rasgo se marca como F1 para que se aplique en la distancia al punto característico más cercano, así como también a su color y localización. Como métrica de distancia le indicamos que sea euclidiana para cumplir la geometría de Euclides con sus respectivos axiomas y teoremas. Quedando un resultado como el de la *figura 87.*

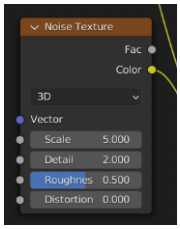

Fig. 87. Cómo se ve Noise Texture Node una vez ejecutado el script

- **Bump**

  El nodo con el código de la *tabla 16* genera un ruido para crear texturas procedurales de algunos efectos generados por ordenador, obteniendo el nodo que aparece en la *Fig. 88. Cómo se ve Bump Node una vez ejecutado el script*.

```
# Bump es el nodo otorga una especie de altura a los vasos sanguíneos
a través del nodo bump
nodo_actual = nodos.new(type='ShaderNodeBump')
nodo_actual.color = (0.608, 0.608, 0.608)
nodo_actual.invert = True
nodo_actual.location = (-65.068, -147.066)
nodo_actual.name = 'Bump'
nodo_actual.select = False
nodo_actual.width = 140.0
nodo_actual.inputs[0].default_value = 0.100
nodo_actual.inputs[1].default_value = 0.100
nodo_actual.inputs[2].default_value = 1.0
nodo_actual.inputs[3].default_value = [0.0, 0.0, 0.0]
nodo_actual.outputs[0].default_value = [0.0, 0.0, 0.0]
```

Tabla. 16. Parte de código sobre Bump Node

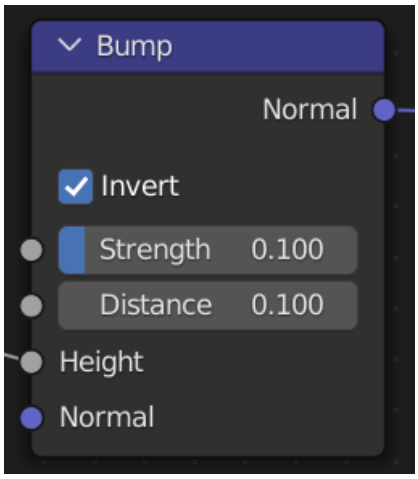

Fig. 88. Cómo se ve Bump Node una vez ejecutado el script

- **Material Output**

  Este nodo que se obtiene de la ejecución de la *Tabla. 17. Parte de código sobre Material Output* es usado para plasmar la información de la superficie del material indicada con los nodos y sus respectivos enlaces para que se aplique a la superficie del objeto. Obteniendo un nodo como el de *Fig. 89. Cómo se ve Material Output Node una vez ejecutado el script*

```
# Material Output es el nodo encargado de que todos los nodos hagan
efecto en la superficie, sin él no se vería reflejado en la consola
blender
nodo_actual = nodos.new(type='ShaderNodeOutputMaterial')
nodo_actual.color = (0.608, 0.608, 0.608)
nodo_actual.is_active_output = True
nodo_actual.location = (576.799, 302.339)
nodo_actual.name = 'Material Output'
nodo_actual.select = False
nodo_actual.target = 'ALL'
nodo_actual.width = 140.0
nodo_actual.inputs[2].default_value = [0.0, 0.0, 0.0]
nodo_actual.inputs[3].default_value = 0.0
```

Tabla. 17. Parte de código sobre Material Output

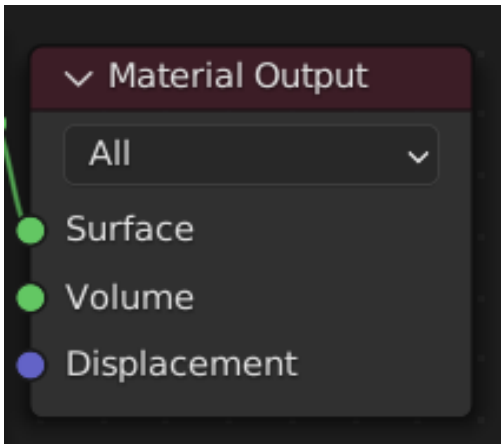

Fig. 89. Cómo se ve Material Output Node una vez ejecutado el script

- **Principled BSDF**

  Es el nodo previo a Material Output por el cual llega la información generada por los anteriores nodos. Este nodo utiliza un sombreador "PBR" para cumplir compatibilidad con otros programas como podrías ser Renderman® y Unreal Engine®. Dicho nodo tendrá el código como el que se refleja en la *tabla 18.*

```
# Principled BSDF es el nodo que se crea por defecto en el cual se
aplicará el nodo Mix_Total, Bump, y el ColorRamp_Vasos_Sanguíneos
nodo_actual = nodos.new(type='ShaderNodeBsdfPrincipled')
nodo_actual.color = (0.608, 0.608, 0.608)
nodo_actual.distribution = 'GGX'
nodo_actual.location = (324.994, 311.986)
nodo_actual.name = 'Principled BSDF'
nodo_actual.subsurface_method = 'BURLEY'
nodo_actual.width = 240.0
nodo_actual.inputs[0].default_value = [0.800, 0.800, 0.800, 1.0]
nodo_actual.inputs[1].default_value = 0.0
nodo_actual.inputs[2].default_value = [1.0, 0.200, 0.100]
nodo_actual.inputs[3].default_value = [0.800, 0.800, 0.800, 1.0]
nodo_actual.inputs[4].default_value = 1.399
nodo_actual.inputs[5].default_value = 0.0
nodo_actual.inputs[6].default_value = 0.0
nodo_actual.inputs[7].default_value = 0.5
nodo_actual.inputs[8].default_value = 0.0
nodo_actual.inputs[9].default_value = 0.0500
nodo_actual.inputs[10].default_value = 0.0
nodo_actual.inputs[11].default_value = 0.0
nodo_actual.inputs[12].default_value = 0.0
nodo_actual.inputs[13].default_value = 0.5
nodo_actual.inputs[14].default_value = 0.0
nodo_actual.inputs[15].default_value = 0.5
nodo_actual.inputs[16].default_value = 1.450
```

```
nodo_actual.inputs[17].default_value = 1.0
nodo_actual.inputs[18].default_value = 0.0
nodo_actual.inputs[19].default_value = [0.0, 0.0, 0.0, 1.0]
nodo_actual.inputs[20].default_value = 1.0
nodo_actual.inputs[21].default_value = 1.0
nodo_actual.inputs[22].default_value = [0.0, 0.0, 0.0]
nodo_actual.inputs[23].default_value = [0.0, 0.0, 0.0]
nodo_actual.inputs[24].default_value = [0.0, 0.0, 0.0]
nodo_actual.inputs[25].default_value = 0.0
```

Tabla. 18. Parte de código sobre Principled BSDF

Este nodo incluye múltiples capas para crear una amplia variedad de materiales. Nosotros como color base le pasaremos el enlace del nodo Mix que engloba todos los anteriores para que se aplique a la superficie del ojo. En él podremos determinar el color base, la textura metálica que en este caso es nula, la textura especular que aparecerá en valor intermedio, y más propiedades que se pueden aplicar como entradas de dicho nodo. Obteniendo un nodo como el de la *Fig. 90. Cómo se ve Principled BSDF Node una vez ejecutado el script.*

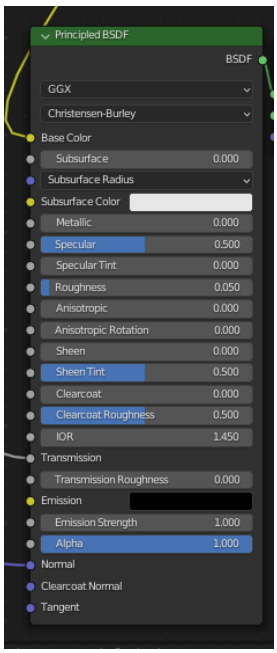

Fig. 90. Cómo se ve Principled BSDF Node una vez ejecutado el script

5.3.2 Explicación de la implementación de los enlaces entre los nodos en código

Un nodo además de aplicar su propia función se compone de inputs (entradas) y de outputs (salidas). No es necesario que todas las entradas y tampoco es necesario que todas las salidas sean utilizadas, de hecho es más común cambiar determinadas propiedades y dejar las demás intactas.Para poder determinar los valores que queremos aplicar en el nodo los modificaremos accediendo a ellos sabiendo que cumple la norma de que se simulan dos especies de arrays: inputs[i] y outputs[i] donde i indica el índice de la posición del array la cual van en orden descendente según lo vemos en la parte gráfica de la aplicación. La *Fig. 91. Nodo genérico para explicar enlaces* representa un nodo genérico, sin aludir a ningún tipo en concreto.

Fig. 91. Nodo genérico para explicar enlaces

Para explicar su funcionamiento, dentro de este nodo centrémonos en la parte específica de Entrada [1] de la figura 91. Para poner el valor de X= 0, Y= 0.8 y Z= 0 en la segunda propiedad deberemos de acceder a ella, sabiendo que el primer índice posible en el array de entradas es el valor de 0 tendremos que ir al índice número 2 por lo que en el array de entrada se reflejaría como `inputs[1]` al estar dentro del nodo para llegar a esta

propiedad del nodo sería tal que `nodo_actual.inputs[1]` a continuación deseamos poner el valor a [X=0,Y=0.8,Z=0]. Por lo que accederemos al valor del índice del array de ese nodo y pasaremos la lista con los valores indicados. `nodo_actual.inputs[1].default_value = [0.0, 0.800, 0.0]`. La forma de implementar este nodo a través de scripting sería con las siguientes líneas de código:

```
# Nodo Ejemplo
nodo_actual = nodos.new(type='NodoEjemplo')
nodo_actual.color = (0.608, 0.608, 0.608) # color del fondo del nodo
nodo_actual.location = (-610.645, 158.537) # Posición en la app
nodo_actual.name = 'Nodo Ejemplo'
nodo_actual.select = False
nodo_actual.width = 240.0 # Ancho del nodo en la app
nodo_actual.inputs[0].default_value = [0.0, 0.0, 0.0]
nodo_actual.inputs[1].default_value = [0.0, 0.800, 0.0]
nodo_actual.inputs[2].default_value = [0.0, 0.0, 0.0]
nodo_actual.inputs[3].default_value = [0.5, 0.5, 0.5]
nodo_actual.outputs[0].default_value = [0.0, 0.0, 0.0]
```

Tabla. 19. Parte de código sobre invención de un nodo

Una vez asignados los valores que deseamos que se apliquen en ese nodo, podemos enlazar los nodos. Una salida de un nodo haría la entrada del siguiente nodo. Utilizaríamos la variable creada con su respectiva arquitectura `enlaces = node_tree.links` y a partir de enlaces podemos enlazar los nodos con la siguiente sintaxis:

```
enlaces.new(nodos["Nodo_1"].outputs[i], nodos["Nodo_2].inputs[j])
```

Donde i y j representan los índices del elemento del arrays que deseamos enlazar. Un ejemplo práctico sería del nodo Texture_Coordinate_Cornea queremos enlazar el Objeto y que funcione como vector en el nodo Mapping_Cornea.

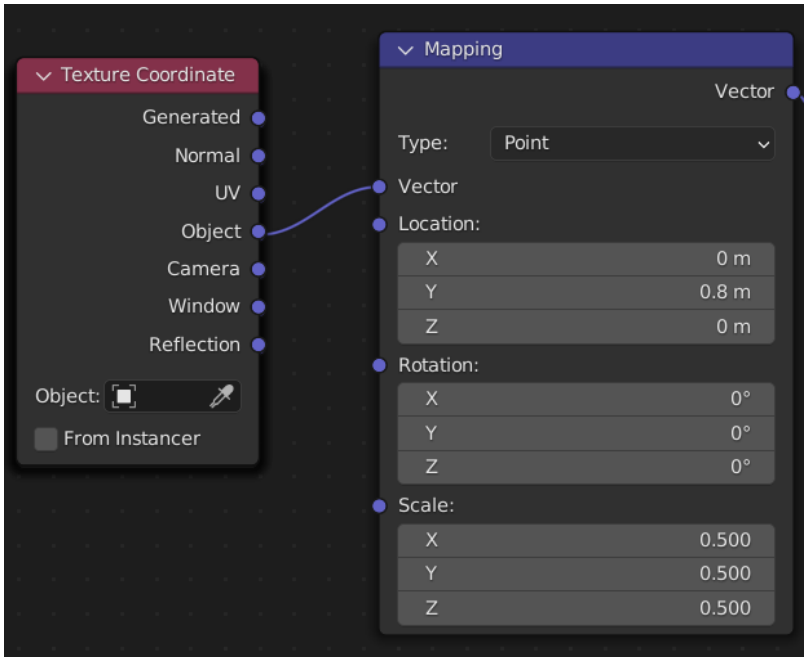

Fig. 92. Enlace entre dos nodos

Como vemos en la *Fig. 92. Enlace entre dos nodos* el parámetro Objeto del nodo Texture_Coordinate_Cornea se encuentra en el elemento número 3 si empezamos contando de arriba a abajo desde cero y al estar a la derecha del nodo sabemos que es una salida, es decir, pertenece al array de outputs de ese nodo. Mientras que la propiedad de Vector del nodo Mapping_Cornea, se encuentra en la posición número 0, al estar en la parte izquierda del nodo sabemos que es una entrada, por lo que pertenece al array inputs de ese nodo. El enlace entre estos nodos quedaría como la *tabla 20.*

```
enlaces.new(nodos["Texture_Coordinate_Cornea"].outputs[3],
nodos["Mapping_Cornea"].inputs[0])
```

Tabla. 20. Parte de código sobre invención de enlace de un nodo

5.3.3 Asignación de los materiales a la mesh.

Al crear un material habiendo seleccionado previamente la mesh en la cual queremos trabajar se aplicará en cada uno de los vértices que componen dicha mesh. Es por eso que el primer material que he optado por crear es el de "Superficie Ojo" el cual se aplicará en toda la mesh. Pero para la creación de los siguientes materiales no se asignarían a la mesh a no ser que se seleccionen los vértices deseados y una vez seleccionados se aplique la asignación del material en esos vértices. Es aquí donde entran las últimas líneas del script *"Creacion_Materiales_Ojo.py"* en la *Tabla. 21. Parte de código sobre asignación de material.*

```
# Definimos los vértices que comprenden el iris en vertices_Iris
bpy.context.object.active_material_index = 1
vertices_Iris = [15,30,45,60,75,90,105,120,136,151,167,182,197,212,
227,241,242,243,244,245,246,247,248,249,250,251,252,253,254,255,256,
257,258,259,260,261,262,263,264,265,266,267,268,269,270,271,272,273,
274,275,276,277,278,279,280,281,282,283,284,285,286,287,288,289,290]

# Bucle para seleccionar los vértices definidos en la lista vertices_Iris
for i in range(len(vertices_Iris)):
     obj.data.vertices[vertices_Iris[i]].select = True

bpy.ops.object.mode_set(mode = 'EDIT')

# Asignamos el material Iris a los vértices seleccionados
bpy.ops.object.material_slot_assign()
bpy.ops.object.mode_set(mode = 'OBJECT')
```

Tabla. 21. Parte de código sobre asignación de material

Donde a través del script auxiliar *"imprimir_vertices_seleccionados.py"* de la forma indicada en este mismo documento se han obtenido los índices de los vértices del ojo como ya veremos en la *Tabla 30. Código de seleccionar_vertices_ojo.py* y a través del script auxiliar *"seleccionar_vertices_ojo.py"* recorremos el array creado poniendo a valor True los índices de la mesh indicados en el propio array.

```
# Bucle para seleccionar los vértices definidos en la lista
vertices_Iris
for i in range(len(vertices_Iris)):
     obj.data.vertices[vertices_Iris[i]].select = True
```

Tabla. 22. Parte de código sobre seleccionar vértices

Y posteriormente lo asignaremos y volveremos al modo objeto para poder ver en la aplicación Blender de manera más cómoda la creación y asignación de los materiales en la mesh del ojo.

```
bpy.ops.object.mode_set(mode = 'EDIT')

# Asignamos el material Iris a los vértices seleccionados
bpy.ops.object.material_slot_assign()
bpy.ops.object.mode_set(mode = 'OBJECT')
```

Tabla. 23. Parte de código sobre cambiar de modo de vista

6. Ampliación de script (Parte I) : "Creación_Materiales_Ojo.py"

Sobre el script de *"Creacion_Materiales_Ojo.py"* se ha decidido crear más materiales del iris, los cuales posteriormente en otro script diferente podrán ser aplicados pulsando un botón. Lo único diferente respecto al anterior script es que hay más materiales del iris, pero los valores de las propiedades de los Color Ramp varían para aplicar colores diferentes de ojo. Por ejemplo, esta parte pertenece a los dos nodos color Ramp encargados del color del ojo, es por ello que los valores difieren de como hemos visto en el anterior apartado, pues ahora tiene unos valores de colores RGB que se ajustan más a la tonalidad verde, en el caso que se expone en la *tabla 24.*

```
# ColorRamp_Iris_Interior es el nodo encargado de las tonalidades de
azul en el interior del iris
nodo_actual = nodos.new(type='ShaderNodeValToRGB')
nodo_actual.color = (0.608, 0.608, 0.608)
cr = nodo_actual.color_ramp
cr.color_mode = 'RGB'
cr.hue_interpolation = 'NEAR'
cr.interpolation = 'LINEAR'
stop_actual = cr.elements.new(0.350)
# para crear el color verde claro queremos en RGB los valores 0.342,
0.274 0.202 y el 1.0 hace alusión a la opacidad siendo 1.0
completamente opaco y 0.0 sería transparente
stop_actual.color = [0.342, 0.274, 0.202, 1.0]
stop_actual = cr.elements.new(0.600)
# para crear el color verde oscuro queremos en RGB los valores 0.235,
0.209, 0.175 y el 1.0 hace alusión a la opacidad siendo 1.0
completamente opaco y 0.0 sería transparente
stop_actual.color = [0.235, 0.209, 0.175, 1.0]
removed_black = removed_white = False
# Esta parte de código es necesaria para que no se creen los puntos
de parada de los extremos (color negro puro y blanco puro)
for i in range(len(cr.elements) - 1, -1, -1):
```

```
    stop = cr.elements[i]
    if not removed_black and stop.position == 0 and
all([stop.color[i] == (0, 0, 0, 1)[i] for i in range(4)]):
        cr.elements.remove(stop)
        removed_black = True
    if not removed_white and stop.position == 1 and
all([stop.color[i] == (1, 1, 1, 1)[i] for i in range(4)]):
        cr.elements.remove(stop)
        removed_white = True
nodo_actual.location = (-235.032, 521.752)
nodo_actual.name = 'ColorRamp_Iris_Interior'
nodo_actual.width = 240.0
nodo_actual.inputs[0].default_value = 0.5
nodo_actual.outputs[0].default_value = [0.0, 0.0, 0.0, 0.0]
nodo_actual.outputs[1].default_value = 0.0

# ColorRamp_Iris_Circunferencia es el nodo encargado de las
tonalidades de azul en el interior del iris
nodo_actual = nodos.new(type='ShaderNodeValToRGB')
nodo_actual.color = (0.608, 0.608, 0.608)
cr = nodo_actual.color_ramp
cr.color_mode = 'RGB'
cr.hue_interpolation = 'NEAR'
cr.interpolation = 'EASE'
stop_actual = cr.elements.new(0.5)
# para crear el color negro queremos en RGB los valores 0.0, 0.0, 0.0
y el 1.0 hace alusión a la opacidad siendo 1.0 completamente opaco y
0.0 sería transparente
stop_actual.color = [0.0, 0.0, 0.0, 1.0]
stop_actual = cr.elements.new(0.550)
# para crear el color verde amarronado oscuro queremos en RGB los
valores 0.154, 0.116, 0.086 y el 1.0 hace alusión a la opacidad
siendo 1.0 completamente opaco y 0.0 sería transparente
```

```
stop_actual.color = [0.154, 0.116, 0.086, 1.0]
stop_actual = cr.elements.new(0.620)
# para crear el color verde amarronado oscuro pálido queremos en RGB
los valores 0.319, 0.287, 0.254 y el 1.0 hace alusión a la opacidad
siendo 1.0 completamente opaco y 0.0 sería transparente
stop_actual.color = [0.319, 0.287, 0.254, 1.0]
stop_actual = cr.elements.new(0.700)
# para crear el color verde amarronado claro queremos en RGB los
valores 0.279, 0.266, 0.238 y el 1.0 hace alusión a la opacidad
siendo 1.0 completamente opaco y 0.0 sería transparente
stop_actual.color = [0.279, 0.266, 0.238, 1.0]
stop_actual = cr.elements.new(0.760)
# para crear el color marrón oscuro pálido queremos en RGB los
valores 0.181, 0.107, 0.100 y el 1.0 hace alusión a la opacidad
siendo 1.0 completamente opaco y 0.0 sería transparente
stop_actual.color = [0.181, 0.107, 0.100, 1.0]
stop_actual = cr.elements.new(0.764)
# para crear el color negro queremos en RGB los valores 0.0, 0.0, 0.0
y el 1.0 hace alusión a la opacidad siendo 1.0 completamente opaco y
0.0 sería transparente
stop_actual.color = [0.0, 0.0, 0.0, 1.0]
removed_black = removed_white = False
# Esta parte de código es necesaria para que no se creen los puntos
de parada de los extremos (color negro puro y blanco puro)
for i in range(len(cr.elements) - 1, -1, -1):
    stop = cr.elements[i]
    if not removed_black and stop.position == 0 and
all([stop.color[i] == (0, 0, 0, 1)[i] for i in range(4)]):
        cr.elements.remove(stop)
        removed_black = True
    if not removed_white and stop.position == 1 and
all([stop.color[i] == (1, 1, 1, 1)[i] for i in range(4)]):
        cr.elements.remove(stop)
```

```
        removed_white = True
nodo_actual.location = (-237.990, 246.606)
nodo_actual.name = 'ColorRamp_Iris_Circunferencia'
nodo_actual.width = 244.526
nodo_actual.inputs[0].default_value = 0.5
nodo_actual.outputs[0].default_value = [0.0, 0.0, 0.0, 0.0]
nodo_actual.outputs[1].default_value = 0.0
```

Tabla. 24. Nodos Color Ramp encargados de tonalidad verde del iris

A continuación, plasmaremos el material "Iris_Verde" para que se pueda observar cómo se refleja cambiando los valores de las paradas de los nodos Color Ramp, cambiando el valor de los colores primarios, para obtener un iris verde como se muestra en la *Fig. 93. Comparación de iris creado con uno real.*, parecido al que se muestra a su derecha.

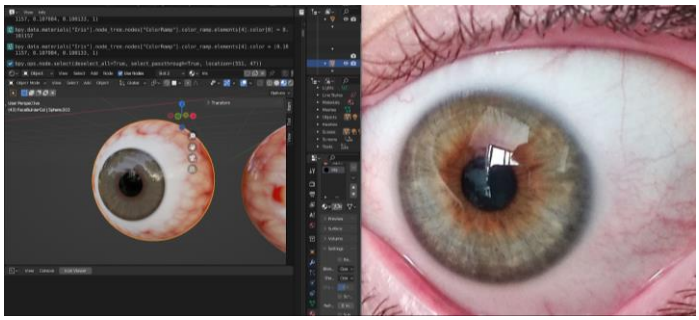

Fig. 93. Comparación de iris creado con uno real.

Con la ejecución del script podemos observar cómo repercute en el editor de listado, más concretamente si nos dirigimos a la sección de materiales de la escena, los creados como se refleja en la *Fig. 94. Materiales en editor listado*. Como ya hemos demostrado la ampliación consiste en una estructura bastante similar al material "Iris" anterior pero ahora se crean los materiales: Iris_Azul, Iris_Azul_1, Iris_Azul_2, Iris_Marron e Iris_Verde en vez de únicamente el material de iris azul que teníamos.

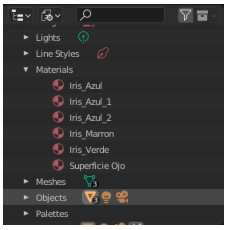

Fig. 94. Materiales en editor listado.

Al ejecutar el script *"Creación_Materiales_Ojo.py"* veremos que el material sólo se aplica en el ojo derecho del sujeto como podemos apreciar en la *Fig. 95. Editor de vista 3D tras ejecución de ampliación*, pero no es ningún problema, ya que este script sólo se encarga de la creación de los materiales y para ver que se aplican correctamente se aplica en dicha malla.

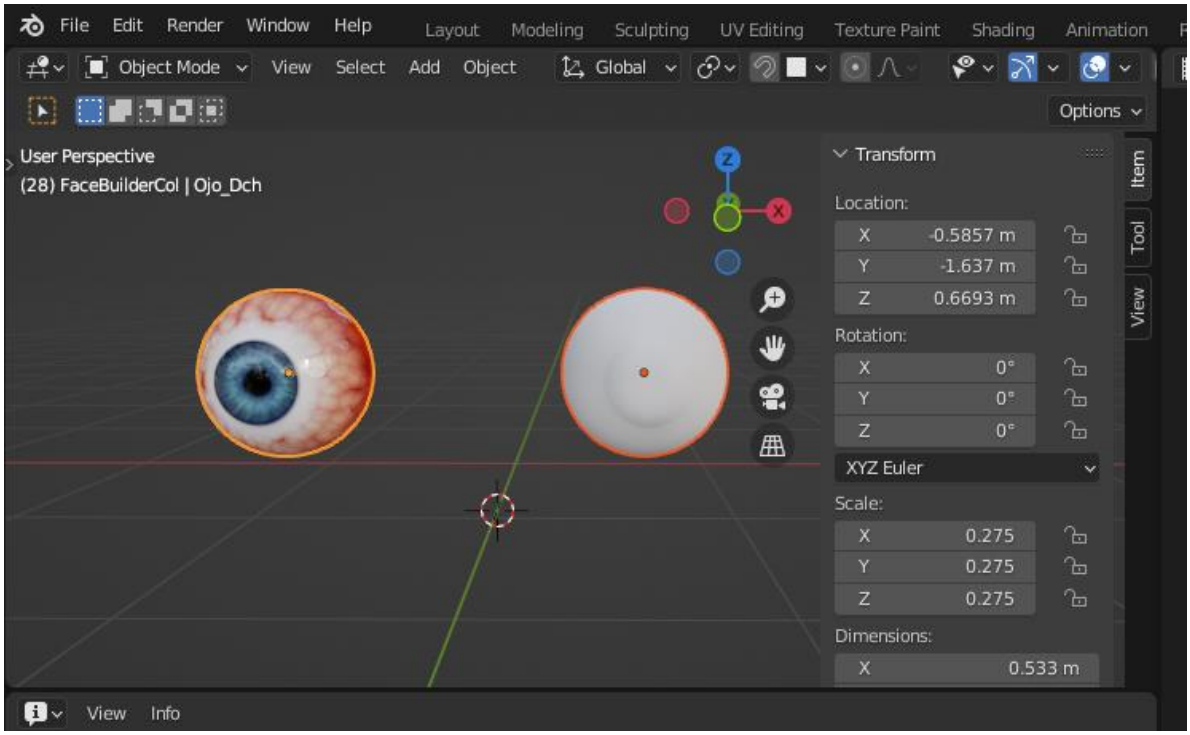

Fig. 95. Editor de vista 3D tras ejecución de ampliación.

7. Ampliación de script (Parte II) : "Aplicar_Materiales.py"

Tras la implementación ampliada de la creación de más materiales se ha optado por crear un botón el cual cuando se le clique eliminará los materiales que estén aplicadas en ese instante a las mallas que forman el ojo derecho y el ojo izquierdo, las cuales tienen los nombres de "Ojo_Dch" y "Ojo_Izq" respectivamente. A continuación, en la Tabla. *25. Código de script Aplicar_Materiales.py* se deja reflejado el código que genera dicho menú auxiliar con el UI Panel del botón que cambia los materiales de los ojos.

```
        ############ Aplicar Materiales A Los Ojos ############

import bpy
import numpy as np
from bpy.types import Panel, Operator

class ADDONNAME_PT_main_panel(Panel):
    bl_label = "Cambiar Color Ojos"
    bl_idname = "ADDONNAME_PT_main_panel"
    bl_space_type = 'VIEW_3D'
    bl_region_type = 'UI'
    bl_category = "Cambiar Color Ojos"

    def draw(self, context):
        layout = self.layout
        # addonname.myop_operator hace referencia a la parte del
botón Iris
        layout.operator("addonname.myop_operator")

class ADDONNAME_OT_my_op(Operator):
    # Iris para que se refleje el texto en el botón del panel
    bl_label = "Pulse aquí"
```

```
    bl_idname = "addonname.myop_operator"
    def execute(self, context):
        f = open("Indice_Material_Ojos.txt", "r")
        cont = int(f.read())
        f.close()

        # Seleccionar ojo izquierdo para aplicar materiales
        objectToSelect = bpy.data.objects["Ojo_Izq"]
        objectToSelect.select_set(True)
        bpy.context.view_layer.objects.active = objectToSelect
        ojo = bpy.context.collection.objects['Ojo_Izq']
        superficie = bpy.data.materials['Superficie Ojo']
        iris = bpy.data.materials[cont]

        #sumar índice para cambiar color
        f = open("Indice_Material_Ojos.txt", "w")
        if (cont == 4):
            cont = 0
        else:
            cont += 1
        f.write(str(cont))

        # Limpair materiales de mesh ojo
        ojo.data.materials.clear()
        ojo.data.materials.append(superficie)
        ojo.data.materials.append(iris)
        # Definimos los vértices que comprenden el iris en
vertices_Iris
        bpy.context.object.active_material_index = 1
        vertices_Iris = [15,30,45,60,75,90,105,120,136,151,167,182,
197,212,227,241,242,243,244,245,246,247,248,249,250,251,252,253,254,
255,256,257,258,259,260,261,262,263,264,265,266,267,268,269,270,271,
272,273,274,275,276,277,278,279,280,281,282,283,284,285,286,287,288,
```

```
289,290]

        # Bucle para seleccionar los vértices definidos en la lista
vertices_Iris
        for i in range(len(vertices_Iris)):
             ojo.data.vertices[vertices_Iris[i]].select = True

        bpy.ops.object.mode_set(mode = 'EDIT')

        # Asignamos el material Iris a los vértices seleccionados
        bpy.ops.object.material_slot_assign()
        bpy.ops.object.mode_set(mode = 'OBJECT')

        # Limpair materiales de mesh ojo
        objectToSelect = bpy.data.objects["Ojo_Dch"]
        objectToSelect.select_set(True)
        bpy.context.view_layer.objects.active = objectToSelect
        ojo = bpy.context.collection.objects['Ojo_Dch']
        ojo.data.materials.clear()
        ojo.data.materials.append(superficie)
        ojo.data.materials.append(iris)

        # Definimos los vértices que comprenden el iris en
vertices_Iris
        bpy.context.object.active_material_index = 1

        # Bucle para seleccionar los vértices definidos en la lista
vertices_Iris
        for i in range(len(vertices_Iris)):
             ojo.data.vertices[vertices_Iris[i]].select = True

        bpy.ops.object.mode_set(mode = 'EDIT')
```

```
        # Asignamos el material Iris a los vértices seleccionados
        bpy.ops.object.material_slot_assign()
        bpy.ops.object.mode_set(mode = 'OBJECT')

        return {'FINISHED'}


classes = [ADDONNAME_PT_main_panel, ADDONNAME_OT_my_op]

def register():
    for cls in classes:
        bpy.utils.register_class(cls)

def unregister():
    for cls in classes:
        bpy.utils.unregister_class(cls)

if __name__ == "__main__":
    register()
```

Tabla. 25. Código de script Aplicar_Materiales.py

Como en anteriores apartados de este proyecto ya se ha hablado de cómo funciona el panel de interfaz de usuario no nos detendremos en ello, pero sí en lo que hace la ejecución de clicar el botón. Para cambiar de material del color del iris de los ojos se intentó crear una variable global en el propio script, pero tiene un funcionamiento complejo al ser un UI Panel, ya que en sí es un menú que contiene ese botón. Por lo que al crear la variable global de un contador que fuera incrementado su valor por cada vez que se clicaba el botón se desechó ya que al acceder a la variable global daba error. Para solucionar este problema se ha desarrollado a través de la función de ***open()*** que ofrece Python, la cual contemplaremos en la *Tabla. 26. Código de script*

Aplicar_Materiales.py parte función open() , ya que, si le puedes pasar dos parámetros, el primero sería el parámetro que contiene el valor del nombre del fichero de texto al que queremos acceder y el segundo parámetro de dicha función sería especificar el permiso con el que queremos acceder a él.

```
    f = open("Indice_Material_Ojos.txt", "w")
    integer = 0
    f.write(str(integer))
    f.close()
```

Tabla. 26. Código de script Aplicar_Materiales.py parte función open()

Como podemos apreciar, en la primera línea del segmento de código de la tabla veintinueve podemos apreciar que estamos asignando una variable encargada de abrir el fichero con el nombre "Indice_Material_Ojos.txt" con el permiso W (write) que es el de escritura. Seguidamente crearemos la variable integer a 0 y escribiremos dicho valor en el fichero de texto para poder acceder al primer valor de los materiales. A continuación, en la parte de **execute(self, context)** la cual se ejecutará cuando se clique, en ella crearemos la variable cont la cual recibirá como valor del número que esté en el fichero de texto, en la primera iteración siempre será el 0 que hemos asignado fuera de la función de ejecución del botón.

```
        #sumar índice para cambiar color
               f = open("Indice_Material_Ojos.txt", "r")
                cont = int(f.read())
                f.close()
```

Tabla. 27. Asignar valor al contador

Seguidamente se seleccionará en el modo objeto una malla a través de la variable ojo = bpy.context.collection.objects['Ojo_Izq'] donde estaremos asignando en este caso el ojo izquierdo a la variable ojo. Igualmente, que se pueden asignar mallas se pueden asignar superficies a variables, por ejemplo, la variable nos valdría para todos los casos, pues el material de superficie del ojo siempre será el mismo (superficie =

bpy.data.materials['Superficie Ojo']) , pero para el del iris tendremos que indexar con la variable cont de la siguiente manera iris = bpy.data.materials[cont]. Pero para poder indexar correctamente en el material que debemos deberemos tener en cuenta la lista actual que forman los materiales la cual sería tal que así:

[Iris_Azul, Iris_Azul_1, Iris_Azul_2, Iris_Marron, Iris_Verde, SuperficieOjo]

Como podemos ver no todos los materiales que tenemos son para aplicar en los vértices del iris, pues como se ordena según el alfabeto, podemos ver que los materiales que aluden al iris se encuentran entre las posiciones cero hasta la cuatro. Es por esta razón por la que cuando el valor de la variable cont sea cuatro no queremos que se incremente a cinco, es por ello por lo que antes de escribir dicha actualización del contador pasará por el ciclo condicional que se muestra en la tala inferior.

```
        #sumar índice para cambiar color
                f = open("Indice_Material_Ojos.txt", "w")
                if (cont == 4):
                    cont = 0
                else:
                    cont += 1
                f.write(str(cont))
```

Tabla.28. Flujo condicional de variable cont

Una vez se tiene actualizado el valor del contador asignaremos el material del iris que se ha indexado, previamente habiendo aplicado el material de superficie del ojo sobre la malla, con la lista de vértices de la tabla 3, como ya hemos visto anteriormente. Seguidamente haremos lo mismo en la siguiente mesh que sería el ojo derecho como queda reflejado en las *figuras 96, 97 y 98*.

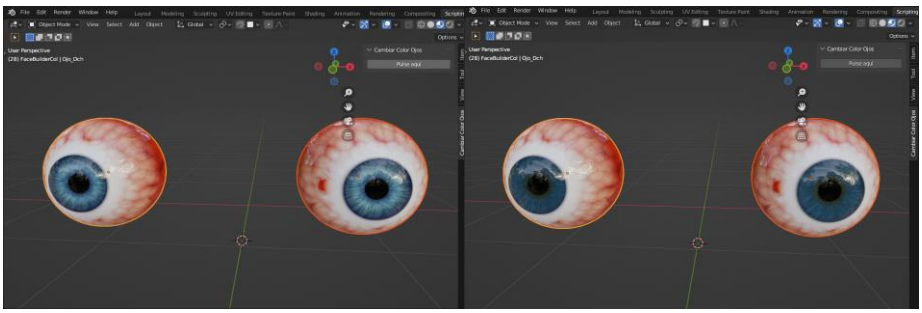

Fig. 96. Resultado de clicar botón de cambiar color ojos (1)

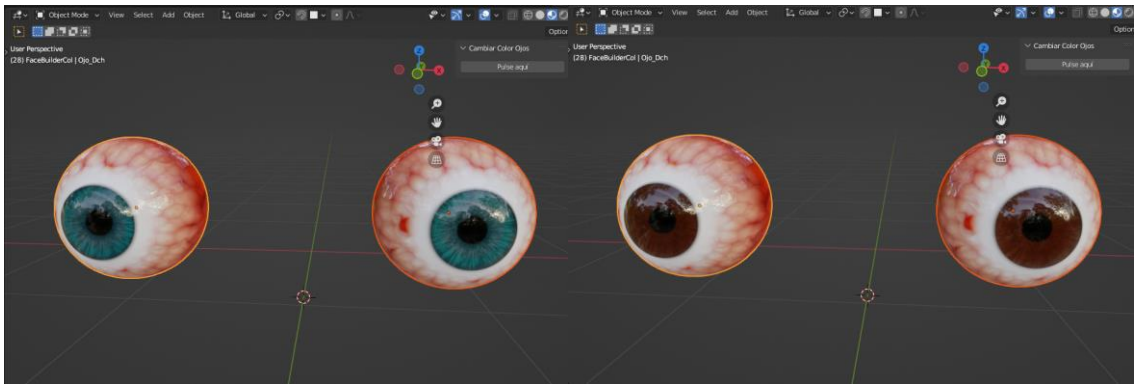

Fig. 97. Resultado de clicar botón de cambiar color ojos (2)

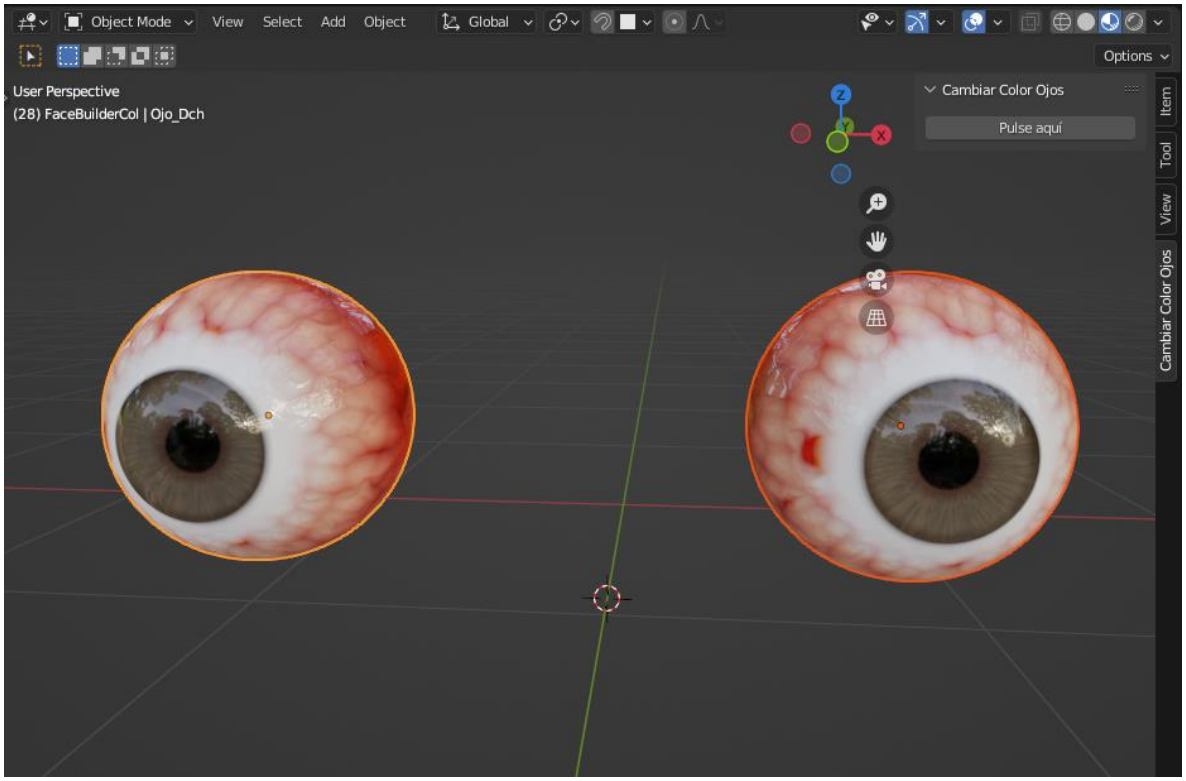

Fig. 98. Resultado de clicar botón de cambiar color ojos (3)

8. Ejecutar scripts de Python en Blender.

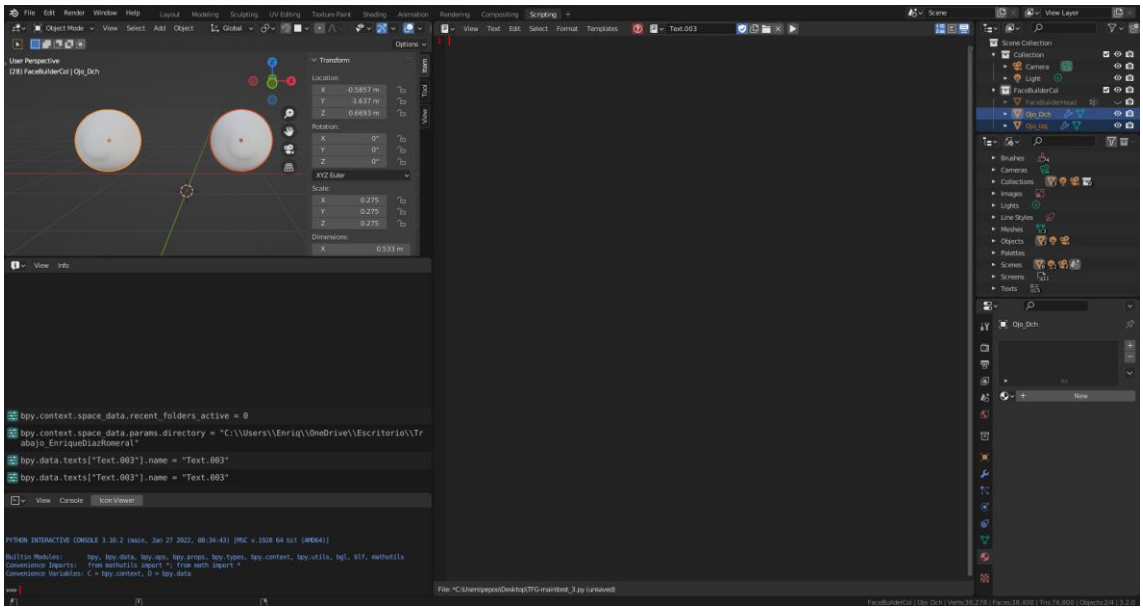

Fig. 99. Espacio de trabajo Scripting

Para la ejecución de los scripts es necesario que nos situemos en el espacio de trabajo Scripting, *Fig. 99. Espacio de trabajo Scripting*, en él podremos ver los editores :

- **Vista 3D:** que es la que contiene las dos mallas de los ojos, en las cuales no tienen ningún material aplicado.

- **Información:** justo el de debajo del anterior, en él podremos ver a nivel de código las aplicaciones que estamos haciendo a través de la aplicación de Blender en su interfaz gráfica.

- **Consola:** este editor es la consola “de entrada” de Blender, en ella podemos utilizar comando en lenguaje de Python que tendrán su repercusión y se verá reflejado en la aplicación. Es también útil para antes de hacer el script completo ir probando los comandos asegurando que funcionan.

- **Editor de texto:** es el más característico de este workspace, pues a través de él podemos programar en lenguaje Python, abrir scripts almacenados y ejecutarlos.
- **Listado:** En la parte superior de arriba donde podremos ver la escena, los objetos que la forman, los materiales creados y demás.

- **Propiedades:** es bastante útil para elegir ajustes que deseamos, por ejemplo, el motor de renderizado que queremos utilizar, si queremos refracciones en el renderizado, poder cambiar de slot de material o alguna propiedad de ellas y mucho más.

Para poder ejecutar el script que crea los materiales tendremos que seleccionar el icono que contiene una carpeta, en él se abrirá un explorador y podemos navegar por todas las rutas del equipo. Cunado encontremos el fichero le daremos a abrir y en el editor de texto se cargará el código creado para dicho script como se observa en la *figura 100*.

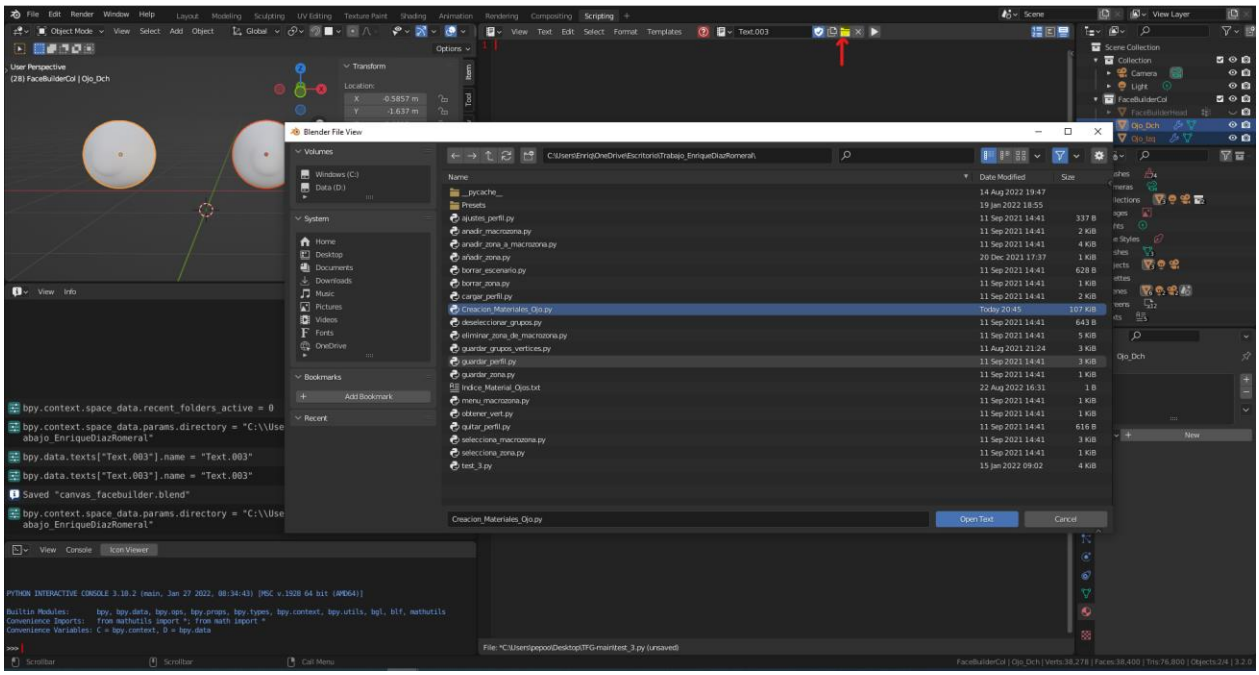

Fig. 100. Abrir script de una ruta.

Y una vez abierto será necesario ejecutarlo a través del icono que está dos botones a la derecha del de la carpeta, tiene un icono de botón play. Al clicar sobre él se ejecutará el script.

9. Enlazando las mallas creadas con herramienta del TFG anterior.

Una vez creadas las mallas de los ojos y aplicando sus respectivos materiales se pretende demostrar que, aunque sean mallas distintas se pueden agregar como zonas, esto nos permitirá seguir con la compatibilidad de seguir con el fundamento de la herramienta y poder avanzar en un futuro con transformaciones, pudiendo agregar más macrozonas, pero para ello se tendrán que crear. Para demostrar dicha compatibilidad con la herramienta creada por el anterior alumno, en estas imágenes optamos por crear una zona en cada malla, comprobando que queda reflejado. Para ver las zonas incluidas debemos seleccionar previamente la malla en la que se encuentra dichas zonas. Es necesario cambiar de vista de modo al de objeto y al seleccionar sobre cualquiera de las zonas que se vean en cada malla se podrá pulsar y se cambiará al modo de edición el cual nos permite ver los vértices, caras o aristas implicados en dicha sección facial. El funcionamiento con el anterior TFG se demuestra en las *figuras 101 y 102.*

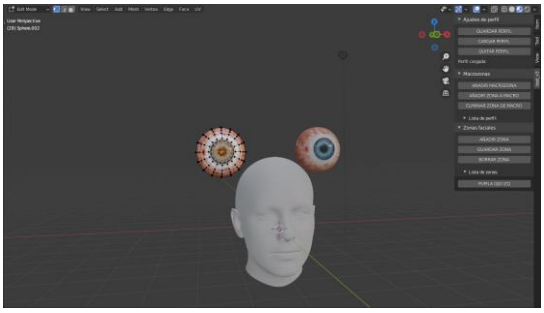

Fig. 101. Selección de ojo izquierda y de zona creada en dicha malla

Fig. 102. Selección mallas restantes y de respectivas zonas

10. Problemas encontrados

Esta sección está destinada a redactar los dilemas encontrados a lo largo del desarrollo del proyecto y de cómo se han resuelto.

10.1 Selección mal de vértices al guardar perfil y cargarlo.

Para hacer un retrato robot más realista se implementó una esfera como se puede ver en la *figura 103*, la cual serviría para hacer de ojo, pues antes de crear materiales de superficie del ojo y del iris se tuvo que investigar de si era viable para guardar compatibilidad con la herramienta creada en el anterior trabajo de fin de grado.

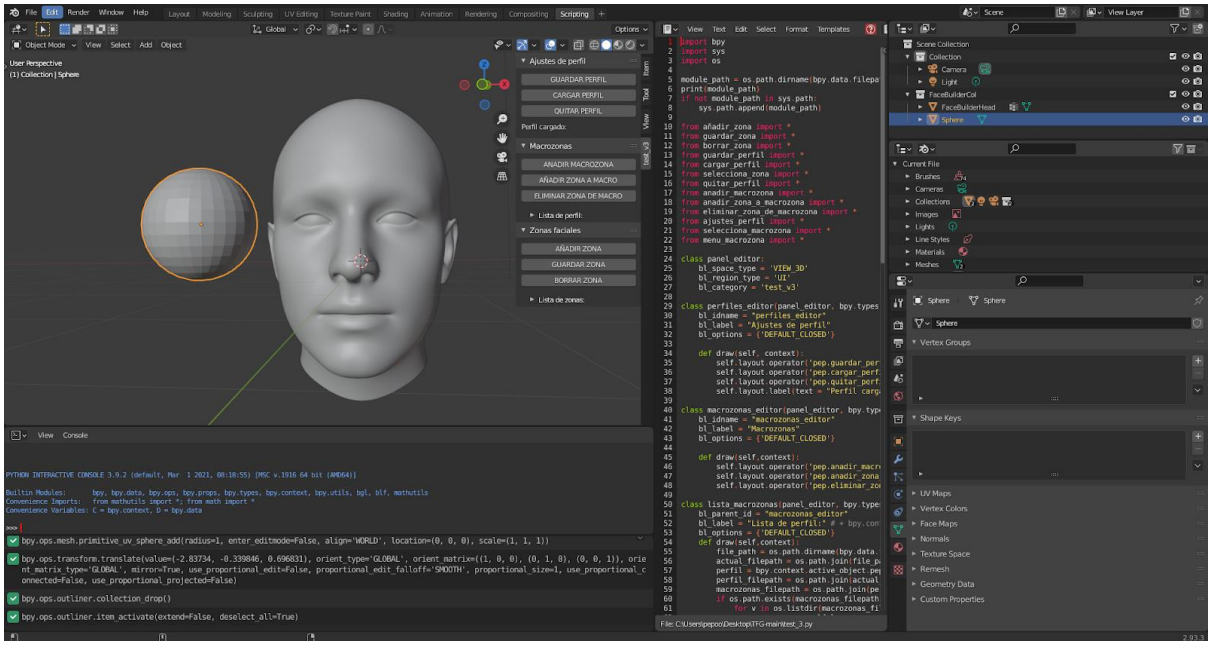

Fig. 103. Creación temporal de la malla ocular

Para probar su funcionamiento una vez creada la esfera se selecciona la malla en el modo objeto. Ya que como hemos visto en anteriores apartados es necesario estar en dicho modo para seleccionar malla y el modo edición para los nodos, las faceta y los bordes. Para ello adaptaremos de manera sencilla y rápida la mesh en el busto y una vez

colocado tendremos que comprobar el funcionamiento del menú. Es por ello por lo que colocaremos de manera superpuestas las mallas por si esto causara una mezcla de vértices que pueda llevar a error a la aplicación, pero comprobamos que los vértices de una malla sólo se ven reflejados en los propios objetos. Como podremos ver en la *Fig. 104. Creación temporal de zona de malla ocular temporal* se han seleccionado todos los vértices que forman el ojo izquierdo y se ha aplicado creado la zona "ojo_izq"

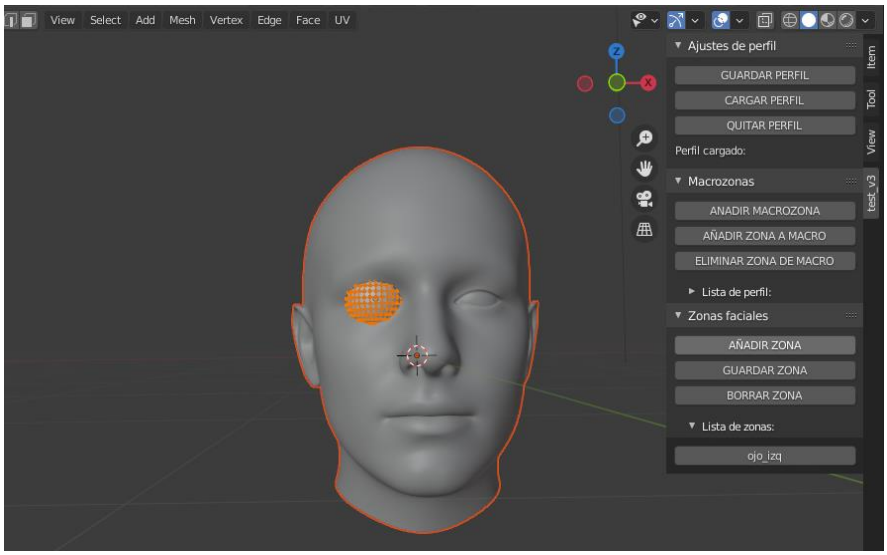

Fig. 104. Creación temporal de zona de malla ocular temporal

Una vez creada la zona de "ojo_izq" guardaremos el perfil para estudiar esta conducta como se aprecia en la *Fig. 105. Guardando perfil con ojo auxiliar*.

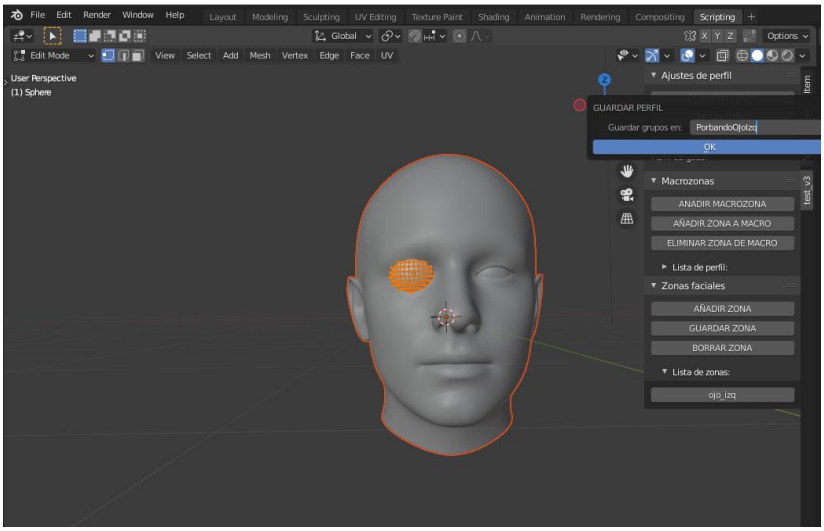

Fig. 105. Guardando perfil con ojo auxiliar

A continuación, cargaremos el perfil, seleccionaremos en el modo objeto la malla auxiliar que forma el ojo derecho desde el punto de vista del sujeto y clicaremos sobre el botón de la zona creado anteriormente. Como se puede observar en la *Fig. 106. Cargando perfil con ojo auxiliar* los vértices seleccionados son de la mesh de la cara en vez de la ocular.

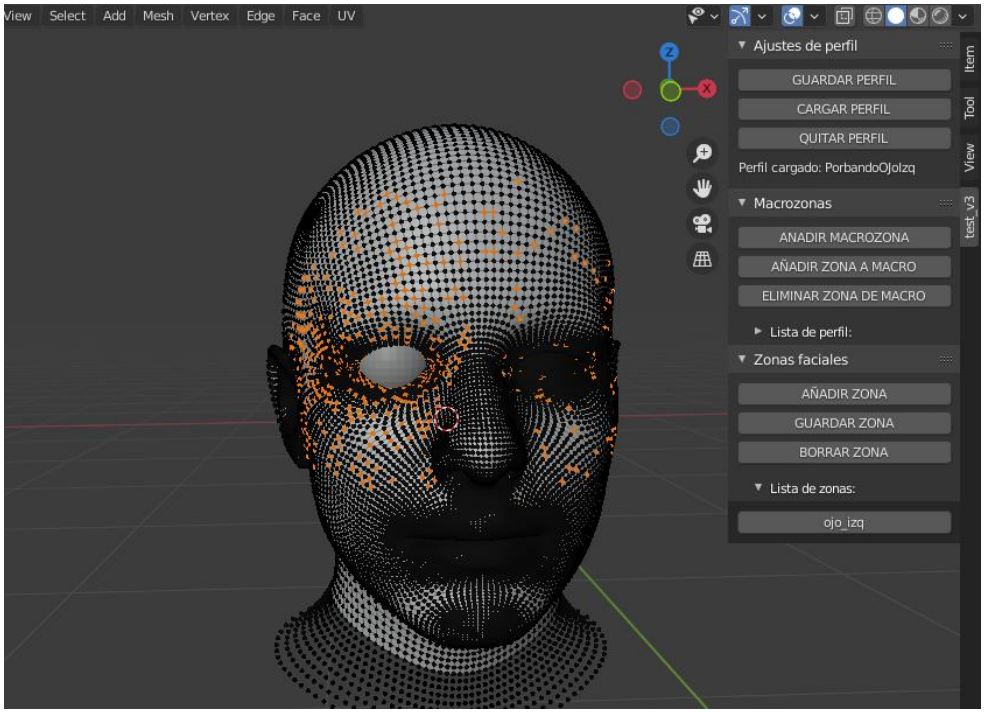

Fig. 106. Cargando perfil con ojo auxiliar

Para resolver este problema, se ha optado por, en vez de que Blender al seleccionar ,en nuestro menú generado por el script, cargar un perfil, cree la esfera nos decantaremos por crear en el propio archivo *"canvas_facebuilder.blend"* a través del entorno gráfico que nos ofrece Blender y guardarremos dicho proyecto. Resolviendo así el malfuncionamiento que se ha reflejado, haciendo que los vértices de la mesh queden seleccionados correctamente, como hemos podido comprobar en las figuras 101 y 102 de este documento en el apartado "Enlazando las mallas creadas con herramienta del TFG anterior".

10.2 Conocimiento de los índices que identifican a los vértices.

Como ya sabemos una malla está formada por vértices, aristas y caras los cuales crean figuras geométricas, cuanto más contengan más definidos estarán las zonas de la malla que por ejemplo tengan una curva en ella. Para poder seleccionar los vértices que necesitamos a través de código la aplicación de Blender se optó por utilizar la navegación tridimensional que nos proporciona la aplicación. Para ello en configuración ha sido necesario habilitar la función de desarrollador (developer) y posteriormente para poder ver los vértices, aristas o caras que tienen dichas mallas es necesario irnos al modo de edición, pues es necesario para actuar con estos elementos de la mesh. Una vez nos encontremos en dicho modo tendremos que desplegar el menú de "viewport overlays" y habilitar el elemento de índices. Esto nos permitirá ver una capa superpuesta la cual mostrará los índices de los vértices tal como refleja en la *figura 107.*

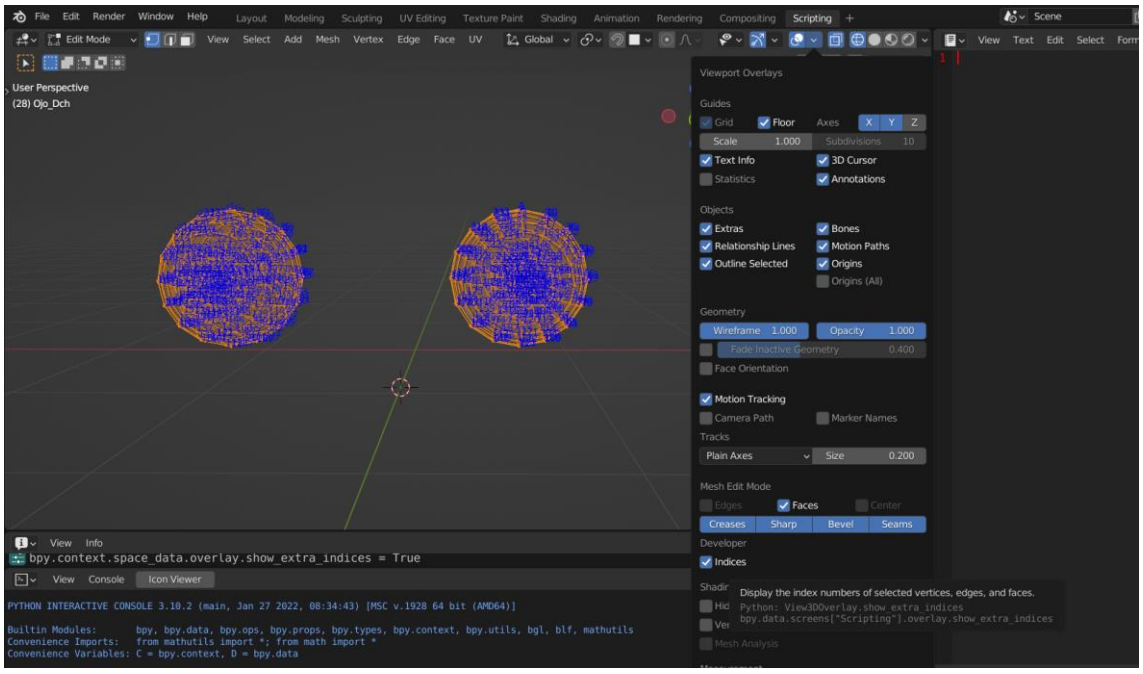

Fig. 107. Habilitar la vista de índices de los vértices

Viendo reflejado el resultado que tiene activar el elemento que nos muestra los índices de cada vértice seleccionado se optó por crear un script auxiliar *"imprimir_vertices_seleccionados.py"* mostrado en la segunda tabla de este documento,

ya que sin la creación de dicho script el proceso de recopilar los índices de los vértices necesarios para aplicar el material sería un proceso tedioso e incluso alargando el tiempo de dicha tarea. La recopilación de los índices involucrados en el iris ha servido para los scripts posteriores en los que tienen una parte específica de aplicar el material del iris en las mallas de los ojos.

Para poder ver el resultado que se imprime es necesario abrir la consola como se indica en la figura número setenta y tres, y una vez impresos los números enteros que identifican a cada vértice se ha llevado a un editor de documentos de texto, en nuestro caso Notepad++ el cual a través del modo de búsqueda extendido como son el retorno de línea, los espacios y demás se ha podido crear la lista de dichos elementos de forma rápida, sencilla y eficaz ya que ha funcionado en los scripts principales de este proyecto, como se observa en la *Fig. 108. Habilitar modo de búsqueda extendido en Notepad++*.

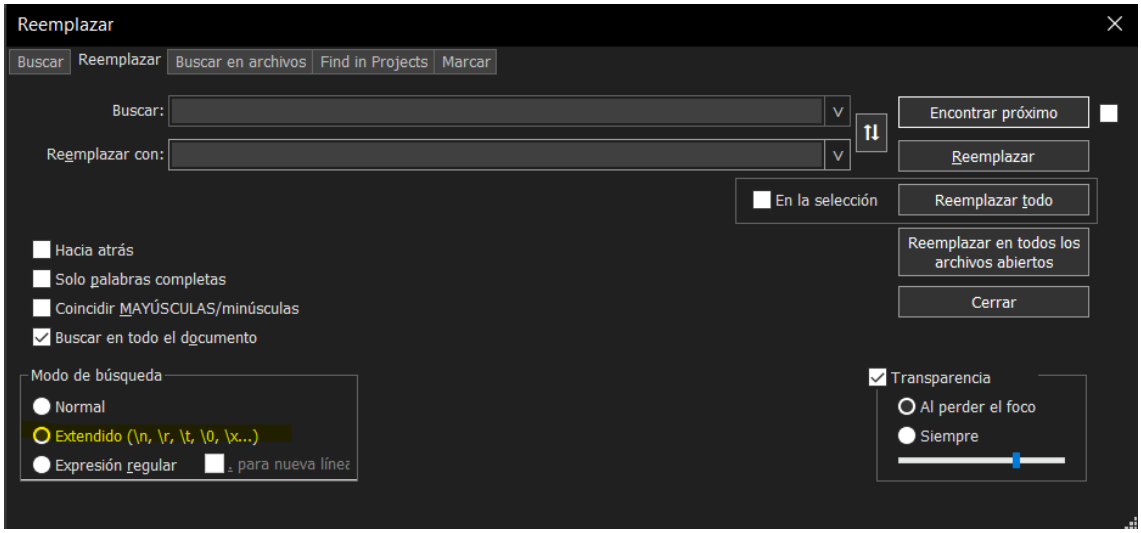

Fig. 108. Habilitar modo de búsqueda extendido en Notepad++

10.3 Variable global en script de Python de interfaz de usuario para cambiar material.

Este problema ha surgido en la implementación del script *"Aplicar_Materiales.py"* ya que se pretendía crear una variable tipo número entero para utilizarse como indexador de la lista perteneciente a los materiales creados en Blender, el cual se iría incrementando hasta llegar al número de índice que pertenece al material de la superficie del ojo con un flujo condicional similar al de la tabla treinta y uno. El problema radica en que si la variable se creaba dentro de la sección `def execute(self, context):` cada vez que se presionara el botón el valor del contador necesitaba tener un valor, el cual la primera vez se pretende que sea 0, por lo que sería siempre dicho valor el cual no cambiaría en ninguna ejecución del botón. Es por ello por lo que se intentó crear una variable global encima de dicha estructura, la cual cuando se llamaba a la variable dentro del execute devolvía un error en que esa variable no estaba creada. Para solucionar este problema se optó por utilizar la función open() de Python vista en la tabla veintinueve, la cual crea un fichero de texto el cual se accederá en modo de escritura cuando se quiera asignar un valor a la variable como podría ser la inicialización de esta a cero, cuando se quiera incrementar o cuando llegue al valor cuatro que se reseteé a cero para que no acceda al material de superficie del ojo. A continuación, dejaré el flujo de cómo el script primero escribe un 0 y después actualiza el valor del contador a través de lecturas y escrituras en la *figura 109*.

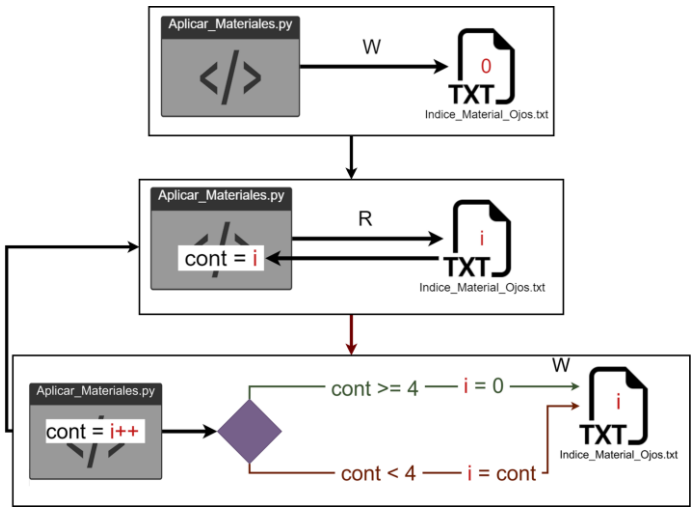

Fig. 109. Flujo del funcionamiento open()

11. Scripts involucrados en el proyecto.

En este apartado se reflejan los scripts anteriormente comentados además de estar en este repositorio de GitHub:

https://github.com/enriquediazr/Blender_Material_Eyes_Scripts

11.1 "Imprimir_vertices_seleccionados.py"

```
                # imprimir_vertices_seleccionados.py
import bpy
import numpy as np
from bpy.types import Panel, Operator

class ADDONNAME_PT_main_panel(Panel):
    bl_label = "Imprimir Vértices Seleccionados"
    bl_idname = "ADDONNAME_PT_main_panel"
    bl_space_type = 'VIEW_3D'
    bl_region_type = 'UI'
    bl_category = "Vértices"

    def draw(self, context):
        layout = self.layout
        layout.operator("addonname.imprimir_vertices")

class ADDONNAME_OT_my_op(Operator):
    bl_label = "Imprimir"
    bl_idname = "addonname.imprimir_vertices"
    def execute(self, context):
        mode = bpy.context.active_object.mode
        # Mantener el control en el modo anterior
        bpy.ops.object.mode_set(mode='OBJECT')
        # Pasar al modo objeto para actualizar los vértices
seleccionados

        obj = bpy.context.object
```

```
        # Obtener el objeto actualmente seleccionado
        sel = np.zeros(len(obj.data.vertices), dtype=bool)
        # Create a numpy array with empty values for each vertex

        obj.data.vertices.foreach_get('select', sel)
        # Crear una matriz numpy con valores vacíos para cada vértice
        print ('Los índices de los vértices son:')
        for ind in np.where(sel==True)[0]:
            # Recorrer en bucle cada uno de los vértices seleccionados
actualmente
            v = obj.data.vertices[ind]
            print('{}'.format(v.index))

        bpy.ops.object.mode_set(mode=mode)
        # Mantener el control en el modo anterior

        return {'FINISHED'}

classes = [ADDONNAME_PT_main_panel, ADDONNAME_OT_my_op]

def register():
    for cls in classes:
        bpy.utils.register_class(cls)

def unregister():
    for cls in classes:
        bpy.utils.unregister_class(cls)

if __name__ == "__main__":
    register()
```

Tabla 29. Código imprimir_vertices_seleccionados.py

11.2 “Seleccionar_vertices_ojo.py”

```
                        # seleccionar_vertices_ojo.py
import bpy
import numpy as np
from bpy.types import Panel, Operator


class ADDONNAME_PT_main_panel(Panel):
    bl_label = "Seleccionar Vertices"
    bl_idname = "ADDONNAME_PT_main_panel"
    bl_space_type = 'VIEW_3D'
    bl_region_type = 'UI'
    bl_category = "Seleccionar Vertices"

    def draw(self, context):
        layout = self.layout
        # addonname.myop_operator hace referencia a la parte del botón Iris
        layout.operator("addonname.myop_operator")
        # addonname.myop_operator2 hace referencia a la parte del botón Superficie Ojo
        layout.operator("addonname.myop_operator2")


class ADDONNAME_OT_my_op(Operator):
    # Iris para que se refleje el texto en el botón del panel
    bl_label = "Iris"
    bl_idname = "addonname.myop_operator"

    def execute(self, context):
        # Array con los índices de los vértices obtenidos con el programa
         imprimir_vertices_seleccionados.py
        vertices_Iris = [15,30,45,60,75,90,105,120,136,151,167,182,197,212,
227,241,242,243,244,245,246,247,248,249,250,251,252,253,254,255,256,257,
```

```
258,259,260,261,262,263,264,265,266,267,268,269,270,271,272,273,274,
275,276,277,278,279,280,281,282,283,284,285,286,287,288,289,290]
        bpy.ops.object.mode_set(mode = 'OBJECT')
        obj = bpy.context.active_object
        bpy.ops.object.mode_set(mode = 'EDIT')
        bpy.ops.mesh.select_mode(type="VERT")
        # Deseleccionamos los vértices que previamente hayan podido ser
          seleccionados por el usuario en la aplicación
        bpy.ops.mesh.select_all(action = 'DESELECT')
        bpy.ops.object.mode_set(mode = 'OBJECT')
        # Seleccionamos los vértices definidos en el Array
        for i in range(len(vertices_Iris)):
             obj.data.vertices[vertices_Iris[i]].select = True

        bpy.ops.object.mode_set(mode = 'EDIT')

        return {'FINISHED'}

class ADDONNAME_OT_my_op2(Operator):
    # Superficie Ojo para que se refleje el texto en el botón del panel
    bl_label = "Superficie Ojo"
    bl_idname = "addonname.myop_operator2"

    def execute(self, context):
        # Array con los índices de los vértices obtenidos con el programa
imprimir_vertices_seleccionados.py tras seleccionar solamente la superficie de la mesh
        vertices_Superficie = [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,
18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,
43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,64,65,66,67,
68,69,70,71,72,73,74,75,76,77,78,79,80,81,82,83,84,85,86,87,88,89,90,91,92,
93,94,95,96,97,98,99,100,101,102,103,104,105,106,107,108,109,110,111,112,113,
114,115,116,117,118,119,120,121,122,123,124,125,126,127,128,129,130,131,132,
133,134,135,136,137,138,139,140,141,142,143,144,145,146,147,148,149,150,151,
```

```
152,153,154,155,156,157,158,159,160,161,162,163,164,165,166,167,168,169,170,
171,172,173,174,175,176,177,178,179,180,181,182,183,184,185,186,187,188,189,
190,191,192,193,194,195,196,197,198,199,200,201,202,203,204,205,206,207,208,
209,210,211,212,213,214,215,216,217,218,219,220,221,222,223,224,225,226,227,
228,229,230,231,232,233,234,235,236,237,238,239,240,241]
        bpy.ops.object.mode_set(mode = 'OBJECT')
        obj = bpy.context.active_object
        bpy.ops.object.mode_set(mode = 'EDIT')
        bpy.ops.mesh.select_mode(type="VERT")
        # Deseleccionamos los vértices que previamente hayan podido ser
seleccionados por el usuario en la aplicación
        bpy.ops.mesh.select_all(action = 'DESELECT')
        bpy.ops.object.mode_set(mode = 'OBJECT')
        # Seleccionamos los vértices definidos en el Array
        for i in range(len(vertices_Superficie)):
             obj.data.vertices[vertices_Superficie[i]].select = True

        bpy.ops.object.mode_set(mode = 'EDIT')

        return {'FINISHED'}

classes = [ADDONNAME_PT_main_panel, ADDONNAME_OT_my_op,ADDONNAME_OT_my_op2]

def register():
    for cls in classes:
        bpy.utils.register_class(cls)

def unregister():
    for cls in classes:
        bpy.utils.unregister_class(cls)
if __name__ == "__main__":
    register()
```

Tabla 30. Código de seleccionar_vertices_ojo.py

11.3 “Creacion_Materiales_Ojo.py”

```
                                # Creacion_Materiales_Ojo.py
import bpy
############################## SUPERFICIE OJO ##############################
obj = bpy.context.object  # Asignación de la mesh seleccionada con el puntero
mat = bpy.data.materials.new(name='Superficie Ojo')  # Creación del material
# Configuración del material
obj.data.materials.append(mat)  # Asignación del material
obj = bpy.context.active_object

new_mat = bpy.data.materials.get('Superficie Ojo')
if not new_mat:
    new_mat = bpy.data.materials.new('Superficie Ojo')

# Activamos las refracción para poder ver la pupila
bpy.context.object.active_material.use_screen_refraction = True
# habilitaremos el uso de nodos, ya que sin ello no podríamos crear ninguno
new_mat.use_nodes = True
# Definimos la estructura que tendrán los nodos y los enlaces para su creación
node_tree = new_mat.node_tree
nodos = node_tree.nodes
nodos.clear()

enlaces = node_tree.links
enlaces.clear()
######### Nodos #########
# Frame_S será el nodo que englobe el nodo Mapping, el nodo Voronoi Texture que da textura
voronoi
a los vasos sanguíneos de menor longitud y el nodo ColorRamp
nodo_actual = nodos.new(type='NodeFrame')
nodo_actual.color = (0.608, 0.608, 0.608)
nodo_actual.label = 'S'
nodo_actual.label_size = 20
nodo_actual.location = (-1756.5, 1353.1)
nodo_actual.name = 'Frame_S'
nodo_actual.select = False
nodo_actual.width = 800

# Frame_M será el nodo que englobe el nodo Mapping, el nodo Voronoi Texture
 que da textura voronoi a los vasos sanguíneos de longitud medio y el nodo ColorRamp
nodo_actual = nodos.new(type='NodeFrame')
nodo_actual.color = (0.608, 0.608, 0.608)
nodo_actual.label = 'M'
nodo_actual.label_size = 20
nodo_actual.location = (-1787.8, 961)
```

```
nodo_actual.name = 'Frame_M'
nodo_actual.select = False
nodo_actual.width = 800

# Frame_L será el nodo que englobe el nodo Mapping, el nodo Voronoi Texture
que da textura Voronoi a los vasos sanguíneos de longitud más larga y el nodo ColorRamp
nodo_actual = nodos.new(type='NodeFrame')
nodo_actual.color = (0.608, 0.608, 0.608)
nodo_actual.label = 'L'
nodo_actual.label_size = 20
nodo_actual.location = (-1763.4, 613.1)
nodo_actual.name = 'Frame_L'
nodo_actual.select = False
nodo_actual.width = 800
# Mapping_S el nodo mapping de los vasos sanguíneos de corta distancia es de tipo 'NORMAL'
y la escala es de (1,2,1) haciendo referencia a (X,Y,Z)
nodo_actual = nodos.new(type='ShaderNodeMapping')
nodo_actual.color = (0.608, 0.608, 0.608)
nodo_actual.location = (-33.8, -25.8)
nodo_actual.name = 'Mapping_S'
# Asignamos el nodo padre, ya que Mapping_S es un nodo que se encuentra en el interior de
Frame_S
parent = nodos.get('Frame_S')
if parent:
    nodo_actual.parent = parent
    while True:
        nodo_actual.location += parent.location
        if parent.parent:
            parent = parent.parent
        else:
            break
nodo_actual.select = False
nodo_actual.vector_type = 'NORMAL'
nodo_actual.width = 240.0
nodo_actual.inputs[0].default_value = [0.0, 0.0, 0.0]
nodo_actual.inputs[1].default_value = [0.0, 0.0, 0.0]
nodo_actual.inputs[2].default_value = [0.0, 0.0, 0.0]
nodo_actual.inputs[3].default_value = [1.0, 2.0, 1.0]
nodo_actual.outputs[0].default_value = [0.0, 0.0, 0.0]

# Voronoi_Texture_M es el encargado de como vemos los vasos sanguíneos de longitud media
nodo_actual = nodos.new(type='ShaderNodeTexVoronoi')
nodo_actual.color = (0.608, 0.608, 0.608)
nodo_actual.distance = 'EUCLIDEAN'
nodo_actual.feature = 'F1'
```

```
nodo_actual.location = (316.1, -30.7)
nodo_actual.name = 'Voronoi_Texture_M'
parent = nodos.get('Frame_M')
# Asignamos el nodo padre, ya que Voronoi_Texture_M es un nodo que se encuentra
 en el interior de Frame_M
if parent:
    nodo_actual.parent = parent
    while True:
        nodo_actual.location += parent.location
        if parent.parent:
            parent = parent.parent
        else:
            break
nodo_actual.select = False
nodo_actual.voronoi_dimensions = '3D'
nodo_actual.width = 140.0
nodo_actual.inputs[0].default_value = [0.0, 0.0, 0.0]
nodo_actual.inputs[1].default_value = 0.0
nodo_actual.inputs[2].default_value = 6.0
nodo_actual.inputs[3].default_value = 1.0
nodo_actual.inputs[4].default_value = 0.5
nodo_actual.inputs[5].default_value = 1.0
nodo_actual.outputs[0].default_value = 0.0
nodo_actual.outputs[1].default_value = [0.0, 0.0, 0.0, 0.0]
nodo_actual.outputs[2].default_value = [0.0, 0.0, 0.0]
nodo_actual.outputs[3].default_value = 0.0
nodo_actual.outputs[4].default_value = 0.0

# Mapping_M el nodo mapping de los vasos sanguíneos de distancia media es de tipo 'NORMAL'
y la escala es de (1,2,1) haciendo referencia a (X,Y,Z)
nodo_actual = nodos.new(type='ShaderNodeMapping')
nodo_actual.color = (0.608, 0.608, 0.608)
nodo_actual.location = (4.8, -22.9)
nodo_actual.name = 'Mapping_M'
# Asignamos el nodo padre, ya que Mapping_M es un nodo que se encuentra
en el interior del nodo de Frame_M
parent = nodos.get('Frame_M')
if parent:
    nodo_actual.parent = parent
    while True:
        nodo_actual.location += parent.location
        if parent.parent:
            parent = parent.parent
        else:
            break
```

```
nodo_actual.select = False
nodo_actual.vector_type = 'NORMAL'
nodo_actual.width = 240.0
nodo_actual.inputs[0].default_value = [0.0, 0.0, 0.0]
nodo_actual.inputs[1].default_value = [0.0, 0.0, 0.0]
nodo_actual.inputs[2].default_value = [0.0, 0.0, 0.0]
nodo_actual.inputs[3].default_value = [1.0, 2.0, 1.0]
nodo_actual.outputs[0].default_value = [0.0, 0.0, 0.0]

# ColorRamp_M es el nodo encargado de proporcionar el color
 rojo a esos vasos sanguíneos de longitud media
nodo_actual = nodos.new(type='ShaderNodeValToRGB')
nodo_actual.color = (0.608, 0.608, 0.608)
cr = nodo_actual.color_ramp
cr.color_mode = 'RGB'
cr.hue_interpolation = 'NEAR'
cr.interpolation = 'LINEAR'
# creamos el punto que delimita la parte "más blanca"
stop_actual = cr.elements.new(0.136)
# para crear el color blanco queremos en RGB los valores 1.0, 1.0, 1.0 y el 1.0
hace alusión a la opacidad siendo 1.0 completamente opaco y 0.0 sería transparente
stop_actual.color = [1.0, 1.0, 1.0, 1.0]
# creamos el punto que delimita la parte "más roja"
stop_actual = cr.elements.new(1.0)
# para crear el color rojo queremos en RGB los valores 0.314, 0.005, 0.0 y el 1.0
hace alusión a la opacidad siendo 1.0 completamente opaco y 0.0 sería transparente
stop_actual.color = [0.314, 0.005, 0.0, 1.0]
removed_black = removed_white = False
for i in range(len(cr.elements) - 1, -1, -1):
    stop = cr.elements[i]
    if not removed_black and stop.position == 0 and
all([stop.color[i] == (0, 0, 0, 1)[i] for i in range(4)]):
        cr.elements.remove(stop)
        removed_black = True
    if not removed_white and stop.position == 1 and
all([stop.color[i] == (1, 1, 1, 1)[i] for i in range(4)]):
        cr.elements.remove(stop)
        removed_white = True
nodo_actual.location = (503.482, -26.962)
nodo_actual.name = 'ColorRamp_M'
# Asignamos el nodo padre, ya que ColorRamp_M es un nodo que se
encuentra en el interior del nodo de Frame_M
parent = nodos.get('Frame_M')
if parent:
    nodo_actual.parent = parent
```

```
    while True:
        nodo_actual.location += parent.location
        if parent.parent:
            parent = parent.parent
        else:
            break
nodo_actual.select = False
nodo_actual.width = 240.0
nodo_actual.inputs[0].default_value = 0.5
nodo_actual.outputs[0].default_value = [0.0, 0.0, 0.0, 0.0]
nodo_actual.outputs[1].default_value = 0.0

# ColorRamp_S es el nodo encargado de proporcionar el color rojo a
esos vasos sanguíneos de longitud media
nodo_actual = nodos.new(type='ShaderNodeValToRGB')
nodo_actual.color = (0.608, 0.608, 0.608)
cr = nodo_actual.color_ramp
cr.color_mode = 'RGB'
cr.hue_interpolation = 'NEAR'
cr.interpolation = 'LINEAR'
stop_actual = cr.elements.new(0.358)
# para crear el color blanco queremos en RGB los valores 1.0, 1.0, 1.0 y
el 1.0 hace alusión a la opacidad siendo 1.0 completamente opaco y 0.0 sería transparente
stop_actual.color = [1.0, 1.0, 1.0, 1.0]
stop_actual = cr.elements.new(1.0)
# para crear el color rojo queremos en RGB los valores 0.314, 0.005, 0.0 y
el 1.0 hace alusión a la opacidad siendo 1.0 completamente opaco y 0.0 sería transparente
stop_actual.color = [0.314, 0.005, 0.0, 1.0]
removed_black = removed_white = False
# Esta parte de código es necesaria para que no se
creen los puntos de parada de los extremos (color negro puro y blanco puro)
for i in range(len(cr.elements) - 1, -1, -1):
    stop = cr.elements[i]
    if not removed_black and stop.position == 0 and
all([stop.color[i] == (0, 0, 0, 1)[i] for i in range(4)]):
        cr.elements.remove(stop)
        removed_black = True
    if not removed_white and stop.position == 1 and
all([stop.color[i] == (1, 1, 1, 1)[i] for i in range(4)]):
        cr.elements.remove(stop)
        removed_white = True
nodo_actual.location = (464.786, -29.943)
nodo_actual.name = 'ColorRamp_S'
# Asignamos el nodo padre, ya que ColorRamp_S es un
nodo que se encuentra en el interior del nodo de Frame_S
```

```
parent = nodos.get('Frame_S')
if parent:
    nodo_actual.parent = parent
    while True:
        nodo_actual.location += parent.location
        if parent.parent:
            parent = parent.parent
        else:
            break
nodo_actual.select = False
nodo_actual.width = 240.0
nodo_actual.inputs[0].default_value = 0.5
nodo_actual.outputs[0].default_value = [0.0, 0.0, 0.0, 0.0]
nodo_actual.outputs[1].default_value = 0.0

# Mix_M_L es el nodo encargado de mezclar los nodos colorRamp de M y
de L con el tipo "DARKEN"
nodo_actual = nodos.new(type='ShaderNodeMixRGB')
nodo_actual.blend_type = 'DARKEN'
nodo_actual.color = (0.608, 0.608, 0.608)
nodo_actual.location = (-864.344, 929.187)
nodo_actual.name = 'Mix_M_L'
nodo_actual.select = False
nodo_actual.use_alpha = False
nodo_actual.use_clamp = False
nodo_actual.width = 140.0
nodo_actual.inputs[0].default_value = 1.0

# Mapping_L el nodo mapping de los vasos sanguíneos de corta distancia es de tipo 'NORMAL'
y la escala es de (1,2,1) haciendo referencia a (X,Y,Z)
nodo_actual = nodos.new(type='ShaderNodeMapping')
nodo_actual.color = (0.608, 0.608, 0.608)
nodo_actual.location = (-19.012, -46.767)
nodo_actual.name = 'Mapping_L'
# Asignamos el nodo padre, ya que Mapping_L es un nodo que se
encuentra en el interior del nodo de Frame_L
parent = nodos.get('Frame_L')
if parent:
    nodo_actual.parent = parent
    while True:
        nodo_actual.location += parent.location
        if parent.parent:
            parent = parent.parent
        else:
            break
```

```
nodo_actual.select = False
nodo_actual.vector_type = 'NORMAL'
nodo_actual.width = 240.0
nodo_actual.inputs[0].default_value = [0.0, 0.0, 0.0]
nodo_actual.inputs[1].default_value = [0.0, 0.0, 0.0]
nodo_actual.inputs[2].default_value = [0.0, 0.0, 0.0]
nodo_actual.inputs[3].default_value = [1.0, 4.0, 1.0]
nodo_actual.outputs[0].default_value = [0.0, 0.0, 0.0]
# Voronoi_Texture_L es el encargado de como vemos los vasos
sanguíneos de larga longitud
nodo_actual = nodos.new(type='ShaderNodeTexVoronoi')
nodo_actual.color = (0.608, 0.608, 0.608)
nodo_actual.distance = 'EUCLIDEAN'
nodo_actual.feature = 'F1'
nodo_actual.location = (292.31591796875, -54.62493896484375)
nodo_actual.name = 'Voronoi_Texture_L'
# Asignamos el nodo padre, ya que Mapping_L es un nodo que se encuentra
en el interior del nodo de Frame_L
parent = nodos.get('Frame_L')
if parent:
    nodo_actual.parent = parent
    while True:
        nodo_actual.location += parent.location
        if parent.parent:
            parent = parent.parent
        else:
            break
nodo_actual.select = False
nodo_actual.voronoi_dimensions = '3D'
nodo_actual.width = 140.0
nodo_actual.inputs[0].default_value = [0.0, 0.0, 0.0]
nodo_actual.inputs[1].default_value = 0.0
nodo_actual.inputs[2].default_value = 3.0
nodo_actual.inputs[3].default_value = 1.0
nodo_actual.inputs[4].default_value = 0.5
nodo_actual.inputs[5].default_value = 1.0
nodo_actual.outputs[0].default_value = 0.0
nodo_actual.outputs[1].default_value = [0.0, 0.0, 0.0, 0.0]
nodo_actual.outputs[2].default_value = [0.0, 0.0, 0.0]
nodo_actual.outputs[3].default_value = 0.0
nodo_actual.outputs[4].default_value = 0.0

# Mix_S_ML es el nodo encargado de mezclar el nodo colorRampS
con el nodo Mix_M_L con el tipo "DARKEN"
nodo_actual = nodos.new(type='ShaderNodeMixRGB')
```

```
nodo_actual.blend_type = 'DARKEN'
nodo_actual.color = (0.608, 0.608, 0.608)
nodo_actual.location = (-381.149, 1011.118)
nodo_actual.name = 'Mix_S_ML'
nodo_actual.select = False
nodo_actual.use_alpha = False
nodo_actual.use_clamp = False
nodo_actual.width = 140.0
nodo_actual.inputs[0].default_value = 0.300

# ColorRamp_Cornea es el nodo encargado de proporcionar
el color blanco de la esclerótica
y determina donde empieza el iris para que sea visible
nodo_actual = nodos.new(type='ShaderNodeValToRGB')
nodo_actual.color = (0.608, 0.608, 0.608)
cr = nodo_actual.color_ramp
cr.color_mode = 'RGB'
cr.hue_interpolation = 'NEAR'
cr.interpolation = 'LINEAR'
stop_actual = cr.elements.new(0.579)
# para crear el color negro queremos en RGB los valores 0.0, 0.0, 0.0
y el 1.0 hace alusión a la opacidad siendo 1.0 completamente opaco y 0.0 sería transparente
stop_actual.color = [0.0, 0.0, 0.0, 1.0]
stop_actual = cr.elements.new(0.607)
# para crear el color blanco queremos en RGB los valores 1.0, 1.0, 1.0
y el 1.0 hace alusión a la opacidad siendo 1.0 completamente opaco y 0.0 sería transparente
stop_actual.color = [1.0, 1.0, 1.0, 1.0]
removed_black = removed_white = False
# Esta parte de código es necesaria para que no se creen los puntos de parada de los extremos
(color negro puro y blanco puro)
for i in range(len(cr.elements) - 1, -1, -1):
    stop = cr.elements[i]
    if not removed_black and stop.position == 0 and
all([stop.color[i] == (0, 0, 0, 1)[i] for i in range(4)]):
        cr.elements.remove(stop)
        removed_black = True
    if not removed_white and stop.position == 1 and
all([stop.color[i] == (1, 1, 1, 1)[i] for i in range(4)]):
        cr.elements.remove(stop)
        removed_white = True
nodo_actual.location = (-116.419, 161.154)
nodo_actual.name = 'ColorRamp_Cornea'
nodo_actual.select = False
nodo_actual.width = 143.66796875
nodo_actual.inputs[0].default_value = 0.5
```

```
nodo_actual.outputs[0].default_value = [0.0, 0.0, 0.0, 0.0]
nodo_actual.outputs[1].default_value = 0.0

# Mapping_Cornea sirve para dar esa transparencia necesaria para poder aprecias el iris/pupila
del ojo
nodo_actual = nodos.new(type='ShaderNodeMapping')
nodo_actual.color = (0.608, 0.608, 0.608)
nodo_actual.location = (-610.645, 158.537)
nodo_actual.name = 'Mapping_Cornea'
nodo_actual.select = False
nodo_actual.vector_type = 'POINT'
nodo_actual.width = 240.0
nodo_actual.inputs[0].default_value = [0.0, 0.0, 0.0]
nodo_actual.inputs[1].default_value = [0.0, 0.800, 0.0]
nodo_actual.inputs[2].default_value = [0.0, 0.0, 0.0]
nodo_actual.inputs[3].default_value = [0.5, 0.5, 0.5]
nodo_actual.outputs[0].default_value = [0.0, 0.0, 0.0]

# Gradient_Texture_Cornea sirve para dar esa transparencia
necesaria para poder aprecias el iris/pupila del ojo, pues de
estar en otro modo la transparencia no se situaría con esas dimensiones en la parte frontal
nodo_actual = nodos.new(type='ShaderNodeTexGradient')
nodo_actual.color = (0.608, 0.608, 0.608)
nodo_actual.gradient_type = 'SPHERICAL'
nodo_actual.location = (-320.645, 96.098)
nodo_actual.name = 'Gradient_Texture_Cornea'
nodo_actual.select = False
nodo_actual.width = 140.0

# Gradient_Texture_Cornea sirve para dar esa transparencia necesaria
para poder aprecias el iris/pupila del ojo,
nodo_actual = nodos.new(type='ShaderNodeTexCoord')
nodo_actual.color = (0.608, 0.608, 0.608)
nodo_actual.from_instancer = False
nodo_actual.location = (-820.645, 141.537)
nodo_actual.name = 'Texture_Coordinate_Cornea'
nodo_actual.object = None
nodo_actual.select = False
nodo_actual.width = 140.0

# Mapping_Iris_Superficie reflejará las líneas características del iris del ojo
nodo_actual = nodos.new(type='ShaderNodeMapping')
nodo_actual.color = (0.608, 0.608, 0.608)
nodo_actual.location = (-742.519, 1637.214)
nodo_actual.name = 'Mapping_Iris_Superficie'
```

```
nodo_actual.select = False
nodo_actual.vector_type = 'POINT'
nodo_actual.width = 240.0
nodo_actual.inputs[0].default_value = [0.0, 0.0, 0.0]
nodo_actual.inputs[1].default_value = [0.0, 0.800, 0.0]
nodo_actual.inputs[2].default_value = [0.0, 0.0, 0.0]
nodo_actual.inputs[3].default_value = [0.5, 0.5, 0.5]
nodo_actual.outputs[0].default_value = [0.0, 0.0, 0.0]

# Gradient_Texture_Iris_Superficie es el nodo que muestrea la textura,
 al estar en modo esférico crea un degradado inverso utilizando
el vector de Mapping_Iris_Superficie
nodo_actual = nodos.new(type='ShaderNodeTexGradient')
nodo_actual.color = (0.608, 0.608, 0.608)
nodo_actual.gradient_type = 'SPHERICAL'
nodo_actual.location = (-452.5185546875, 1574.775634765625)
nodo_actual.name = 'Gradient_Texture_Iris_Superficie'
nodo_actual.select = False
nodo_actual.width = 140.0

# Texture_Coordinate_Iris_Superficie  utilizando las coordenadas del espacio objeto
nodo_actual = nodos.new(type='ShaderNodeTexCoord')
nodo_actual.color = (0.608, 0.608, 0.608)
nodo_actual.from_instancer = False
nodo_actual.location = (-952.5184326171875, 1620.214111328125)
nodo_actual.name = 'Texture_Coordinate_Iris_Superficie'
nodo_actual.object = None
nodo_actual.select = False
nodo_actual.width = 140.0

# Nodo para ver los vasos sanguíneos de la superfice del ojo
ya que se mezcla con el nodo Mix_S_ML
nodo_actual = nodos.new(type='ShaderNodeValToRGB')
nodo_actual.color = (0.608, 0.608, 0.608)
cr = nodo_actual.color_ramp
cr.color_mode = 'RGB'
cr.hue_interpolation = 'NEAR'
cr.interpolation = 'EASE'
stop_actual = cr.elements.new(0.455)
# para crear el color negro queremos en RGB los valores 0.0, 0.0, 0.0
y el 1.0 hace alusión a la opacidad siendo 1.0 completamente
opaco y 0.0 sería transparente
stop_actual.color = [0.0, 0.0, 0.0, 1.0]
stop_actual = cr.elements.new(0.577)
# para crear el color blanco queremos en RGB los valores 1.0, 1.0, 1.0
```

```
y el 1.0 hace alusión a la opacidad siendo 1.0 completamente
 opaco y 0.0 sería transparente
stop_actual.color = [1.0, 1.0, 1.0, 1.0]
removed_black = removed_white = False
# Esta parte de código es necesaria para que no se creen los puntos de parada de los extremos
(color negro puro y blanco puro)
for i in range(len(cr.elements) - 1, -1, -1):
    stop = cr.elements[i]
    if not removed_black and stop.position == 0 and
all([stop.color[i] == (0, 0, 0, 1)[i] for i in range(4)]):
        cr.elements.remove(stop)
        removed_black = True
    if not removed_white and stop.position == 1 and
all([stop.color[i] == (1, 1, 1, 1)[i] for i in range(4)]):
        cr.elements.remove(stop)
        removed_white = True
nodo_actual.location = (-248.293, 1639.831)
nodo_actual.name = 'ColorRamp_Vasos_Sanguíneos'
nodo_actual.select = False
nodo_actual.width = 349.480
nodo_actual.inputs[0].default_value = 0.5
nodo_actual.outputs[0].default_value = [0.0, 0.0, 0.0, 0.0]
nodo_actual.outputs[1].default_value = 0.0

# Texture_Coordinate_S_M_L utilizando las coordenadas
del espacio objeto el cual irá al nodo Mapping_S, Mapping_M y
Mix_L el cual posteriormente la salida de color irá al nodo Mapping_L
nodo_actual = nodos.new(type='ShaderNodeTexCoord')
nodo_actual.color = (0.608, 0.608, 0.608)
nodo_actual.from_instancer = False
nodo_actual.location = (-2357.006, 985.429)
nodo_actual.name = 'Texture_Coordinate_S_M_L'
nodo_actual.object = None
nodo_actual.select = False
nodo_actual.width = 140.0

# Noise_Texture_L es un ruido 3D el cual le da ese efecto a los vasos sanguíneos
nodo_actual = nodos.new(type='ShaderNodeTexNoise')
nodo_actual.color = (0.608, 0.608, 0.608)
nodo_actual.location = (-2281.211, 671.236)
nodo_actual.name = 'Noise_Texture_L'
nodo_actual.noise_dimensions = '3D'
nodo_actual.select = False
nodo_actual.width = 140.0
nodo_actual.inputs[0].default_value = [0.0, 0.0, 0.0]
```

```
nodo_actual.inputs[1].default_value = 0.0
nodo_actual.inputs[2].default_value = 5.0
nodo_actual.inputs[3].default_value = 2.0
nodo_actual.inputs[4].default_value = 0.5
nodo_actual.inputs[5].default_value = 0.0
nodo_actual.outputs[0].default_value = 0.0
nodo_actual.outputs[1].default_value = [0.0, 0.0, 0.0, 0.0]

# Mix_L fusiona con el modo Luz Lineal entre Texture_Coordinate_S_M_L
y el ruido 3D del Noise_Texture_L
nodo_actual = nodos.new(type='ShaderNodeMixRGB')
nodo_actual.blend_type = 'LINEAR_LIGHT'
nodo_actual.color = (0.608, 0.608, 0.608)
nodo_actual.location = (-2060.082, 661.099)
nodo_actual.name = 'Mix_L'
nodo_actual.select = False
nodo_actual.use_alpha = False
nodo_actual.use_clamp = False
nodo_actual.width = 140.0
nodo_actual.inputs[0].default_value = 0.30000001192092896
nodo_actual.inputs[1].default_value = [0.5, 0.5, 0.5, 1.0]
nodo_actual.inputs[2].default_value = [0.5, 0.5, 0.5, 1.0]
nodo_actual.outputs[0].default_value = [0.0, 0.0, 0.0, 0.0]

# ColorRamp_L es el nodo encargado de proporcionar
el color rojo a esos vasos sanguíneos de longitud media
nodo_actual = nodos.new(type='ShaderNodeValToRGB')
nodo_actual.color = (0.608, 0.608, 0.608)
cr = nodo_actual.color_ramp
cr.color_mode = 'RGB'
cr.hue_interpolation = 'NEAR'
cr.interpolation = 'LINEAR'
stop_actual = cr.elements.new(0.543)
# para crear el color BLANCO queremos en RGB los valores 1.0, 1.0, 1.0
y el 1.0 hace alusión a la opacidad siendo 1.0
completamente opaco y 0.0 sería transparente
stop_actual.color = [1.0, 1.0, 1.0, 1.0]
stop_actual = cr.elements.new(0.848)
# para crear el color ROJO queremos en RGB los valores 0.314, 0.005 ,0.0
y el 1.0 hace alusión a la opacidad siendo 1.0
completamente opaco y 0.0 sería transparente
stop_actual.color = [0.314, 0.005, 0.0, 1.0]
removed_black = removed_white = False
# Esta parte de código es necesaria para que no se creen los puntos
de parada de los extremos (color negro puro y blanco puro)
```

```
for i in range(len(cr.elements) - 1, -1, -1):
    stop = cr.elements[i]
    if not removed_black and stop.position == 0 and
all([stop.color[i] == (0, 0, 0, 1)[i] for i in range(4)]):
        cr.elements.remove(stop)
        removed_black = True
    if not removed_white and stop.position == 1 and
all([stop.color[i] == (1, 1, 1, 1)[i] for i in range(4)]):
        cr.elements.remove(stop)
        removed_white = True
nodo_actual.location = (475.047, -50.812)
nodo_actual.name = 'ColorRamp_L'
# Asignamos el nodo padre, ya que Mapping_L es un
nodo que se encuentra en el interior del nodo de Frame_L
parent = nodos.get('Frame_L')
if parent:
    nodo_actual.parent = parent
    while True:
        nodo_actual.location += parent.location
        if parent.parent:
            parent = parent.parent
        else:
            break
nodo_actual.select = False
nodo_actual.width = 240.0
nodo_actual.inputs[0].default_value = 0.5
nodo_actual.outputs[0].default_value = [0.0, 0.0, 0.0, 0.0]
nodo_actual.outputs[1].default_value = 0.0


# ColorRamp_Rojo_Superficie es el nodo encargado de poner un color rojo
emulando a la sangre en la parte trasera del ojo aclarándose según avanza
 a la parte frontal del ojo además de delimitar donde terminan los vasos sanguíneos
nodo_actual = nodos.new(type='ShaderNodeValToRGB')
nodo_actual.color = (0.608, 0.608, 0.608)
cr = nodo_actual.color_ramp
cr.color_mode = 'RGB'
cr.hue_interpolation = 'NEAR'
cr.interpolation = 'EASE'
stop_actual = cr.elements.new(0.091)
# para crear el color ROJO queremos en RGB los valores 0.314, 0.005 ,0.0
y el 1.0 hace alusión a la opacidad siendo 1.0
completamente opaco y 0.0 sería transparente
stop_actual.color = [0.314, 0.005, 0.0, 1.0]
stop_actual = cr.elements.new(0.347)
```

```
# para crear el color ROJO SUAVE queremos en RGB los valores 0.666, 0.264 ,0.174
y el 1.0 hace alusión a la opacidad siendo 1.0
completamente opaco y 0.0 sería transparente
stop_actual.color = [0.666, 0.264, 0.174, 1.0]
stop_actual = cr.elements.new(0.611)
# para crear el color BLANCO queremos en RGB los valores 1.0, 1.0, 1.0
y el 1.0 hace alusión a la opacidad siendo 1.0
completamente opaco y 0.0 sería transparente
stop_actual.color = [1.0, 1.0, 1.0, 1.0]
removed_black = removed_white = False
# Esta parte de código es necesaria para que no se creen los puntos
de parada de los extremos (color negro puro y blanco puro)
for i in range(len(cr.elements) - 1, -1, -1):
    stop = cr.elements[i]
    if not removed_black and stop.position == 0 and
all([stop.color[i] == (0, 0, 0, 1)[i] for i in range(4)]):
        cr.elements.remove(stop)
        removed_black = True
    if not removed_white and stop.position == 1 and
all([stop.color[i] == (1, 1, 1, 1)[i] for i in range(4)]):
        cr.elements.remove(stop)
        removed_white = True
nodo_actual.location = (-113.402, 384.835)
nodo_actual.name = 'ColorRamp_Rojo_Superficie'
nodo_actual.select = False
nodo_actual.width = 194.450
nodo_actual.inputs[0].default_value = 0.5
nodo_actual.outputs[0].default_value = [0.0, 0.0, 0.0, 0.0]
nodo_actual.outputs[1].default_value = 0.0

# Mix_Total es el nodo que mezcla de modo superpuesto todos los ColorRamp
relacionados con el rojo sanguíneo para mandarlo
como color base al nodo principal BSDF
nodo_actual = nodos.new(type='ShaderNodeMixRGB')
nodo_actual.blend_type = 'OVERLAY'
nodo_actual.color = (0.608, 0.608, 0.608)
nodo_actual.location = (629.845, 1025.724)
nodo_actual.name = 'Mix_Total'
nodo_actual.select = False
nodo_actual.use_alpha = False
nodo_actual.use_clamp = False
nodo_actual.width = 140.0
nodo_actual.inputs[0].default_value = 1.0
nodo_actual.inputs[1].default_value = [0.5, 0.5, 0.5, 1.0]
nodo_actual.inputs[2].default_value = [0.5, 0.5, 0.5, 1.0]
```

```
nodo_actual.outputs[0].default_value = [0.0, 0.0, 0.0, 0.0]

# Mix_Vasos_Sanguineos es el nodo que mezcla de modo superpuesto todos los ColorRamp
 para dar una especie de altura a los vasos sanguíneos a través del nodo bump
nodo_actual = nodos.new(type='ShaderNodeMixRGB')
nodo_actual.blend_type = 'MIX'
nodo_actual.color = (0.608, 0.608, 0.608)
nodo_actual.location = (54.530, 1012.599)
nodo_actual.name = 'Mix_Vasos_Sanguineos'
nodo_actual.select = False
nodo_actual.use_alpha = False
nodo_actual.use_clamp = False
nodo_actual.width = 140.0
nodo_actual.inputs[0].default_value = 0.5
nodo_actual.inputs[1].default_value = [0.5, 0.5, 0.5, 1.0]
nodo_actual.inputs[2].default_value = [1.0, 1.0, 1.0, 1.0]
nodo_actual.outputs[0].default_value = [0.0, 0.0, 0.0, 0.0]

# Bump es el nodo qotorga una especie de altura a los vasos
sanguíneos a través del nodo bump
nodo_actual = nodos.new(type='ShaderNodeBump')
nodo_actual.color = (0.608, 0.608, 0.608)
nodo_actual.invert = True
nodo_actual.location = (-65.068, -147.066)
nodo_actual.name = 'Bump'
nodo_actual.select = False
nodo_actual.width = 140.0
nodo_actual.inputs[0].default_value = 0.100
nodo_actual.inputs[1].default_value = 0.100
nodo_actual.inputs[2].default_value = 1.0
nodo_actual.inputs[3].default_value = [0.0, 0.0, 0.0]
nodo_actual.outputs[0].default_value = [0.0, 0.0, 0.0]

# ColorRamp_Bump es el nodo qotorga color para esa sensación
de altura que ofrece el nodo Bump
nodo_actual = nodos.new(type='ShaderNodeValToRGB')
nodo_actual.color = (0.608, 0.608, 0.608)
cr = nodo_actual.color_ramp
cr.color_mode = 'RGB'
cr.hue_interpolation = 'NEAR'
cr.interpolation = 'B_SPLINE'
stop_actual = cr.elements.new(0.0)
# para crear el color NEGRO queremos en RGB los valores 0.0, 0.0, 0.0
y el 1.0 hace alusión a la opacidad siendo 1.0 completamente
opaco y 0.0 sería transparente
```

```
stop_actual.color = [0.0, 0.0, 0.0, 1.0]
stop_actual = cr.elements.new(1.0)
# para crear el color BLANCO queremos en RGB los valores 1.0, 1.0, 1.0
y el 1.0 hace alusión a la opacidad siendo 1.0 completamente
opaco y 0.0 sería transparente
stop_actual.color = [1.0, 1.0, 1.0, 1.0]
removed_black = removed_white = False
# Esta parte de código es necesaria para que no se creen los puntos
de parada de los extremos (color negro puro y blanco puro)
for i in range(len(cr.elements) - 1, -1, -1):
    stop = cr.elements[i]
    if not removed_black and stop.position == 0 and
all([stop.color[i] == (0, 0, 0, 1)[i] for i in range(4)]):
        cr.elements.remove(stop)
        removed_black = True
    if not removed_white and stop.position == 1 and
all([stop.color[i] == (1, 1, 1, 1)[i] for i in range(4)]):
        cr.elements.remove(stop)
        removed_white = True
nodo_actual.location = (-687.004, 647.723)
nodo_actual.name = 'ColorRamp_Bump'
nodo_actual.select = False
nodo_actual.width = 240.0
nodo_actual.inputs[0].default_value = 0.5
nodo_actual.outputs[0].default_value = [0.0, 0.0, 0.0, 0.0]
nodo_actual.outputs[1].default_value = 0.0


# Voronoi_Texture_S es el encargado de como vemos los
 vasos sanguíneos de menor longitud
nodo_actual = nodos.new(type='ShaderNodeTexVoronoi')
nodo_actual.color = (0.608, 0.608, 0.608)
nodo_actual.distance = 'EUCLIDEAN'
nodo_actual.feature = 'F1'
nodo_actual.location = (277.432, -33.755)
nodo_actual.name = 'Voronoi_Texture_S'
# Asignamos el nodo padre, ya que Mapping_L es un nodo que
se encuentra en el interior del nodo de Frame_S
parent = nodos.get('Frame_S')
if parent:
    nodo_actual.parent = parent
    while True:
        nodo_actual.location += parent.location
        if parent.parent:
            parent = parent.parent
```

```
        else:
            break
nodo_actual.select = False
nodo_actual.voronoi_dimensions = '3D'
nodo_actual.width = 140.0
nodo_actual.inputs[0].default_value = [0.0, 0.0, 0.0]
nodo_actual.inputs[1].default_value = 0.0
nodo_actual.inputs[2].default_value = 20.0
nodo_actual.inputs[3].default_value = 1.0
nodo_actual.inputs[4].default_value = 0.5
nodo_actual.inputs[5].default_value = 1.0
nodo_actual.outputs[0].default_value = 0.0
nodo_actual.outputs[1].default_value = [0.0, 0.0, 0.0, 0.0]
nodo_actual.outputs[2].default_value = [0.0, 0.0, 0.0]
nodo_actual.outputs[3].default_value = 0.0
nodo_actual.outputs[4].default_value = 0.0

# Material Output es el nodo encargado de que todos los
nodos hagan efecto en la superficie, sin él no se vería reflejado en la consola blender
nodo_actual = nodos.new(type='ShaderNodeOutputMaterial')
nodo_actual.color = (0.608, 0.608, 0.608)
nodo_actual.is_active_output = True
nodo_actual.location = (576.799, 302.339)
nodo_actual.name = 'Material Output'
nodo_actual.select = False
nodo_actual.target = 'ALL'
nodo_actual.width = 140.0
nodo_actual.inputs[2].default_value = [0.0, 0.0, 0.0]
nodo_actual.inputs[3].default_value = 0.0

# Principled BSDF es el nodo que se crea por defecto en el
cual se aplicará el nodo Mix_Total, Bump, y el ColorRamp_Vasos_Sanguíneos
nodo_actual = nodos.new(type='ShaderNodeBsdfPrincipled')
nodo_actual.color = (0.608, 0.608, 0.608)
nodo_actual.distribution = 'GGX'
nodo_actual.location = (324.994, 311.986)
nodo_actual.name = 'Principled BSDF'
nodo_actual.subsurface_method = 'BURLEY'
nodo_actual.width = 240.0
nodo_actual.inputs[0].default_value = [0.800, 0.800, 0.800, 1.0]
nodo_actual.inputs[1].default_value = 0.0
nodo_actual.inputs[2].default_value = [1.0, 0.200, 0.100]
nodo_actual.inputs[3].default_value = [0.800, 0.800, 0.800, 1.0]
nodo_actual.inputs[4].default_value = 1.399
nodo_actual.inputs[5].default_value = 0.0
```

```
nodo_actual.inputs[6].default_value = 0.0
nodo_actual.inputs[7].default_value = 0.5
nodo_actual.inputs[8].default_value = 0.0
nodo_actual.inputs[9].default_value = 0.0500
nodo_actual.inputs[10].default_value = 0.0
nodo_actual.inputs[11].default_value = 0.0
nodo_actual.inputs[12].default_value = 0.0
nodo_actual.inputs[13].default_value = 0.5
nodo_actual.inputs[14].default_value = 0.0
nodo_actual.inputs[15].default_value = 0.5
nodo_actual.inputs[16].default_value = 1.450
nodo_actual.inputs[17].default_value = 1.0
nodo_actual.inputs[18].default_value = 0.0
nodo_actual.inputs[19].default_value = [0.0, 0.0, 0.0, 1.0]
nodo_actual.inputs[20].default_value = 1.0
nodo_actual.inputs[21].default_value = 1.0
nodo_actual.inputs[22].default_value = [0.0, 0.0, 0.0]
nodo_actual.inputs[23].default_value = [0.0, 0.0, 0.0]
nodo_actual.inputs[24].default_value = [0.0, 0.0, 0.0]
nodo_actual.inputs[25].default_value = 0.0

######### Enlaces entre los nodos #########

enlaces.new(nodos["Mapping_Cornea"].outputs[0], nodos["Gradient_Texture_Cornea"].inputs[0])
enlaces.new(nodos["Texture_Coordinate_Cornea"].outputs[3], nodos["Mapping_Cornea"].inputs[0])

enlaces.new(nodos["Gradient_Texture_Cornea"].outputs[0], nodos["ColorRamp_Cornea"].inputs[0])

enlaces.new(nodos["ColorRamp_Cornea"].outputs[0], nodos["Principled BSDF"].inputs[17])
enlaces.new(nodos["Gradient_Texture_Cornea"].outputs[0],
nodos["ColorRamp_Rojo_Superficie"].inputs[0])
enlaces.new(nodos["Mapping_L"].outputs[0], nodos["Voronoi_Texture_L"].inputs[0])
enlaces.new(nodos["Voronoi_Texture_L"].outputs[0], nodos["ColorRamp_L"].inputs[0])
enlaces.new(nodos["Mapping_M"].outputs[0], nodos["Voronoi_Texture_M"].inputs[0])
enlaces.new(nodos["Voronoi_Texture_M"].outputs[0], nodos["ColorRamp_M"].inputs[0])
enlaces.new(nodos["Mapping_S"].outputs[0], nodos["Voronoi_Texture_S"].inputs[0])
enlaces.new(nodos["Voronoi_Texture_S"].outputs[0], nodos["ColorRamp_S"].inputs[0])
enlaces.new(nodos["Texture_Coordinate_S_M_L"].outputs[3], nodos["Mapping_M"].inputs[0])
enlaces.new(nodos["Texture_Coordinate_S_M_L"].outputs[3], nodos["Mapping_S"].inputs[0])
enlaces.new(nodos["ColorRamp_M"].outputs[0], nodos["Mix_M_L"].inputs[1])
enlaces.new(nodos["ColorRamp_L"].outputs[0], nodos["Mix_M_L"].inputs[2])
enlaces.new(nodos["Mix_M_L"].outputs[0], nodos["Mix_S_ML"].inputs[1])
enlaces.new(nodos["ColorRamp_S"].outputs[0], nodos["Mix_S_ML"].inputs[2])
enlaces.new(nodos["Mix_Total"].outputs[0], nodos["Principled BSDF"].inputs[0])
enlaces.new(nodos["Mix_S_ML"].outputs[0], nodos["Mix_Vasos_Sanguineos"].inputs[1])
```

```
enlaces.new(nodos["Mapping_Iris_Superficie"].outputs[0],
nodos["Gradient_Texture_Iris_Superficie"].inputs[0])
enlaces.new(nodos["Texture_Coordinate_Iris_Superficie"].outputs[3],
nodos["Mapping_Iris_Superficie"].inputs[0])
enlaces.new(nodos["Gradient_Texture_Iris_Superficie"].outputs[0],
 nodos["ColorRamp_Vasos_Sanguíneos"].inputs[0])
enlaces.new(nodos["ColorRamp_Vasos_Sanguíneos"].outputs[0],
nodos["Mix_Vasos_Sanguineos"].inputs[0])
enlaces.new(nodos["Mix_Vasos_Sanguineos"].outputs[0], nodos["Mix_Total"].inputs[2])
enlaces.new(nodos["ColorRamp_Rojo_Superficie"].outputs[0], nodos["Mix_Total"].inputs[1])
enlaces.new(nodos["Noise_Texture_L"].outputs[1], nodos["Mix_L"].inputs[2])
enlaces.new(nodos["Texture_Coordinate_S_M_L"].outputs[3], nodos["Mix_L"].inputs[1])
enlaces.new(nodos["Mix_L"].outputs[0], nodos["Mapping_L"].inputs[0])
enlaces.new(nodos["Mix_Vasos_Sanguineos"].outputs[0], nodos["ColorRamp_Bump"].inputs[0])
enlaces.new(nodos["ColorRamp_Bump"].outputs[0], nodos["Bump"].inputs[2])
enlaces.new(nodos["Bump"].outputs[0], nodos["Principled BSDF"].inputs[22])
enlaces.new(nodos["Principled BSDF"].outputs[0], nodos["Material Output"].inputs[0])

############################## IRIS ###################################

mat = bpy.data.materials.new(name='Iris')  # Creación del material
# Configuración del material
obj.data.materials.append(mat)  # Asignación del material

new_mat = bpy.data.materials.get('Iris')
if not new_mat:
    new_mat = bpy.data.materials.new('Iris')

# habilitaremos el uso de nodos, ya que sin ello no podríamos crear ninguno
new_mat.use_nodes = True

# Definimos la estructura que tendran los nodos y los enlaces para su creación
node_tree = new_mat.node_tree
nodos = node_tree.nodes
nodos.clear()

enlaces = node_tree.links
enlaces.clear()

######### Nodos #########

# Mapping_Pupila determina el radio de la pupila
nodo_actual = nodos.new(type='ShaderNodeMapping')
nodo_actual.color = (0.608, 0.608, 0.608)
nodo_actual.location = (-760.619, 177.171)
```

```
nodo_actual.name = 'Mapping_Pupila'
nodo_actual.vector_type = 'POINT'
nodo_actual.width = 240.0
nodo_actual.inputs[0].default_value = [0.0, 0.0, 0.0]
nodo_actual.inputs[1].default_value = [0.0, 0.649, 0.0]
nodo_actual.inputs[2].default_value = [0.0, 0.0, 0.0]
nodo_actual.inputs[3].default_value = [1.0, 1.0, 1.0]
nodo_actual.outputs[0].default_value = [0.0, 0.0, 0.0]

# Gradient_Texture_Iris_Medida es el nodo que muestrea la textura,
al estar en modo esférico crea un degradado inverso utilizando el vector de Mapping_Pupila
nodo_actual = nodos.new(type='ShaderNodeTexGradient')
nodo_actual.color = (0.608, 0.608, 0.608)
nodo_actual.gradient_type = 'SPHERICAL'
nodo_actual.location = (-470.620, 115.160)
nodo_actual.name = 'Gradient_Texture_Iris_Medida'
nodo_actual.width = 140.0

# Texture_Coordinate_Iris_Base utilizando las coordenadas del
espacio objeto el cual irá al nodo Mapping_Pupila
nodo_actual = nodos.new(type='ShaderNodeTexCoord')
nodo_actual.color = (0.608, 0.608, 0.608)
nodo_actual.from_instancer = False
nodo_actual.location = (-970.619, 160.171)
nodo_actual.name = 'Texture_Coordinate_Iris_Base'
nodo_actual.object = None
nodo_actual.width = 140.0

# Texture_Coordinate_Iris_Interior utilizando las coordenadas
del espacio objeto el cual irá al nodo Mapping_Pupila
nodo_actual = nodos.new(type='ShaderNodeTexCoord')
nodo_actual.color = (0.608, 0.608, 0.608)
nodo_actual.from_instancer = False
nodo_actual.location = (-987.080, 524.705)
nodo_actual.name = 'Texture_Coordinate_Iris_Interior'
nodo_actual.object = None
nodo_actual.width = 140.0

# Mapping_Iris es el nodo utilizado para las "líneas" del Iris
nodo_actual = nodos.new(type='ShaderNodeMapping')
nodo_actual.color = (0.608, 0.608, 0.608)
nodo_actual.location = (-777.080, 541.705)
nodo_actual.name = 'Mapping_Iris'
nodo_actual.vector_type = 'NORMAL'
nodo_actual.width = 240.0
```

```
nodo_actual.inputs[0].default_value = [0.0, 0.0, 0.0]
nodo_actual.inputs[1].default_value = [0.0, 0.0, 0.0]
nodo_actual.inputs[2].default_value = [0.0, 0.0, 0.0]
nodo_actual.inputs[3].default_value = [1.0, 15.600, 1.0]
nodo_actual.outputs[0].default_value = [0.0, 0.0, 0.0]

# Noise_Texture_Iris es el nodo que trata la distorsión y lo nítido que es el Iris
nodo_actual = nodos.new(type='ShaderNodeTexNoise')
nodo_actual.color = (0.608, 0.608, 0.608)
nodo_actual.location = (-480.180, 462.844)
nodo_actual.name = 'Noise_Texture_Iris'
nodo_actual.noise_dimensions = '3D'
nodo_actual.width = 140.0
nodo_actual.inputs[0].default_value = [0.0, 0.0, 0.0]
nodo_actual.inputs[1].default_value = 0.0
nodo_actual.inputs[2].default_value = 5.0
nodo_actual.inputs[3].default_value = 10.0
nodo_actual.inputs[4].default_value = 0.5
nodo_actual.inputs[5].default_value = 2.5
nodo_actual.outputs[0].default_value = 0.0
nodo_actual.outputs[1].default_value = [0.0, 0.0, 0.0, 0.0]

# ColorRamp_Iris_Interior es el nodo encargado de las tonalidades de azul en el interior del
iris
nodo_actual = nodos.new(type='ShaderNodeValToRGB')
nodo_actual.color = (0.608, 0.608, 0.608)
cr = nodo_actual.color_ramp
cr.color_mode = 'RGB'
cr.hue_interpolation = 'NEAR'
cr.interpolation = 'LINEAR'
stop_actual = cr.elements.new(0.350)
# para crear el color azul claro queremos en RGB los valores
0.420, 0.700, 0.700 y el 1.0 hace alusión a la opacidad siendo 1.0
completamente opaco y 0.0 sería transparente
stop_actual.color = [0.420, 0.700, 0.700, 1.0]
stop_actual = cr.elements.new(0.600)
# para crear el color azul oscuro queremos en RGB los valores
0.120, 0.232, 0.400 y el 1.0 hace alusión a la opacidad siendo 1.0
completamente opaco y 0.0 sería transparente
stop_actual.color = [0.120, 0.232, 0.400, 1.0]
removed_black = removed_white = False
# Esta parte de código es necesaria para que no se creen
los puntos de parada de los extremos (color negro puro y blanco puro)
for i in range(len(cr.elements) - 1, -1, -1):
    stop = cr.elements[i]
```

```
    if not removed_black and stop.position == 0 and
all([stop.color[i] == (0, 0, 0, 1)[i] for i in range(4)]):
        cr.elements.remove(stop)
        removed_black = True
    if not removed_white and stop.position == 1 and
all([stop.color[i] == (1, 1, 1, 1)[i] for i in range(4)]):
        cr.elements.remove(stop)
        removed_white = True
nodo_actual.location = (-235.032, 521.752)
nodo_actual.name = 'ColorRamp_Iris_Interior'
nodo_actual.width = 240.0
nodo_actual.inputs[0].default_value = 0.5
nodo_actual.outputs[0].default_value = [0.0, 0.0, 0.0, 0.0]
nodo_actual.outputs[1].default_value = 0.0

# ColorRamp_Iris_Circunferencia es el nodo encargado
de las tonalidades de azul en el interior del iris
nodo_actual = nodos.new(type='ShaderNodeValToRGB')
nodo_actual.color = (0.608, 0.608, 0.608)
cr = nodo_actual.color_ramp
cr.color_mode = 'RGB'
cr.hue_interpolation = 'NEAR'
cr.interpolation = 'EASE'
stop_actual = cr.elements.new(0.5)
# para crear el color negro queremos en RGB los valores
0.0, 0.0, 0.0 y el 1.0 hace alusión a la opacidad siendo 1.0
completamente opaco y 0.0 sería transparente
stop_actual.color = [0.0, 0.0, 0.0, 1.0]
stop_actual = cr.elements.new(0.550)
# para crear el color azul oscuro queremos en RGB los valores
 0.020, 0.028, 0.100 y el 1.0 hace alusión a la
opacidad siendo 1.0 completamente opaco y 0.0 sería transparente
stop_actual.color = [0.020, 0.028, 0.100, 1.0]
stop_actual = cr.elements.new(0.620)
# para crear el color azul oscuro pálido queremos en RGB los valores
 0.120, 0.152, 0.200 y el 1.0 hace alusión a la opacidad siendo 1.0
completamente opaco y 0.0 sería transparente
stop_actual.color = [0.120, 0.152, 0.200, 1.0]
stop_actual = cr.elements.new(0.700)
# para crear el color azul claro queremos en RGB los valores
0.020, 0.028, 0.100 y el 1.0 hace alusión a la opacidad
siendo 1.0 completamente opaco y 0.0 sería transparente
stop_actual.color = [0.302, 0.476, 0.550, 1.0]
stop_actual = cr.elements.new(0.760)
# para crear el color azul oscuro pálido queremos en RGB
```

```
los valores 0.120, 0.152, 0.200 y el 1.0 hace alusión
a la opacidad siendo 1.0 completamente opaco y 0.0 sería transparente
stop_actual.color = [0.087, 0.133, 0.25, 1.0]
stop_actual = cr.elements.new(0.764)
# para crear el color negro queremos en RGB los valores
 0.0, 0.0, 0.0 y el 1.0 hace alusión a la opacidad siendo
1.0 completamente opaco y 0.0 sería transparente
stop_actual.color = [0.0, 0.0, 0.0, 1.0]
removed_black = removed_white = False
# Esta parte de código es necesaria para que no se
creen los puntos de parada de los extremos (color negro puro y blanco puro)
for i in range(len(cr.elements) - 1, -1, -1):
    stop = cr.elements[i]
    if not removed_black and stop.position == 0 and
all([stop.color[i] == (0, 0, 0, 1)[i] for i in range(4)]):
        cr.elements.remove(stop)
        removed_black = True
    if not removed_white and stop.position == 1 and
all([stop.color[i] == (1, 1, 1, 1)[i] for i in range(4)]):
        cr.elements.remove(stop)
        removed_white = True
nodo_actual.location = (-237.990, 246.606)
nodo_actual.name = 'ColorRamp_Iris_Circunferencia'
nodo_actual.width = 244.526
nodo_actual.inputs[0].default_value = 0.5
nodo_actual.outputs[0].default_value = [0.0, 0.0, 0.0, 0.0]
nodo_actual.outputs[1].default_value = 0.0

# Mix es el nodo que mezcla de modo superpuesto ColorRamp_Iris_Interior
con ColorRamp_Iris_Circunferencia para dar ese patrón al iris
nodo_actual = nodos.new(type='ShaderNodeMixRGB')
nodo_actual.blend_type = 'OVERLAY'
nodo_actual.color = (0.608, 0.608, 0.608)
nodo_actual.location = (281.812, 481.425)
nodo_actual.name = 'Mix'
nodo_actual.use_alpha = False
nodo_actual.use_clamp = False
nodo_actual.width = 140.0
nodo_actual.inputs[0].default_value = 1.0

# Material Output es el nodo encargado de que todos los nodos
hagan efecto en la mesh, sin él no se vería reflejado en la aplicación de Blender
nodo_actual = nodos.new(type='ShaderNodeOutputMaterial')
nodo_actual.color = (0.608, 0.608, 0.608)
nodo_actual.is_active_output = True
```

```
nodo_actual.location = (1551.203, 451.363)
nodo_actual.name = 'Material Output'
nodo_actual.target = 'ALL'
nodo_actual.width = 140.0
nodo_actual.inputs[2].default_value = [0.0, 0.0, 0.0]
nodo_actual.inputs[3].default_value = 0.0

# Principled BSDF es el nodo que se crea por defecto en el cual
 se aplicará el nodo Mix en el color Base y el Bump para la nitidez del iris
nodo_actual = nodos.new(type='ShaderNodeBsdfPrincipled')
nodo_actual.color = (0.608, 0.608, 0.608)
nodo_actual.distribution = 'GGX'
nodo_actual.location = (1213.195, 444.630)
nodo_actual.name = 'Principled BSDF'
nodo_actual.subsurface_method = 'BURLEY'
nodo_actual.width = 240.0
nodo_actual.inputs[0].default_value = [0.008, 0.0, 0.800, 1.0]
nodo_actual.inputs[1].default_value = 0.0
nodo_actual.inputs[2].default_value = [1.0, 0.200, 0.100]
nodo_actual.inputs[3].default_value = [0.800, 0.800, 0.800, 1.0]
nodo_actual.inputs[4].default_value = 1.399
nodo_actual.inputs[5].default_value = 0.0
nodo_actual.inputs[6].default_value = 0.699
nodo_actual.inputs[7].default_value = 0.0
nodo_actual.inputs[8].default_value = 0.0
nodo_actual.inputs[9].default_value = 0.5
nodo_actual.inputs[10].default_value = 0.0
nodo_actual.inputs[11].default_value = 0.0
nodo_actual.inputs[12].default_value = 0.0
nodo_actual.inputs[13].default_value = 0.5
nodo_actual.inputs[14].default_value = 0.0
nodo_actual.inputs[15].default_value = 0.029
nodo_actual.inputs[16].default_value = 1.450
nodo_actual.inputs[17].default_value = 0.0
nodo_actual.inputs[18].default_value = 0.0
nodo_actual.inputs[19].default_value = [0.0, 0.0, 0.0, 1.0]
nodo_actual.inputs[20].default_value = 1.0
nodo_actual.inputs[21].default_value = 1.0
nodo_actual.inputs[22].default_value = [0.0, 0.0, 0.0]
nodo_actual.inputs[23].default_value = [0.0, 0.0, 0.0]
nodo_actual.inputs[24].default_value = [0.0, 0.0, 0.0]
nodo_actual.inputs[25].default_value = 0.0

# ColorRamp_Iris_Textura es el nodo encargado de las tonalidades del iris internas
nodo_actual = nodos.new(type='ShaderNodeValToRGB')
```

```
nodo_actual.color = (0.608, 0.608, 0.608)
cr = nodo_actual.color_ramp
cr.color_mode = 'RGB'
cr.hue_interpolation = 'NEAR'
cr.interpolation = 'LINEAR'
stop_actual = cr.elements.new(0.050)
# para crear el color negro queremos en RGB los valores
0.0, 0.0, 0.0 y el 1.0 hace alusión a la opacidad siendo
1.0 completamente opaco y 0.0 sería transparente
stop_actual.color = [0.0, 0.0, 0.0, 1.0]
stop_actual = cr.elements.new(1.0)
# para crear el color blanco queremos en RGB los valores
1.0, 1.0, 1.0 y el 1.0 hace alusión a la opacidad siendo
1.0 completamente opaco y 0.0 sería transparente
stop_actual.color = [1.0, 1.0, 1.0, 1.0]
removed_black = removed_white = False
# Esta parte de código es necesaria para que no se creen
 los puntos de parada de los extremos (color negro puro y blanco puro)
for i in range(len(cr.elements) - 1, -1, -1):
    stop = cr.elements[i]
    if not removed_black and stop.position == 0 and
all([stop.color[i] == (0, 0, 0, 1)[i] for i in range(4)]):
        cr.elements.remove(stop)
        removed_black = True
    if not removed_white and stop.position == 1 and
all([stop.color[i] == (1, 1, 1, 1)[i] for i in range(4)]):
        cr.elements.remove(stop)
        removed_white = True
nodo_actual.location = (523.012, 309.724)
nodo_actual.name = 'ColorRamp_Iris_Textura'
nodo_actual.width = 240.0
nodo_actual.inputs[0].default_value = 0.5
nodo_actual.outputs[0].default_value = [0.0, 0.0, 0.0, 0.0]
nodo_actual.outputs[1].default_value = 0.0

# Bump es el nodo qotorga una especie de altura a las
líneas que determinan el patrón del Iris
nodo_actual = nodos.new(type='ShaderNodeBump')
nodo_actual.color = (0.608, 0.608, 0.608)
nodo_actual.invert = False
nodo_actual.location = (931.314, 292.384)
nodo_actual.name = 'Bump'
nodo_actual.width = 140.0
nodo_actual.inputs[0].default_value = 0.100
nodo_actual.inputs[1].default_value = 0.100
```

```
nodo_actual.inputs[2].default_value = 1.0
nodo_actual.inputs[3].default_value = [0.0, 0.0, 0.0]
nodo_actual.outputs[0].default_value = [0.0, 0.0, 0.0]

######### Enlaces entre los nodos #########

enlaces.new(nodos["Mapping_Pupila"].outputs[0],
nodos["Gradient_Texture_Iris_Medida"].inputs[0])
enlaces.new(nodos["Gradient_Texture_Iris_Medida"].outputs[0],
nodos["ColorRamp_Iris_Circunferencia"].inputs[0])
enlaces.new(nodos["Texture_Coordinate_Iris_Base"].outputs[3],
nodos["Mapping_Pupila"].inputs[0])
enlaces.new(nodos["Mapping_Iris"].outputs[0], nodos["Noise_Texture_Iris"].inputs[0])
enlaces.new(nodos["Texture_Coordinate_Iris_Interior"].outputs[3],
nodos["Mapping_Iris"].inputs[0])
enlaces.new(nodos["Noise_Texture_Iris"].outputs[0],
nodos["ColorRamp_Iris_Interior"].inputs[0])
enlaces.new(nodos["ColorRamp_Iris_Circunferencia"].outputs[0], nodos["Mix"].inputs[1])
enlaces.new(nodos["ColorRamp_Iris_Interior"].outputs[0], nodos["Mix"].inputs[2])
enlaces.new(nodos["Mix"].outputs[0], nodos["Principled BSDF"].inputs[0])
enlaces.new(nodos["Mix"].outputs[0], nodos["ColorRamp_Iris_Textura"].inputs[0])
enlaces.new(nodos["Principled BSDF"].outputs[0], nodos["Material Output"].inputs[0])
enlaces.new(nodos["ColorRamp_Iris_Textura"].outputs[0], nodos["Bump"].inputs[2])
enlaces.new(nodos["Bump"].outputs[0], nodos["Principled BSDF"].inputs[22])

# Definimos los vértices que comprenden el iris en vertices_Iris
bpy.context.object.active_material_index = 1
vertices_Iris =
[15,30,45,60,75,90,105,120,136,151,167,182,197,212,227,241,242,243,244,245,246,247
,248,249,250,251,252,253,254,255,256,257,258,259,260,261,262,263,264,265,266,267,268,
269,270,271,272,273,274,275,276,277,278,279,280,281,282,283,284,285,286,287,288,289,290]

# Bucle para seleccionar los vértices definidos en la lista vertices_Iris
for i in range(len(vertices_Iris)):
     obj.data.vertices[vertices_Iris[i]].select = True

bpy.ops.object.mode_set(mode = 'EDIT')

# Asignamos el material Iris a los vértices seleccionados
bpy.ops.object.material_slot_assign()
bpy.ops.object.mode_set(mode = 'OBJECT')
```

Tabla 31. Código de Creacion_Materiales_Ojo.py

12. Conclusión y futuros trabajos.

Con la realización de este proyecto de fin de grado se ha podido comprender cómo funciona la aplicación Blender, ha sido bastante interesante poder observar como una aplicación que a priori es solamente para creación de gráficos en 3D se puede utilizar para utilidades como es la de desarrollar un retrato robot con la ayuda de poder programar scripts de Python. Para la creación de dichos scripts ha sido necesario entender la geometría de las mallas, las cuales están formadas por vértices, aristas y caras y con estos elementos podemos crear mallas con la forma o volumen que deseamos. Donde hemos podido tomar el papel de desarrollador porque hemos creado dos mallas complejas para poder incorporar los ojos, hemos comprendido el funcionamiento de los nodos y las propiedades de algunos de ellos para poder crear materiales y la aplicación de ellos en la malla deseada. Y también hemos podido tomar el papel de programador, pues una vez entendido lo anterior nos ha servido para crear scripts que nos devuelvan los vértices seleccionados, que aplique materiales en las mallas deseadas a través de los cambios de modo de interacción del usuario con la mesh, la selección de ciertos vértices para su aplicación y la creación de un panel gráfico donde se incorporan botones los cuales hacen la acción que deseamos cuando se pulsan.

Como continuación en un futuro inmediato se podría seguir desarrollando la cara del sujeto genérico para poder implementar las transformaciones y los gestos de la forma más realista posible, aunque pueda ser tedioso el tener que entender partes de la anatomía humana estimo que es necesario para que esta herramienta que estamos desarrollando sea eficaz, por ejemplo, si en el próximo paso se crea una boca, con dientes y lengua se podrá realizar gestos más afines a la realidad y no sintiéndose forzados. Además de poder seguir avanzando con el menú que sirva de interacción de un usuario de la policía con el entorno de Blender para poder plasmar los rasgos característicos que va obteniendo del sujeto que se pretende identificar y seguir con la creación de los materiales que hacen falta para el busto como es la piel y diversos tonos de piel, el cabello y demás.

Como pasos futuros una vez obtenido el busto con sus respectivos órganos los cuales han de crearse por separado para añadirles movimiento propio, e independiente o para aplicar transformaciones del mismo modo para que no repercuta negativamente en el resto del retrato tridimensional, se podría continuar con el "menú" a nivel de código para poder implementar de alguna forma el control de los gestos y de transformaciones a nivel de código.

Haber podido continuar con un proyecto ya comenzado me ha dado la experiencia de poder adaptarme a las ideas de otro compañero e investigar y mejorarlas; espero y así lo hagan con el mío otros compañeros.

Realizar este proyecto ha sido una experiencia gratificante al poder aportar un apoyo al cuerpo policial para la identificación de sujetos en un entorno tridimensional. Ojalá esto sea el comienzo de un estudio y proyecto mucho más grande que llegue a ser usado para las fuerzas del Estado.

Haber podido continuar con un proyecto ya comenzado me ha dado la experiencia de poder adaptarme a las ideas de otro compañero e investigar y mejorarlas; espero y así lo hagan con el mío otros compañeros.

13. Bibliografía

(s. f.). Test Page for the HTTP Server on Red Hat Enterprise Linux. https://biblus.us.es/bibing/proyectos/abreproy/11511/descargar_fichero/PFC+Silvia+Blasco+Vadillo%2FCapítulo+9+-+Anexo+2.pdf+#:~:text=La%20Transformada%20Wavelet%20es%20un,imágenes%20médicas%20y%20señales%20biológicas.

10 Best Movies Made with Blender - Artisticrender.com. (s. f.). Artisticrender.com. https://artisticrender.com/10-best-movies-made-with-blender/

3Dilusion Arte Blender. (2019, 13 de agosto). Blender 2.8 español---Texturas procedurales--para principiantes [Video]. YouTube. https://www.youtube.com/watch?v=wW3o1pYZtXI

About — blender.org. (s. f.). blender.org. https://www.blender.org/about/Roosendaal

Blender Python Tutorial: Lists - Create, Append and Remove. (s. f.). Darkfall. https://darkfallblender.blogspot.com/2020/11/blender-python-tutorial-lists-create.html

Blender-Python: Select and iterate vertices after each other from left to rigth, top to bottom. (s. f.). Blender Stack Exchange. https://blender.stackexchange.com/questions/218410/blender-python-select-and-iterate-vertices-after-each-other-from-left-to-rigth

BMesh Module (bmesh) — Blender Python API. (s. f.). Blender Documentation - blender.org. https://docs.blender.org/api/current/bmesh.html

Bump Node — Blender Manual. (s. f.). Blender Documentation - blender.org. https://docs.blender.org/manual/es/3.2/render/shader_nodes/vector/bump.html

CG Cookie. (2020, 11 de marzo). Creating a Procedural Eyeball with Blender 2.8 [Video]. YouTube. https://www.youtube.com/watch?v=JcHX4AT1vtg

CGPython. (2022, 25 de abril). Beginner Blender Python Exercise: Repeating code with for loops [Video]. YouTube. https://www.youtube.com/watch?v=tj1ZmEdf9sA

Connecting an Operator with a Panel Float Slider. (s. f.). Blender Stack Exchange. https://blender.stackexchange.com/questions/245133/connecting-an-operator-with-a-panel-float-slider

Constraint(bpy_struct) — Blender 2.68.0 r58372 - API documentation. (s. f.). Blender Documentation - blender.org. https://docs.blender.org/api/blender_python_api_2_68_release/bpy.types.Constraint.html#bpy.types.Constraint

Cortés, J. (2018, 15 de agosto). Next Gen | La Nueva Generación - La nueva Pelicula de Animación de Netflix hecha en BLENDER - Notodoanimacion.es. Notodoanimacion.es | noticias, recursos, tutoriales y empleo para Artistas Digitales. https://www.notodoanimacion.es/next-gen-la-nueva-generacion-la-nueva-pelicula-de-animacion-de-netflix/#:~:text=hecha%20en%20BLENDER-,Next%20Gen%20|%20La%20Nueva%20Generación%20–%20La%20nueva%20Pelicula%20de%20Animación,de%20Netflix%20hecha%20en%20BLENDER

Darkfall. (2020, 10 de agosto). Blender Python Tutorial : How to create and assign a Material Shader [learn python for beginners] [Video]. YouTube. https://www.youtube.com/watch?v=eo7UjKFiagk

Darkfall. (2020a, 12 de enero). Blender Python Tutorial : An Introduction to Scripting [how to learn python for beginners] [Video]. YouTube. https://www.youtube.com/watch?v=cyt0O7saU4Q

Darkfall. (2020b, 9 de noviembre). Blender Python Tutorial : Lists - Create, Append and Remove [learn python for beginners] [Video]. YouTube. https://www.youtube.com/watch?v=5T0sSKH2_0o

Darkfall. (2020c, 3 de diciembre). Blender Python Tutorial : How to Display Info Messages [learn python for beginners] [Video]. YouTube. https://www.youtube.com/watch?v=TgQ9ahMuDMw

Darkfall. (s. f.). Darkfall. https://darkfallblender.blogspot.com/

Del retrato robot al reconocimiento facial: el rostro de los delincuentes. (s. f.). Crónica Global. https://cronicaglobal.elespanol.com/vida/retrato-robot-reconocimiento-facial-rostro-delincuentes_465973_102.html

Diferenciador. (2020, 11 de diciembre). Reflexión y refracción de la luz: en qué consisten y ejemplos. https://www.diferenciador.com/reflexion-y-refraccion-de-la-luz/#:~:text=La%20reflexión%20y%20la%20refracción,cambia%20su%20ángulo%20de%20propagación.

Distancia euclidiana: concepto, fórmula, cálculo, ejemplo. (s. f.). Lifeder. https://www.lifeder.com/distancia-euclidiana/

Efficient way to get selected vertices via python (without iterating over the entire mesh). (s. f.). Blender Stack Exchange. https://blender.stackexchange.com/questions/1412/efficient-way-to-get-selected-vertices-via-python-without-iterating-over-the-en

esclerótico, esclerótica | Diccionario de la lengua española. (s. f.). «Diccionario de la lengua española» - Edición del Tricentenario. https://dle.rae.es/esclerótico

Estudio y desarrollo de una aplicación de retrato robot en Blender. (s. f.). e_Buah Biblioteca Digital de la UAH. https://ebuah.uah.es/dspace/handle/10017/49496

Flip Normals (inverted faces) – Blender Knowledgebase. (s. f.). Blender Game, Design and Editing tutorials and tools : KatsBits.com. https://www.katsbits.com/codex/flip-normals/

Gary Heiting, OD (english). (2019, 27 de febrero). Sclera (White of the Eye). All About Vision. https://www.allaboutvision.com/resources/sclera.htm

Gradient Texture Node — Blender Manual. (s. f.). Blender Documentation - blender.org. https://docs.blender.org/manual/en/latest/render/shader_nodes/textures/gradient.html

How do I select specific vertices in blender using python script? (s. f.). Blender Stack Exchange. https://blender.stackexchange.com/questions/43127/how-do-i-select-specific-vertices-in-blender-using-python-script

How do I select specific vertices in blender using python script? (s. f.). Blender Stack Exchange. https://blender.stackexchange.com/questions/43127/how-do-i-select-specific-vertices-in-blender-using-python-script

How to put buttons, sliders, ect. in the N panel with an addon. (s. f.). Blender Stack Exchange. https://blender.stackexchange.com/questions/252083/how-to-put-buttons-sliders-ect-in-the-n-panel-with-an-addon

How to use node_tree.nodes.add_node(“ShaderNodeTexImage”). (s. f.). Blender Artists Community. https://blenderartists.org/t/how-to-use-node-tree-nodes-add-node-shadernodeteximage/1155487/3

Person identification system using an identikit picture of the suspect. (s.f) Janusz Bobulski and Mariusz Kubanek Czestochowa University of Technology. https://www.researchgate.net/publication/278179219_Person_identification_system_using_an_identikit_picture_of_the_suspect

import bpyfrom bpy.types import Panel, Operatorclass ADDONNAME_P - Pastebin.com. (s. f.). Pastebin. https://pastebin.com/QF83xPrc

Interpolation — Blender Manual. (s. f.). Blender Documentation - blender.org. https://docs.blender.org/manual/en/latest/grease_pencil/animation/interpolation.html

IntroducciÃ³n — Blender Manual. (s. f.). Blender Documentation - blender.org. https://docs.blender.org/manual/es/dev/editors/outliner/introduction.html

Introduction — Blender Manual. (s. f.). Blender Documentation - blender.org. https://docs.blender.org/manual/en/latest/advanced/scripting/introduction.html#:~:text=Python%20scripts%20are%20a%20versatile,of%20the%20tightly%20integrated%20API%20.

Introduction — Blender Manual. (s. f.). Blender Documentation - blender.org. https://docs.blender.org/manual/en/latest/render/workbench/introduction.html

Kolingerová, I. (s. f.). Deformation method for 3D identikit creation. IEEE Xplore. https://ieeexplore.ieee.org/document/7296037

Linnet's How To. (2016, 5 de diciembre). How To Show Tooltips In Blender [Video]. YouTube. https://www.youtube.com/watch?v=PesVZA8CxeI

Liz Segre/illustration by Steve Bagi - Spanish. (2019, 8 de marzo). Anatomía del ojo humano - Explicación de las partes del ojo. All About Vision. https://www.allaboutvision.com/es/recursos/anatomia-del-ojo.htm

Mallas | Blender: 3D en la educaciÃ³n. (s. f.). Cursos tutorizados en línea. https://formacion.intef.es/pluginfile.php/44656/mod_imscp/content/1/mallas.html

Mallas | Blender: 3D en la educaciÃ³n. (s. f.). Cursos tutorizados en línea. https://formacion.intef.es/pluginfile.php/44656/mod_imscp/content/1/mallas.html#:~:text=Las%20mallas%20son%20estructuras%20basadas,zona,%20borrar%20otra…

Mapping Node — Blender Manual. (s. f.). Blender Documentation - blender.org. https://docs.blender.org/manual/es/2.79/render/cycles/nodes/types/vector/mapping.html

Material Node — Blender Manual. (s. f.). Blender Documentation - blender.org. https://docs.blender.org/manual/es/3.2/render/shader_nodes/output/material.html

Modos de Objeto — Blender Manual. (s. f.). Blender Documentation - blender.org. https://docs.blender.org/manual/es/2.79/editors/3dview/modes.html

Next Gen Netflix 7723 Robot 4K #23380. (s. f.). 4K and 8K Ultra-HD Wallpapers | UHD PIXEL. https://uhdpixel.com/wall/next-gen-netflix-7723-robot-4k-x7810/

NodeTree(ID) — Blender Python API. (s. f.). Blender Documentation - blender.org. https://docs.blender.org/api/current/bpy.types.NodeTree.html

Nodo de Coordenadas de Texturizado — Blender Manual. (s. f.). Blender Documentation - blender.org. https://docs.blender.org/manual/es/3.2/render/shader_nodes/input/texture_coordinate.html

Nodo Marco — Blender Manual. (s. f.). Blender Documentation - blender.org. https://docs.blender.org/manual/es/3.2/interface/controls/nodes/frame.html

Nodo Mezclar — Blender Manual. (s. f.). Blender Documentation - blender.org. https://docs.blender.org/manual/es/3.2/render/shader_nodes/color/mix.html

Nodo Rampa de Color — Blender Manual. (s. f.). Blender Documentation - blender.org. https://docs.blender.org/manual/es/3.2/render/shader_nodes/converter/color_ramp.html

Noise Texture Node — Blender Manual. (s. f.). Blender Documentation - blender.org. https://docs.blender.org/manual/en/latest/render/shader_nodes/textures/noise.html

numpy get index where value is true. (s. f.). Stack Overflow. https://stackoverflow.com/questions/16094563/numpy-get-index-where-value-is-true

numpy.array — NumPy v1.23 Manual. (s. f.). NumPy. https://numpy.org/doc/stable/reference/generated/numpy.array.html

Óptica por la Cara | El ojo humano en números. (s. f.). Óptica por la Cara. http://opticaporlacara.com/2011/02/el-ojo-humano-en-numeros/

Preface: What is opengl? | openglbook.com. (s. f.). OpenGLBook.com | A Free OpenGL Programming Book. https://openglbook.com/chapter-0-preface-what-is-opengl.html

Principled BSDF — Blender Manual. (s. f.). Blender Documentation - blender.org. https://docs.blender.org/manual/en/latest/render/shader_nodes/shader/principled.html

Qué es Blender, características y formatos. (s. f.). Profesional Review. https://www.profesionalreview.com/2022/02/20/blender-que-es-y-para-que-se-utiliza/

Qué Es y Cómo Hacer Una Escena en Blender 2.8 | 12 Design Note. (s. f.). 12 Design Note. https://12design.top/cursos/blender/como-crear-una-escena-en-blender/

Released Scripts and Themes. (s. f.). Blender Artists Community. https://blenderartists.org/c/coding/released-scripts-and-themes/50

Requirements — blender.org. (s. f.). blender.org. https://www.blender.org/download/requirements/

Retratos Robot. (s. f.). Excon-Art Perito Calígrafo - Pericia Judicial. https://periciacaligrafica.org/retratos-robot/

Ruido Perlin (artículo) | Ruido | Khan Academy. (s. f.). Khan Academy. https://es.khanacademy.org/computing/computer-programming/programming-natural-simulations/programming-noise/a/perlin-noise

Stored Views — Blender Manual. (s. f.). Blender Documentation - blender.org. https://docs.blender.org/manual/en/latest/addons/3d_view/stored_views.html

Tema 3: Diagrama de Voronoi. (s. f.). http://asignatura.us.es/fgcitig/contenidos/gctem3ma.htm

Tengo venas rojas en los ojos, ¿es grave? (s. f.). Área Oftalmológica Avanzada. https://areaoftalmologica.com/blog/salud-ocular/venas-rojas-en-los-ojos/

tok.wiki. (s. f.). Faceta (geometría) Referenciasyenlaces externos. leer wikipedia con nuevo diseño. https://hmong.es/wiki/Facet_(geometry)

tok.wiki. (s. f.). Ruido de Worley Algoritmo basicoyVer también. leer wikipedia con nuevo diseño. https://hmong.es/wiki/Worley_noise

TypeError: 'list' object cannot be interpreted as an integer | bobbyhadz. (s. f.). Blog | bobbyhadz. https://bobbyhadz.com/blog/python-list-object-cannot-be-interpreted-as-an-integer#:~:text=The%20Python%20"TypeError:%20'list,an%20integer%20to%20the%20function.

Unit Cube Mesh is not defined. (s. f.). FEniCS Project. https://fenicsproject.discourse.group/t/unit-cube-mesh-is-not-defined/1273

vértice - Diccionario de Matemáticas | Superprof. (s. f.). Diccionario de Matemáticas | Superprof. https://www.superprof.es/diccionario/matematicas/geometria/vertice.html

Viewport Shading — Blender Manual. (s. f.-a). Blender Documentation - blender.org. https://docs.blender.org/manual/en/latest/editors/3dview/display/shading.html?highlight=solid

Viewport Shading — Blender Manual. (s. f.). Blender Documentation - blender.org. https://docs.blender.org/manual/es/2.80/editors/3dview/controls/shading.html

Voronoi Texture Node — Blender Manual. (s. f.). Blender Documentation - blender.org. https://docs.blender.org/manual/es/3.2/render/shader_nodes/textures/voronoi.html

Workspaces (Espacios de Trabajo) — Blender Manual. (s. f.). Blender Documentation - blender.org. https://docs.blender.org/manual/es/2.91/interface/window_system/workspaces.html

Yo-C チャンネル. (2021, 23 de septiembre). 有料テクスチャを使わずにBlenderで目を作成する [Video]. YouTube. https://www.youtube.com/watch?v=QmD3dln1ZQM

Universidad de Alcalá

Escuela Politécnica Superior

ESCUELA POLITECNICA SUPERIOR