Escuela Politécnica Superior de Jaén

Grado en Ingeniería Informática

Universidad de Jaén

Escuela Politécnica Superior de Jaén

Trabajo Fin de Grado

Herramienta 3D de Edición de Modelos Humanos Digitales para la Prevención de Transtornos Mentales.

Alumno: Erik Tordera Bermejo

Tutor: Antonio Jesús Rueda Ruiz
Dpto: Departamento de Informática.

Cotutor: Silvia Moreno Domínguez
Dpto: Departamento de Psicología

Septiembre, 2017

Universidad de Jaén

Escuela Politécnica Superior de Jaén

Departamento de Informática

Don Antonio Jesús Rueda Ruiz y Doña Silvia Moreno Domínguez, tutores del Proyecto Fin de Carrera titulado: Herramienta 3D de edición de modelos humanos digitales para la prevención de trastornos mentales, que presenta Erik Tordera Bermejo, autoriza su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, Septiembre de 2017

El alumno: Los tutores:

Erik Tordera Bermejo. Antonio Jesús Rueda Ruiz.

Silvia Moreno Domínguez.

Índice

Presentación.

En este TFG (Trabajo Fin de Grado) tratamos de crear una herramienta que ayude a la comunidad médica a detectar y prevenir posibles desórdenes alimenticios en los pacientes con los que se utilice, para ello, proponemos el desarrollo de una aplicación informática con la que el paciente sea capaz de modelar fácilmente tanto el cuerpo que desea tener como el cuerpo que cree tener y trás esto, poder ver la diferencia física entre ambos.

Rápidamente establecimos los requerimientos mínimos y deseables que esta aplicación debía tener, de los cuales hablaremos en la sección de "Requisitos", pero para alcanzar estos requisitos y obtener la aplicación final, tuvimos que hacer una investigación previa, en la cual tuvimos en cuenta multitud de artículos de los que hablaremos más adelante y de qué manera podríamos integrarlos en nuestro proyecto, uno de ellos, SMPL[7] recoge los modelos de la base de datos CISCO[8] y crea un modelo matemático que calcula el modelo de forma fácil y eficiente cambiando unas pocos argumentos, aunque no de la manera que queremos nosotros. Se nos ocurrió usar SMPL[7] para obtener nuestra solución final, para ello, realizando un pequeño estudio sobre las modificaciones que ocurren sobre el modelo al cambiar cada argumento, y los límites de los mismos, obtuvimos los valores llamados "estándar" con lo que generamos los .OBJ "Archivos de modelos 3D" con los humanos base para nuestra aplicación, además, subdividimos los resultados obtenidos de dos maneras diferentes: según a qué parte(s) del cuerpo afectan y según cómo las afectan.

Teniendo los datos obtenidos de la investigación con SMPL[7], sólo quedó diseñar e implementar el software de la aplicación, la interfaz gráfica de las mismas y la interconexión entre ambas partes, este proceso llevó un tiempo considerable, aunque nos sentimos orgullosos del resultado final, aunque aún es mejorable de diferentes maneras.

Idea original.

En el momento de asignación del TFG, concebimos la idea como una colaboración entre el departamento de informática y el de psicología de la Universidad de Jaén para crear una herramienta que ayudara a los doctores a prevenir sobre todo desordenes alimenticios en los pacientes que utilizaran la aplicación.

En la idea original, el modo de empleo era el siguiente:

1. El doctor le presenta la aplicación al paciente.
2. El paciente modela el cuerpo que cree tener.
3. El paciente modela el cuerpo que desea tener.
4. Apoyado en estos modelos, el doctor puede detectar más fácilmente factores de riesgo antes de que el paciente caiga enfermo.

Basados en esto, propusimos unos requisitos mínimos e ideales.

Requisitos.

Definir los requisitos de la aplicación fué lo primero que hicimos cuando empezamos nuestro proyecto. Este paso era realmente importante para precisar el perfil de nuestra aplicación ya que de otra forma, podríamos haber perdido mucho tiempo haciendo cara a contradicciones en el diseño.

Mínimos.

Los requisitos mínimos se definen como los requerimientos indispensables para nuestra aplicación, sin ellos cumplidos, el objetivo de este proyecto no hubiese sido realizado.

- La aplicación tiene que producir resultados creíbles.

Es totalmente necesario para el cumplimiento del objetivo que la aplicación produzca resultados mínimamente realistas para que el paciente se vea reflejado en el modelo que produce.

- La aplicación debe de ser usable.
  Para que los pacientes se sientan cómodos usando la aplicación, esta tiene que ser rápida y sin interrupciones.

- La aplicación debe de ser intuitiva
  El tiempo desde que el usuario inicia por primera vez la aplicación hasta que aprenda a usarla debe de ser mínimo.

Ideales.

Los requisitos ideales son las metas que definimos para este proyecto y, como su nombre indica, sería el objetivo final de trabajo el cumplirlos todos.

- La carga de la aplicación debe de ser rápida.
  El usuario no puede perder mucho tiempo esperando a que la aplicación inicie.

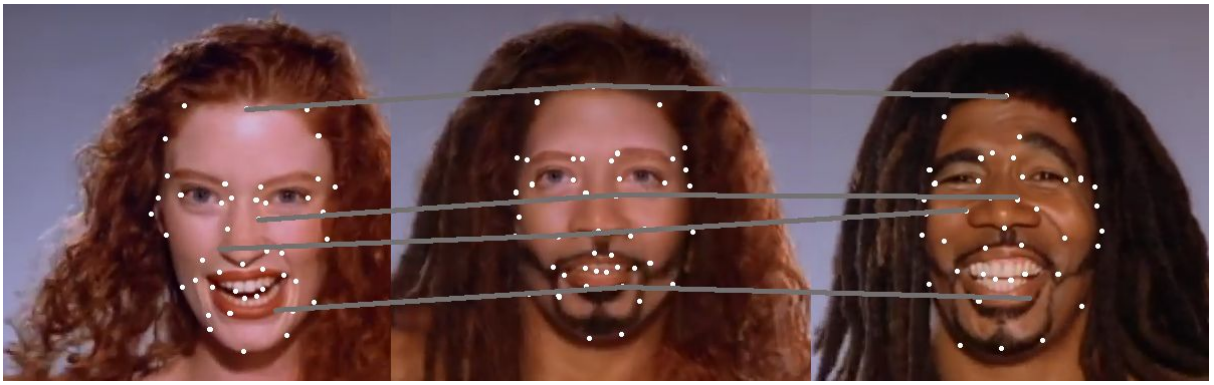
- La aplicación debe de ser fluida.
  Además de ser una aplicación usable, también debería llegar a una tasa de refresco de 60 imágenes por segundo para asegurar la comodidad del usuario.

- El manejo de memoria de la aplicación debe de ser eficiente.
  No debemos dejar que el uso de memoria de la aplicación supere el gigabyte ya que esto puede provocar el mal funcionamiento de la misma y del ordenador que la ejecute, según la potencia del mismo.

Proceso de Investigación.

La investigación fué la parte que más tiempo consumió del proyecto, también es la más importante que hemos llevado a cabo. Nuestras primeras ideas, fueron dirigidas hacia un proceso de morphing.

Morphing es una técnica de deformación que se puede aplicar tanto a elementos 3D como en nuestro caso como elementos 2D, se basa en, en dos elementos extremos colocar puntos ancla, así, se hace muy sencilla la interpolación de los puntos de cualquier elemento intermedio a los extremos escogido. Un ejemplo de esta técnica se utiliza en el videoclip "Black or White, de Michael Jackson".



Tres fotogramas del videoclip "Black or White" de Michael Jackson [5] editados para mostrar la técnica de Morphing.

Al empezar a estudiar diversas investigaciones que abajo mencionamos, observamos que apuntábamos en la dirección correcta y descubrimos que una de las maneras de llevar a cabo nuestro proyecto con solvencia es usar *Morphing*.

En [2] se nos muestra cómo por medio del *Morphing* somos capaces de deformar un modelo 3D humanoide en tiempo real. usando una base de datos de modelos 3D y un algoritmo basado en puntos ancla (Puntos situados manualmente en los mismos lugares sobre diferentes modelos). Aunque la investigación de [2], se centra más en el torso del paciente y las formas que deberían generar al cambiar la posición del mismo, según esté sentado, saltando o cualquier otra opción.

En [3] utilizan una técnica bastante más compleja, basada en la generación de esqueleto, sobre este, implantar primitivas que generan músculos mediante meta-bolas. (Las meta bolas, son esferas de diferentes tamaños que mediante un algoritmo se unen formando estructuras más orgánicas y complejas. [4])

En el Nº. 1 (sept./oct. 1992) de "Tech Images Internationales", se habla sobre "meta ball", una primitiva única que crea imágenes con superficies suaves.

En [1] la tesis trata sobre edición de un modelo 3D humanoide por partes usando funciones de deformación, en las cuales profundizaremos de nuevo algo más tarde, estas se basan en un conjunto de dos a cinco funciones que se aplican a cada uno de los vértices de la parte afectada, los resultados no son tan realistas, pero es un algoritmo mucho más rápido ya que no utiliza información de otros modelos, si no simples funciones matemáticas.

Primeros pasos.

Como ya hemos mencionado arriba, en un primer momento nos decidimos por continuar sobre la línea del morphing y de [2] ya que nos parecía una manera más natural de modelar en tiempo real un ser humano realista que [1] y más sencilla que [3].

La idea original consistía en obtener una base de datos de modelos 3D de seres humanos ya escaneados, estos modelos hemos de arreglarlos, ya que, normalmente, al escanear un modelo 3D suelen quedar huecos donde el escaner no alcanza, y por lo tanto, no obtenemos geometría en ese espacio como las plantas de los piés. Esto se realiza con software de edición 3D como Blender[11] o Maya[10].

Tras solventar el problema de los modelos incompletos, deberíamos, también manualmente disponer los puntos anclas con los que apoyar el Morphing. Solo quedaría entonces, dividir los

puntos anclas en las partes del cuerpo y aplicarle pesos a los vértices por cada parte del cuerpo (Así aplicamos el morphing por partes y de manera suave)

Este método quedó descartado debido a varias cosas:

CAESAR.

CAESAR[8] es la única base de datos de modelos humanos 3D realistas que encontramos a la que pudiésemos acceder, pero motivos económicos no nos permitieron llevar a cabo está metodología.

CAESAR ofrece 2 productos principales que nos interesaban, la base de datos de modelos 3D humanos escaneados de Estados Unidos de América y de Europa, pero el precio nos impidió usarlo, esta es la principal razón para cambiar de método y dejar el Morphing a un lado.

Tiempo.

Entre ambas bases de datos (modelos europeos y estadounidenses) tendríamos que considerar 4.400 modelos diferentes y, por lo tanto arreglarlos todos y colocar los puntos anclas en todos, un trabajo que no se hacía posible con una sóla persona.

Debido a la falta de fondos y de tiempo, dejamos la idea del Morphing de lado aunque, algunos conceptos volverían a aparecer más tarde.

Línea definitiva.

Debido a la falta de fondos y de tiempo, nos vemos obligados a volver al inicio de la investigación para encontrar otro método que se adaptara a lo que nosotros necesitábamos, el algoritmo expuesto en [3] nos pareció demasiado complejo para llevarlo a cabo por una sola persona sin conocimientos de anatomía por lo que volvimos a [1], un método “fácil” y entendible.

Análisis de la metodología de [1]

En Parameterized human body model for real-time applications[1] se nos muestra cómo es posible deformar un modelo 3D "estándar" (llamamos modelo estándar al modelo original que vamos a deformar) de forma realista usando funciones matemáticas.

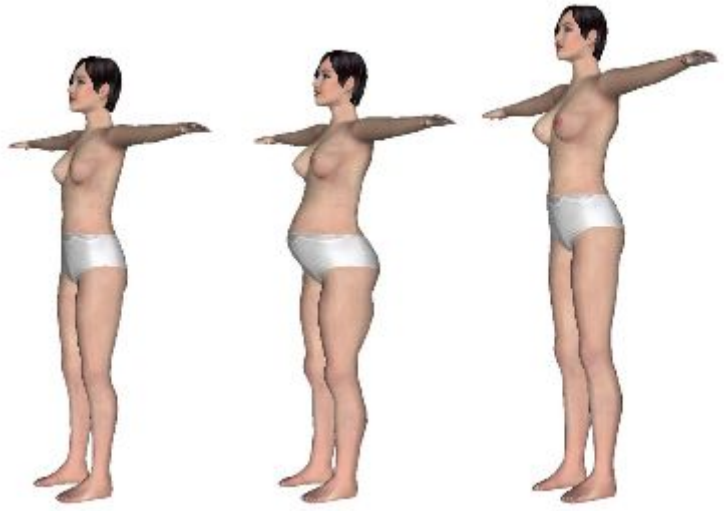


Imagen extraída de [1] (Figura 9) Modelo estandard y clones deformados.

Estos resultados nos parecieron lo suficientemente realistas como para cumplir el requisito de realistas, aunque creíamos que podríamos mejorarlo de alguna manera para lograr salidas más convincentes.

Con esta metodología debemos subdividir el cuerpo en las zonas que deseamos deformar, en el paper original subdividen su modelo en 24 partes, a nosotros nos pareció algo excesivo, ya que no queremos tanta exactitud en la deformación, si no algo más inteligible para el usuario.

Con el modelo subdividido, debemos asignar a cada una de las partes entre dos y cinco funciones de deformación que al sumarlas de la manera que expresan (No es relevante para nosotros) y aplicadas a los vértices de cada parte estos son modificados acorde a las funciones, consiguiendo así que cada parte crezca de forma diferente.

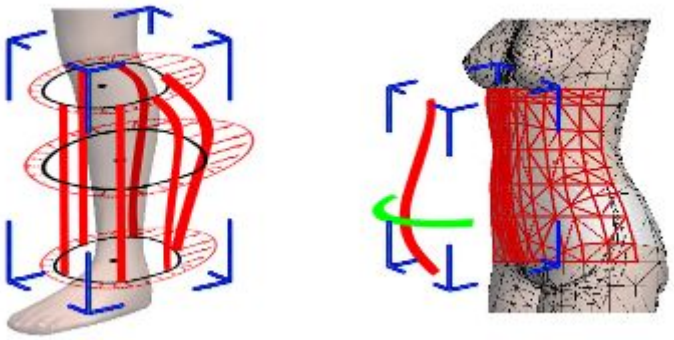


Imagen extraída de [1] (Figura 8) Funciones de deformación en los planos horizontal y vertical

Aquí encontramos el primer problema para nuestro método, las funciones matemáticas están basadas en la anatomía del cuerpo humano y la persona que realiza este trabajo no tiene grandes conocimientos sobre ella, además, no se expresa en ningún momento tampoco el procedimiento para hallar las susodichas funciones. Este método ha sido referenciado multitud de veces, por lo que indagamos en esos trabajos que lo referencian para intentar averiguar cómo diseñar esas funciones, pero en ninguno se profundiza en ellas y por lo tanto, nos vemos obligados a hallar otra manera de deformar el cuerpo.

Es en este momento es cuando llega a nuestros oídos SMPL[7] del que hablaremos algo más abajo. SMPL[7] resuelve nuestro problema de la deformación y nos permite avanzar hasta el siguiente punto.

Al deformar las partes del cuerpo por separado, se generan escalones entre las partes alteradas y las que no, por lo que es totalmente necesario un algoritmo de suavizado.

Dado que esta investigación no nos produce grandes problemas, nos decidimos por ella.

SMPL

MPI - SMPL[7] es un modelo 3D de un ser humano totalmente editable, tanto en forma como en posición, en la investigación de este proyecto, descubrimos que está basado en la base de datos de CAESAR[8] la cual pretendíamos usar nosotros para llevar a cabo Morphing entre los modelos de esta. SMPL recoge los datos de CAESAR[8] y con ellos genera dos conjuntos de datos (uno para modelos femeninos y otro para modelos masculinos) con el mismo número de vértices y con la misma topología (Cómo los vértices están conectados entre sí) y otro conjunto más para definir las posiciones de los modelos, con todo este trabajo, entrenan los dos modelos estándar para minimizar el error generado entre los modelos estándar modificados para reconstruir los modelos de los conjuntos de datos y las mallas originales.

"We train the SMPL model parameters to minimize reconstruction error on two datasets."
Paper de MPI-SMPL[7] página 6, punto 4.

Nosotros usamos la versión de SMPL[7] que es ejecutada en el lenguaje de programación Python[6] ya que nos parece ideal para hacer la recolección de datos previa a la implementación de la aplicación que necesitaremos, pero también es posible obtener una versión compatible con el motor gráfico Unity[9] y para Maya[10].

En la versión de Python[6] de SMPL[7] es realmente fácil generar nuevos modelos 3D, ya que SMPL sólo utiliza dos argumentos que rellenar, una lista de valores de formas (shapes) y otra lista de valores de posición (poses), las poses no son de nuestro interés, ya que siempre usaremos la pose por defecto, por otra parte la lista de valores a las formas tuvimos que investigarla a fondo.

Lista de valores de forma(shapes)

La lista de valores de forma se utiliza para cambiar la forma de todo el cuerpo de diferentes maneras, esta lista tiene diez posiciones (0 ~ 9) las cuales cambian, como hemos dicho, la totalidad del cuerpo de diferentes maneras cada una.

- [0] Altura general del modelo.
- [1] Grasa general del modelo
- [2] Produce deformaciones
- [3] Altura de las piernas
- [4] Valores varios del tronco*
- [5] Valores varios del pecho*
- [6] Valores varios de la cadera*
- [7] Valores varios del tronco*
- [8] Valores varios del pecho y vientre alto*
- [9] Longitud del cuello

**Estos valores serán desgranados al implementarlos en la aplicación.*

Como se puede observar, muchos de los valores se pueden extrapolar a que afecten a una o varias partes del cuello, aunque de forma normal se aplican a la totalidad del mismo.
El valor [2] lo ignoramos ya que produce deformaciones en la parte en la que más afecta, la cual es la clavícula, trasladando el cuello y cabeza a la derecha demasiado, hasta el punto en el que queda en la posición del hombro del modelo.

Datos de interés sobre SMPL

Además de SMPL, MPI-SMPL[7] provee también de otro estudio de alto interés:

DMPL[12] - (Dynamic SMPL) Funciona con SMPL calculando los cambios que se efectúan en el cuerpo según la posición del mismo para evitar clipping (Superposición de partes del modelo sobre el mismo) y variando la forma que genera, creando así resultados altamente realistas. No nos interesa ya que en nuestro proyecto el modelo no cambia de posición

Obtención de datos.

Antes de diseñar la aplicación debemos saber exáctamente cómo queremos que esta funcione, por ahora, todo lo que sabemos es que queremos usar la metodología de [1] de la deformación por partes del cuerpo y usaremos SMPL[7] para obtener las funciones de deformación.

En un primer momento se nos ocurrió obtener una base de datos de modelos divididos por partes utilizando SMPL[7], esto sería una tarea sencilla:

1. Investigar los límites máximos y mínimos del realismo de la lista de formas.
2. Generar unos modelos masculino y femenino a partir de los valores medios de cada forma, estos modelos serán los llamados estándar.
3. Dividir los modelos estándar en partes.
4. Generar a partir de las partes estándar otras partes con los argumentos de la lista de formas modificadas.

En este momento tendríamos una base de datos de modelos tremendamente grande con la que trabajaríamos usando *Morphing*, con cada vértice como punto ancla y como transformaciones

interpolaciones lineales, ya que al ser todos los vértices puntos anclas, no nos es necesario algo más complejo.

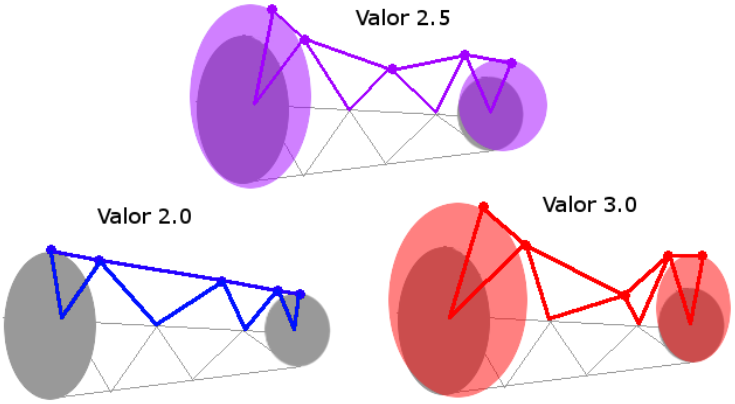


Figura de la modificación de dos vértices por medio de interpolación lineal.

Pero este método nos provoca un problema difícilmente eludible, que es la carga en memoria de los modelos. Además de llevar mucho tiempo, también es posible que haga la aplicación poco eficiente cuando hablamos de la memoria que podría llegar a ocupar, por lo que pensamos una manera diferente de llevar a cabo el mismo método.

La idea de que hizo todo encajar fué cambiar la base de datos de los modelos de partes del cuerpo por el desplazamiento producido en los vértices a cada paso de los argumentos, de esta manera nos quedaría una aplicación con un coste de carga considerable, pero más optimizado, ya que en lugar de abrir miles de ficheros de vértices, sólo tendremos que abrir y manejar nueve ficheros (uno por cada posición de la lista de formas de SMPL[7]) aunque estos serán masivos (desde los 15Mb hasta los 56Mb) lo que ralentizará el inicio de la aplicación.

Por lo tanto, obtener los datos previos y necesarios para el funcionamiento quedará así:

1. Investigar los límites máximos y mínimos del realismo de la lista de formas.

2. Generar unos modelos masculino y femenino a partir de los valores medios de cada forma, estos modelos serán los llamados estándar.
3. Dividir los modelos estándar en partes.
4. Generar los archivos de desplazamiento entre vértices con un script. (Pequeño programa, normalmente ejecutado en consola, que no suele tener más de un archivo de código)
5. Generar los archivos, con otro script, que nos especifique qué vértices se incluyen en qué partes del cuerpo.
6. Definir qué partes del modelo son afectadas por cada argumento de la lista de formas de SIMPL[7]
7. Cotejar los archivos de desplazamiento entre vértices con los archivos de vértices en cada parte del modelo, de manera que los archivos de desplazamiento entre vértices por forma queden reducidos a únicamente los vértices que nos interesan.

Investigación de los límites de las formas de SMPL

Llevar a cabo el trabajo de limitar las formas de SMPL no fué complejo. Nos basamos en coger un pequeño script escrito en Python “ebermejo_getinfo.py”, en este programa, introducimos el género del modelo resultante, el índice de la forma a probar (0 ~ 9) y un par de argumentos más el mínimo a probar y el máximo. De esta manera, se guardan dos modelos con el valor mínimo y máximo de la forma. Nosotros los visualizamos en Blender[11] de manera que podemos observar las diferencias fácilmente. De esta forma, ajustamos los valores máximos y mientras los modelos resultantes fuesen deformes. Normalmente usando -50.0 para el mínimo y 50.0 para el máximo. De esta manera obtuvimos la siguiente lista de formas con los límites máximos, mínimos y el valor medio entero.

```
0 5 -5 0
1 3 -4 -1
2 7 -7 0
3 10 -10 0
```

```
4 10 -10 0
5 7 -10 -2
6 7 -8 -1
7 12 -7 3
8 15 -15 0
9 15 -15 0
```

Archivo "ebermejo_ins/ebermejo_datasmpl.txt"

De manera que la primera columna especifica el índice de la forma de la que hablamos, la segunda nos dice los valores máximos para cada forma, la tercera el mínimo y la cuarta el valor medio entero.

Generación de los modelos masculinos y femeninos estándar

En esta fase sólo tuvimos que recoger los valores medios de "ebermejo_ins/ebermejo_datasmpl.txt" y generar los los modelos(.obj) asociados a ellos, tanto el masculino como el femenino.

División en partes de los modelos

En [1] proponen la siguiente división del cuerpo, generando 24 partes diferenciadas.

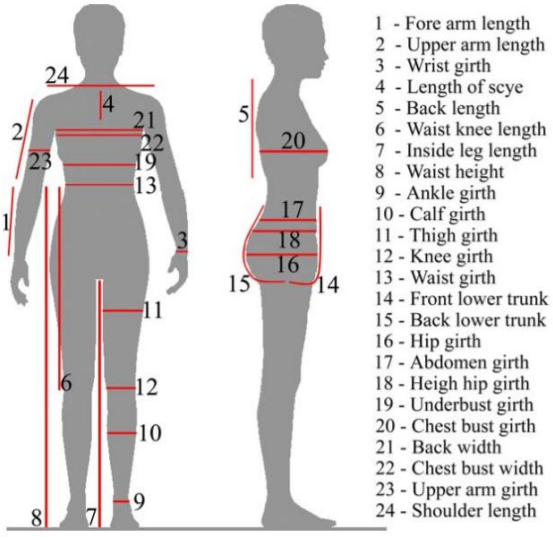
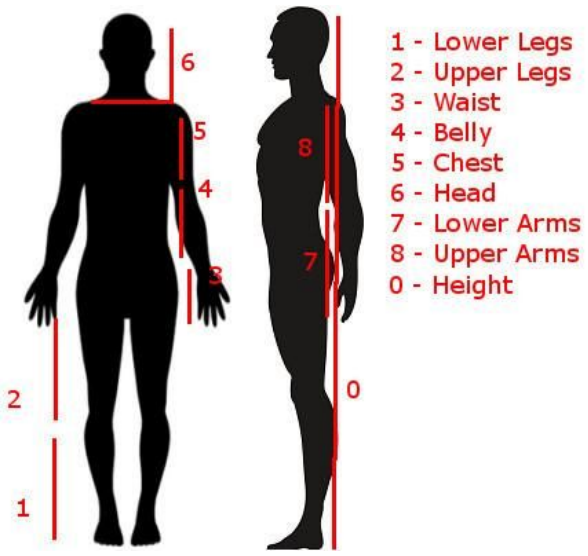


Imagen extraída de [1] (Figure 1) Donde se muestra una proposición de división del cuerpo humano.

Para nosotros esto es excesivo, ya que si le pedimos al usuario que modifique cada una de las partes por separado la aplicación se volverá algo incómoda de usar, por lo tanto decidimos proponer otra, menos específica, pero no es necesario para nosotros ser tan específicos.



Nuestra proposición de división del cuerpo.

Aunque, más tarde, al implementarlo, alguna deformación actúa sobre varias partes a la vez, generando así nuevas partes, como por ejemplo el pecho y el vientre deformandose juntos para modificar la espalda.

En nuestra división se simplifican las originales de [1] haciendo que las partes 14, 15, 16, 17, 18 funcionen como una solo siendo esta la cadera (Waist) , 11 y 12 se unen y forman la parte de arriba de las piernas (Upper Legs), así como 9 y 10 forman la parte de abajo de las mismas (Lower Legs), 2 y 23 forman la parte de arriba de los brazos (Upper Arms), 13, 19, 20, 21 y 22 son el pecho (Chest) y 4 y 24 forman la cabeza (Head).

El troceado del modelo lo hicimos usando Blender[11] de forma que ningún vértice se incluya en más de una parte para evitar problemas al modificarlos. Además, también eliminamos todo el modelo de la parte excepto los bordes de la misma con otras para incluir esta información para el algoritmo de suavizado.

Generar los archivos de desplazamiento entre vértices.

Estos archivos deben de reflejar cómo varía un mismo vértice en una forma(shape) de SMPL[7] a lo largo de todo el espectro que hemos definido anteriormente en *"ebermejo_ins/ebermejo_datasmpl.txt"* para ello podríamos hacerlo de dos maneras:

- Calcular el desplazamiento entre el mismo vértice en el modelo estándar y en el que se está evaluando
- Calcular el desplazamiento entre el mismo vértice en el modelo anterior al que se está evaluando y el actual, comenzando desde el estándar.

Entre estas dos maneras nos decidimos por la primera, ya que en nuestra aplicación podríamos decidir el valor de cada parte del cuerpo para cada forma de forma discreta y continua, por lo tanto, es mucho mejor calcular la diferencia entre el modelo estándar y el que se está evaluando en el momento.

Para calcular estos desplazamientos programamos el script "*ebermejo_getdiff.py"* este script lee y abre el archivo de rangos (*"ebermejo_ins/ebermejo_datasmpl.txt")* y, dejando abiertos los modelos estándar del varón y la mujer, generan los todos los modelos para cada forma por separado que se incluyen en el rango de la misma y calcula los desplazamientos en porcentaje de cada vértice del modelo estándar con sí mismo en el modelo que está siendo evaluado de manera tal que:

$$\alpha[i] = \frac{v_t[i]}{v_o[i]}$$

Siendo alpha los desplazamientos en formato (desplazamiento en X, desplazamiento en Y, desplazamiento en Z) y v_t el valor del vértice del modelo siendo probado y v_0 el valor del vértice en el modelo estándar.

De esta manera se generan 9 ficheros de la forma:

```
588 1.03614250044 1.03659279253 1.1162946352 1.03578107544 1.0362268646 1.11513168885...
589 1.02723602846 1.05048415313 1.06207398001 1.02696366818 1.0499793116 1.06145324021...
590 1.02964997135 1.05075796511 1.09574256127 1.02935347164 1.05025038546 1.09478513566...
591 1.0404125516 1.03225831471 1.17152555649 1.04000842609 1.03193573157 1.16981030092...
592 1.02941103329 1.0685794304 1.06634929951 1.02911692295 1.06789363609 1.06568580651...
```

Extracto del fichero "ebermejo_out_getdiff_m_shape_9.shadif" donde se muestran 5 vértices y algunos cambios que sufren durante el cambio de la forma 9.

En estos archivos, la primera columna se conforma del índice de los vértices a los que hace referencia el resto de línea y (n * 3) porcentajes que expresan el desplazamiento en cada una de las componentes. (Siendo n el número de modelos generados, el rango * 10, ya que los generamos en intervalos de 0.1 unidades.

La extensión de estos archivos es ".shadif" y son masivos, ya que la información que recogen aún no está optimizada, pueden ocupar hasta los 80Mb.

Generar los archivos de vértices en cada parte.

Con el script "ebermejo_getvertexs.py" de el modelo estándar origen de la parte, la propia parte en formato y los bordes de la misma (todo en formato .obj) cotejamos los datos e incluímos en el fichero de salida todos índices de los vértices del modelo estándar y además, todos los vértices que además estuviesen en los bordes del modelo de partes, se le incluyen una 'B' de manera que sabemos que es un borde de la parte, esto genera un archivo como el siguiente al que llamamos vertin (por la extensión que le dimos al fichero).

```
808
809
807
```

```
810
816
815
822
823
830 B
831 B
833 B
834 B
832
```

Extracto del fichero "ebermejo_outs/ebermejo_m_waist.vertin"

En la primera columna se indica el índice del vértice que pertenece a la parte del cuerpo correspondiente al nombre del archivo y en la segunda se especifica si ese vértice es parte del borde de la parte o no.
Como vemos en el nombre del archivo, se especifica el género 'm' de "male" (varón en inglés) y la parte del cuerpo a la que hace referencia.

Se generaron 18 elementos de la clase ".vertin" siendo los vértices correspondientes a cada parte de cada género.

Definir qué partes del cuerpo se ven afectadas por cada forma(shape) de SMPL

Definir qué parte del cuerpo es afectada por qué forma(shape) de SMPL[7] no fué una tarea sencilla, dado que no disponemos de un modelo matemático que nos lo diga sin lugar a duda, podríamos haber escrito un script en el cual calcularemos el potencial cambio por cada parte del cuerpo con respecto al modelo estándar, pero en ese caso, estaríamos ignorando las posibles deformaciones no deseadas que se pueden producir, por lo tanto, decidimos que la mejor manera fué observar cuidadosamente los cambios realizados por las diferentes formas a las diversas partes del cuerpo.

Varias de las formas de SMPL[7] fueron simples de localizar, ya que o afectan al cuerpo entero de la misma manera o a partes muy localizadas del mismo.

- [0] Afecta a todo el cuerpo de manera uniforme como la altura del modelo.
- [1] Afecta al modelo de manera uniforme (Aunque dividiremos en las diferentes partes) como grasa almacenada en el.
- [2] Afecta a todo el torso, ya que principalmente actúa sobre la espalda del modelo, pero, como ya hemos dicho, este argumento lo excluimos por causar deformaciones.
- [3] Afecta a las piernas como su altura, y por deformación, a la cadera y al vientre del modelo.
- [4] Son diversas deformaciones del torso, afectando así a la cabeza, al pecho, al vientre y a la cadera.
- [5] Son diversas deformaciones del torso, afectando así al pecho, al vientre y a la cadera.
- [6] Son diversas deformaciones del torso, afectando así al pecho, al vientre y a la cadera.
- [7] Son diversas deformaciones del torso, afectando así al pecho, al vientre y a la cadera.
- [8] Deforma al volumen del pecho, principalmente, y por deformación, al vientre.
- [9] Son diversas deformaciones del torso, afectando así a la cabeza, al pecho, al vientre y a la cadera.

Reducir los archivos ".shadif" usando las partes que afectan a cada forma de SMPL

Teniendo en cuenta que formas de SMPL(shapes) afectan a qué partes se nos hace muy fácil reducir los archivos ".shadif", recordemos que pueden llegar a ocupar 80Mb en el estado actual, por lo tanto, es imperativo reducirlo para mejorar el rendimiento y la velocidad de carga de nuestra aplicación, con este objetivo, escribimos el script "ebermejo_reduceshadif.py" que recibe un numero variable de argumentos de entrada. siendo el primero siempre el archivo ".shadif" que deseamos reducir y el último siempre el archivo de destino ".shadif", todos los argumentos

intermedios, los las partes del cuerpo que se ven afectadas, representadas por los archivos ".vertin".

"ebermejo_reduceshadif.py" coteja cada vertice del archivo ".shadif" original con todos los vértices de cada ".vertin" que le pasamos, eliminando todas las líneas del archivo original, cuyo vértice vinculado no aparezca en ninguno de los archivos ".vertin" que le pasamos, reduciendo drásticamente el tamaño del archivo original. En los mejores casos, reducimos un archivo de 80Mb de información a uno de 15Mb y en los peores casos (aquellos que usan todas las partes del cuerpo) no se mejora de ninguna manera.
Usando este script, reducimos el tamaño total de la carpeta de recursos desde los 1200Mb(aprox.) a unos 600Mb.

Unity como motor gráfico.

En este momento debíamos empezar a diseñar el software que finalmente llegaría a ser nuestra aplicación, pero antes de eso tendríamos que elegir si la desarrollaríamos desde cero o si por el contrario, usaríamos algún framework o aplicación ya programada que nos ayudase en la tarea. Debido a que la programación de un motor gráfico estable desde cero lleva mucho tiempo, nos decidimos por usar el motor Unity[9] como base de nuestro trabajo.

Elegimos Unity[9] por razones económicas, dado que es el único motor gráfico gratuito, junto con Unreal Engine[13] pero ya habíamos trabajado con el primero, además de que no sólo nos serviría como motor gráfico 3D, sino que también como plataforma donde crear nuestra interfaz de usuario.

En Unity[9] se nos permite trabajar con dos lenguajes de programación simultáneamente, estos son C# y JavaScript, aspecto que nos era realmente atractivo ya que son lenguajes que ya dominábamos.

Aún así, necesitábamos alguna forma de continuar llevando un control de versiones eficiente aunque trabajásemos con un motor gráfico ya que aunque Unity[9] ofrece su propio control de versiones, no era suficiente, en nuestro caso, ya que debíamos también almacenar archivos de fuera del proyecto, como los scripts de python anteriores. Para este problema, Unity[9] nos ofrece una potente solución, los paquetes (packages) los cuales encapsulan todas las dependencias del proyecto en un sólo archivo comprimido, el cual, agregándolo otra vez al motor, automáticamente se regenera todo lo que tendríamos hecho en el caso de que tuviésemos creados las instancias llamadas "prefabs", este paquete, lo agregaremos al repositorio ya creado y tendríamos todo nuestro proyecto bajo un mismo control de versiones.

Diseño de la aplicación

Sabiendo ya cómo y con qué trabajaremos, debemos diseñar el software que programaremos, y para esto, es imperativo tener claros los **Requerimientos** específicos de la aplicación.

Requerimientos de la aplicación.

"Propiedades o restricciones determinadas de forma precisa que deben satisfacerse."

Definición de requerimientos según [14], diapositiva 4.

Funcionales.

1. La aplicación debe de ser capaz de modificar un modelo humano 3D por partes de manera lo suficientemente realista como para poder identificarse con el resultado.

No funcionales.

1. La aplicación debe de tener un tiempo de carga inferior a 5 minutos.
2. La ejecución de la aplicación debe de ser ininterrumpida y fluida.
3. Las deformaciones realizadas deben de de ser totalmente fluidas (Obtener una velocidad mínima de refresco de pantalla durante la deformación de 30 imágenes por segundo)
4. La aplicación debe de poder tener una rápida distribución,
5. La aplicación debe de ser operable sin acceso a internet.
6. Trás el trabajo realizado no debe de quedar rastro de él/los modelos generados para asegurar que esta quede privada entre paciente - doctor.
7. La aplicación debe de ser capaz de guardar un modelo en memoria para, sin modificar este, empezar a modificar un segundo.
8. La aplicación debe de permitir calcular la distancia entre un modelo anterior y el que está siendo modificado actualmente.
9. La aplicación debe de permitir el cambio entre sexos.
10. La aplicación debe de tener una amplia gama de deformaciones a las distintas partes del cuerpo.
11. La aplicación debe de contener un botón de reinicio en el caso de que el usuario no está de acuerdo con el resultado producido.
12. La aplicación debe de cumplir con la legislación española sobre pruebas de software con humanos.
13. La aplicación debe de ser fácilmente puesta a prueba y usada por individuos sin conocimientos informáticos.

De ahora en adelante nos referiremos a estos requerimientos como la inicial del tipo: **F para funcionales** y **N para no funcionales seguida por el número del requerimiento.**

*p.e: **NF7** ("La aplicación debe de ser capaz de guardar un modelo en memoria para, sin modificar este, empezar a modificar un segundo.")*

Más adelante, comprobaremos si cada uno de estos requerimientos han sido cumplidos.

Diseño de la estructuras de datos del programa.

Dado que una de las cosas que más problemas nos puede producir es el manejo de información, porque debemos mantener cargada en memoria una gran cantidad de ella al mismo tiempo, las estructuras de datos de nuestra aplicación serán cruciales.

"En la práctica, la mayor parte de información útil no aparece aislada en forma de datos simples, sino que lo hace de forma organizada y estructurada. Los diccionarios, guías, enciclopedias, etc., son colecciones de datos que serían inútiles si no estuvieran organizadas de acuerdo con unas determinadas reglas. Además, tener estructurada la información supone ventajas adicionales, al facilitar el acceso y el manejo de los datos. Por ello parece razonable desarrollar la idea de la agrupación de datos, que tengan un cierto tipo de estructura y organización interna."

Fragmento extraído de [16](Página 171) donde se explica resumidamente el porqué y para qué de las estructuras de datos.

En nuestra aplicación usaremos "dos" estructuras de datos diferentes que trabajarán entre ellas, Una principal en la que almacenaremos los datos, y una secundaria donde almacenaremos cómo se relacionan entre ellos.

Antes de explicar cada una de ellas y el porqué de esa elección, deberíamos exponer los datos que estaremos manejando, en las dos siguientes páginas presentamos un diagrama UML explicando las clases utilizadas y cómo se relacionan entre ellas. Las clases "*AVL_tree"* y "*AVL_node"* no se encuentran desgranadas dado que las vamos a explicar por separado en la siguiente sección.

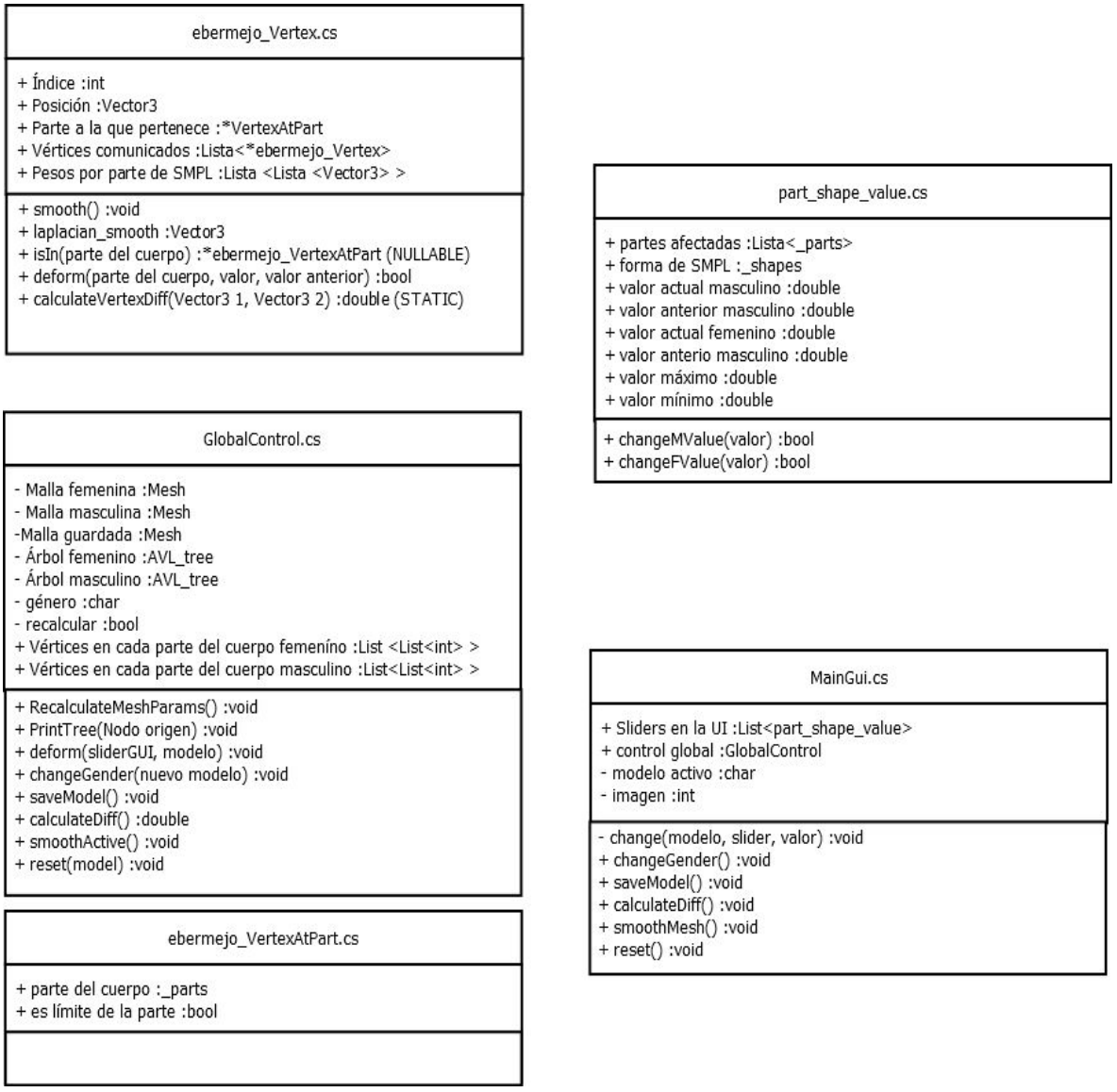


Diagrama de clases de nuestra aplicación. Creado con [15].

Nota: Los tipos de datos que empiezan con un guión bajo no son tipos de datos, si no que son de tipo enum e identifican normalmente una posición en un array que contiene información relevante para diversas funciones.

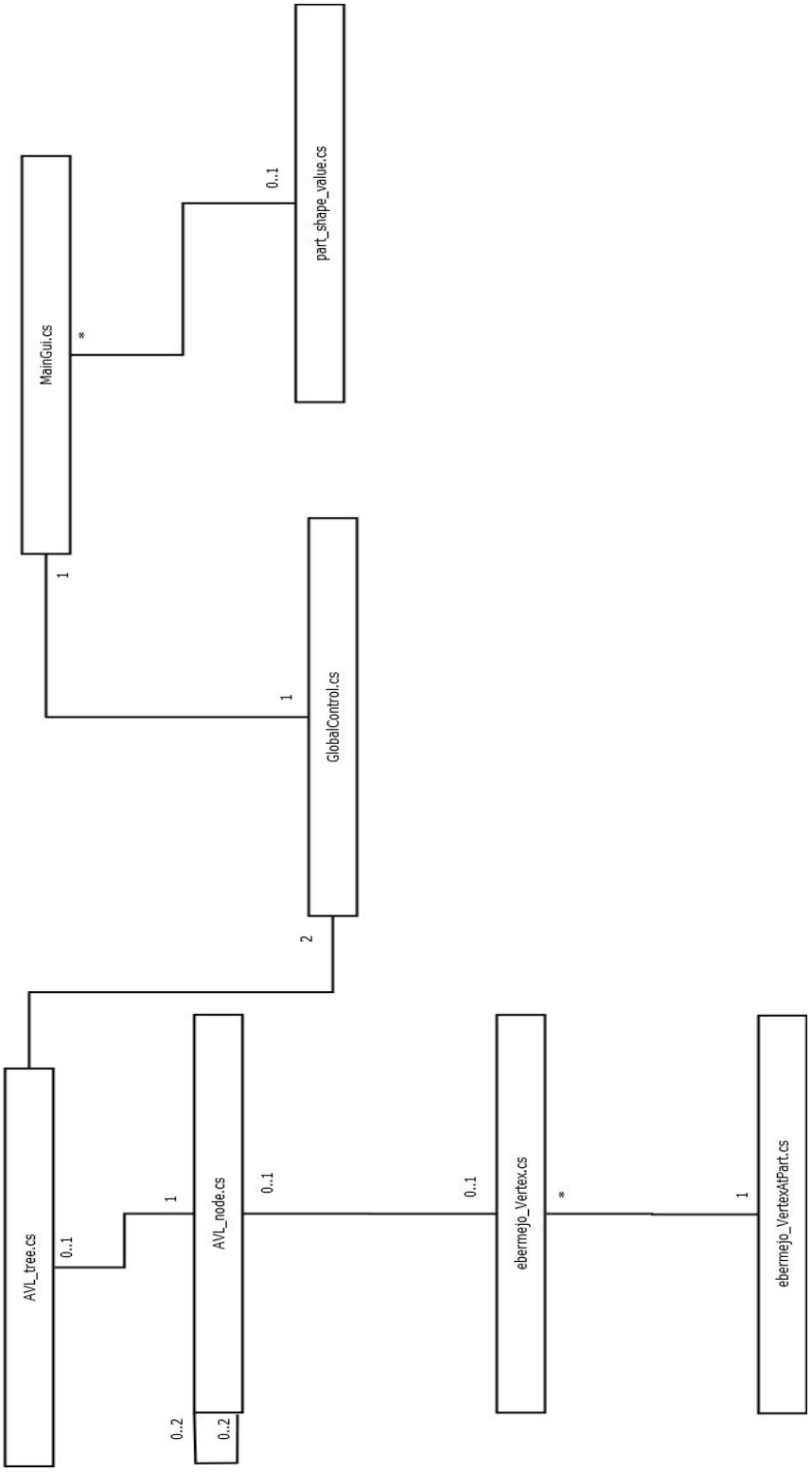


Diagrama UML con las relaciones entre clases. Creado con [15].

Árbol AVL.

Al elegir una estructura de datos principal (almacén de datos) nos surgían serias dudas sobre dos opciones diferentes, pensábamos en un vector de datos o si por el contrario, un árbol AVL, esto es porque pretendíamos almacenar los datos ordenados por su índice, por lo tanto, un vector hubiese bastado y además, hubiese tenido un acceso a memoria más rápido, por otra parte, un vector no es resistente a fallos, por lo que nos decantamos por un árbol AVL cambiando la complejidad de la búsqueda de $O(1)$ por una algo más costosa: $O(log_2 n)$
Esto parece ir en contra del requerimiento **N2**, pero tras haberlo implementado, podemos asegurar que no tenemos problemas en ese sentido con el árbol AVL.

Como hemos dicho, el árbol AVL pretende almacenar los vértices del modelo, los cuales modificaremos cada uno por separado. Un AVL necesita una forma de ser ordenado, para ello, usaremos el índice del vértice, obtenido de la línea que ocupaban en el fichero .obj del modelo original, de esta manera, el vértice situado en la primera línea del archivo, tendrá como índice el número cero.

En nuestra aplicación, estamos ocupando mucha memoria contínuamente por causa de la cantidad de datos que debemos mantener cargados continuamente, por lo tanto, buscamos un algoritmo que no usase recursividad (llamada a una función dentro de sí misma) para minimizar lo máximo posible la memoria que usamos. **Keith Wood** publicó una interesantísima entrada sobre esta estructura de datos sin recursividad en su página personal[17].

Antes hemos definido dos clases que no hemos llegado a explicar realmente, estas son *"AVL_tree"* y *"AVL_node.cs"*

- **AVL_tree.cs** contiene una referencia al primer AVL_node del árbol y las operaciones básicas para gestionarlo, las cuales son:

- *insertNode(ebermejo_Vertex v)* genera un nodo con clave igual al índice del vértice y con vértice igual al argumento que nos pasan, trás esto lo inserta en el árbol re-ordenándose si es necesario.
- *search(int index)* busca en el árbol con el algoritmo antes explicado usando el argumento index como clave a buscar.

- **AVL_node.cs** contiene referencias a los dos nodos hijos (izquierdo y derecho) además de una referencia al nodo padre del mismo para una fácil navegación por el árbol. Además, contiene un atributo llamado "int *key"* el cual hace referencia al índice por el que se buscará el nodo y un atributo llamado *"ebermejo_Vertex value"* que contiene el vértice que debe de contener el nodo.

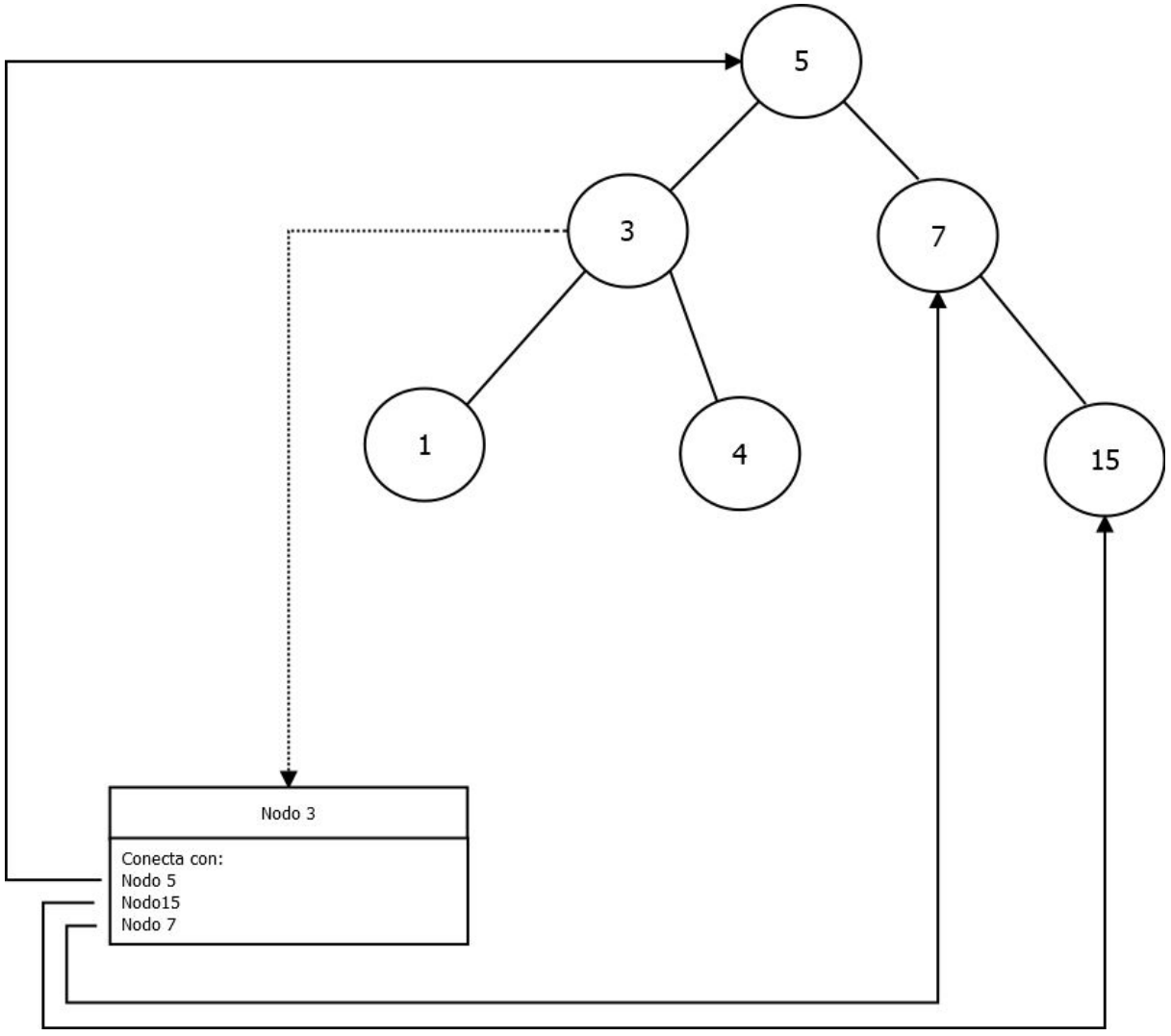
Estructura de datos topológica.

En nuestro caso, necesitábamos además de contener los vértices, tener toda la topología de cada vértice, esto es, todos los vértices con los que se comunica formando una cara del modelo. Con esto en mente, decidimos diseñar una estructura de datos topológica, almacenando en cada vértice referencias (Las referencias apuntan al dato que nos interesa, en lugar de copiarlo, ahorrando así memoria y simplificando los algoritmos) a cada uno de los vértices relativos (llamamos vértice relativo a cualquier vértice que se comunica a otro formando parte con él de alguna de las caras del modelo) a el.

La información de que vértice es relativo a otro la obtenemos en el momento en el que leemos el fichero .obj del modelo ya que al leer las caras que forman triángulos, podemos encontrar los vértices que la conforman y por tanto, rellenar la información topológica en cada uno de ellos.

Esta estructura será imprescindible cuando queramos suavizar las uniones de las partes trás deformarlas, algoritmo del que hablaremos más tarde.

Por lo tanto, al final del proceso de carga de la estructura de datos obtenemos algo como esto, pero con 6900 vértices (El tamaño de nuestros modelos) en lugar de 6 enteros.



Esquema representando la estructura final generada en nuestra aplicación, creada con [15]

Nota: En la figura anterior se representa solamente el nodo '3' en nuestra estructura por razones de legibilidad, pero la misma estructura se repite en cada uno de los nodos.

Diseño del flujo de ejecución de la aplicación.

La aplicación debe de llevar a cabo un proceso de carga bastante costoso, principalmente por causa de los archivos *".shadif"* y dado que es lo más complejo que se ejecuta, deberíamos diseñar cuidadosamente el proceso de inicialización.

En la siguiente figura mostramos un diagrama de secuencias reflejando todo el proceso de carga.

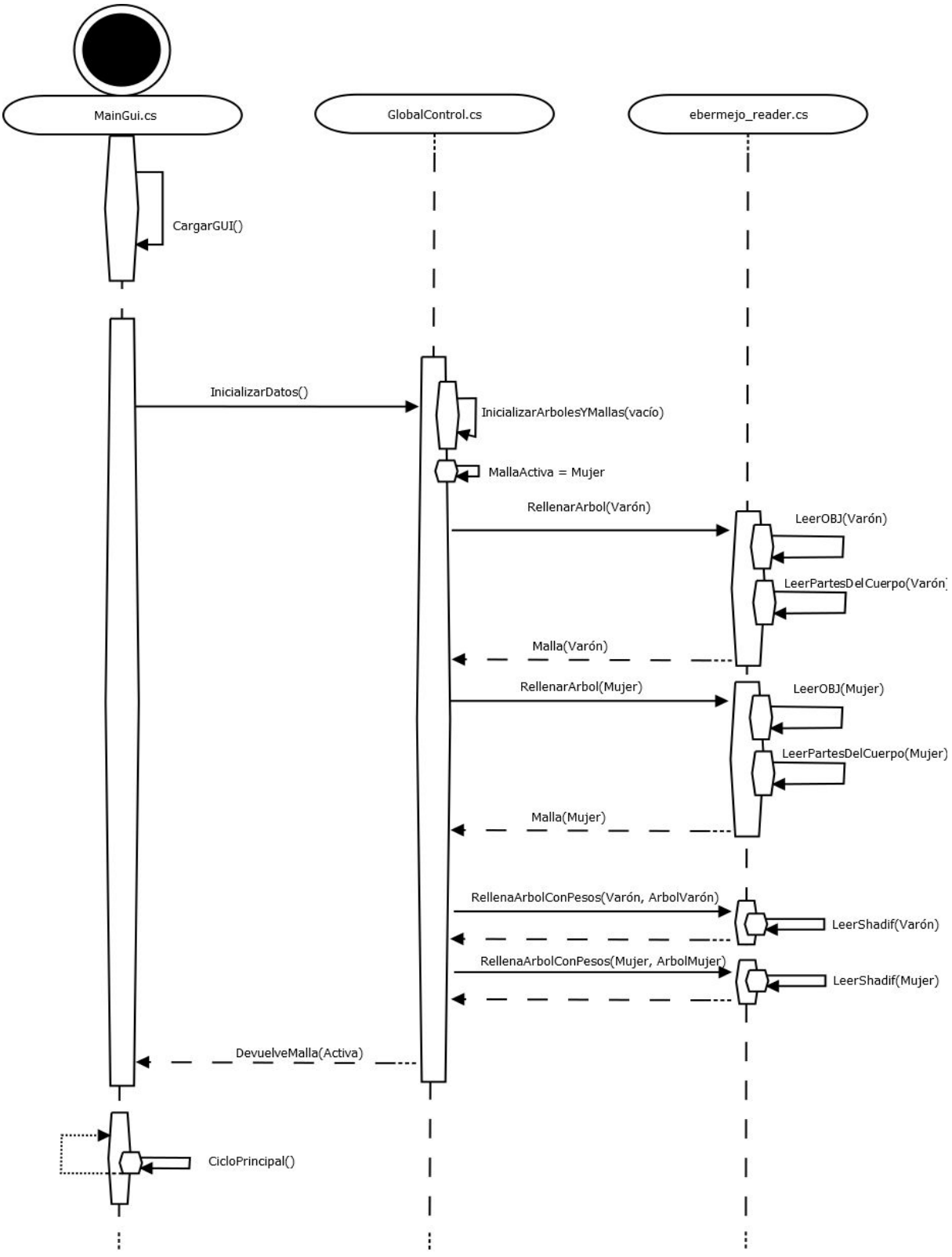


Diagrama de secuencias del proceso de carga de nuestra aplicación. Creado con [15].

En la figura anterior se representan funciones que tal vez son poco legibles, por lo que explicamos su funcionalidad aquí.

- *InicializarArbolesYMallas(vacío)* rellena los árboles masculino y femenino y las cuatro mallas del programa (femenina, masculina, activa y guardad) a null(vacío).
- *RellenarArbolConPesos(género, árbolAsociadoAlGenero)* debe leer los archivos shadif para rellenar los vértices que se encuentran en los árboles con sus correspondientes pesos para cada parte que los modifican.
- *CicloPrincipal()* **No** es un método como tal, si no que representa como le devolvemos el control al hilo de ejecución principal de Unity[9].

Tras este proceso, toda la información necesaria para el correcto funcionamiento de la aplicación queda cargada en memoria en nuestras estructuras de datos. Este proceso puede llevar desde 2 min hasta 10 min, por lo que provoca que en ocasiones, el requerimiento **N1** pueda no quedar cumplido. En el apartado de *"Mejoras"* proponemos un arreglo a este problema.

Flujo de ejecución de una deformación.

Al devolverle en control a Unity[9], el programa empieza y las próximas veces que nos devuelve el control es porqué ocurre un evento en la interfaz de usuario (UI), esto ocurre cuando el usuario pulsa un botón o cambia el valor de algúno de las barras deslizadoras (Slides) que deforma el modelo, el proceso que ocurre al pulsar cada botón lo cubriremos en la sección dedicada a la interfaz de usuario, en cambio, cuando se actualiza un slide y debemos deformar el modelo queda reflejado en siguiente diagrama de secuencia.

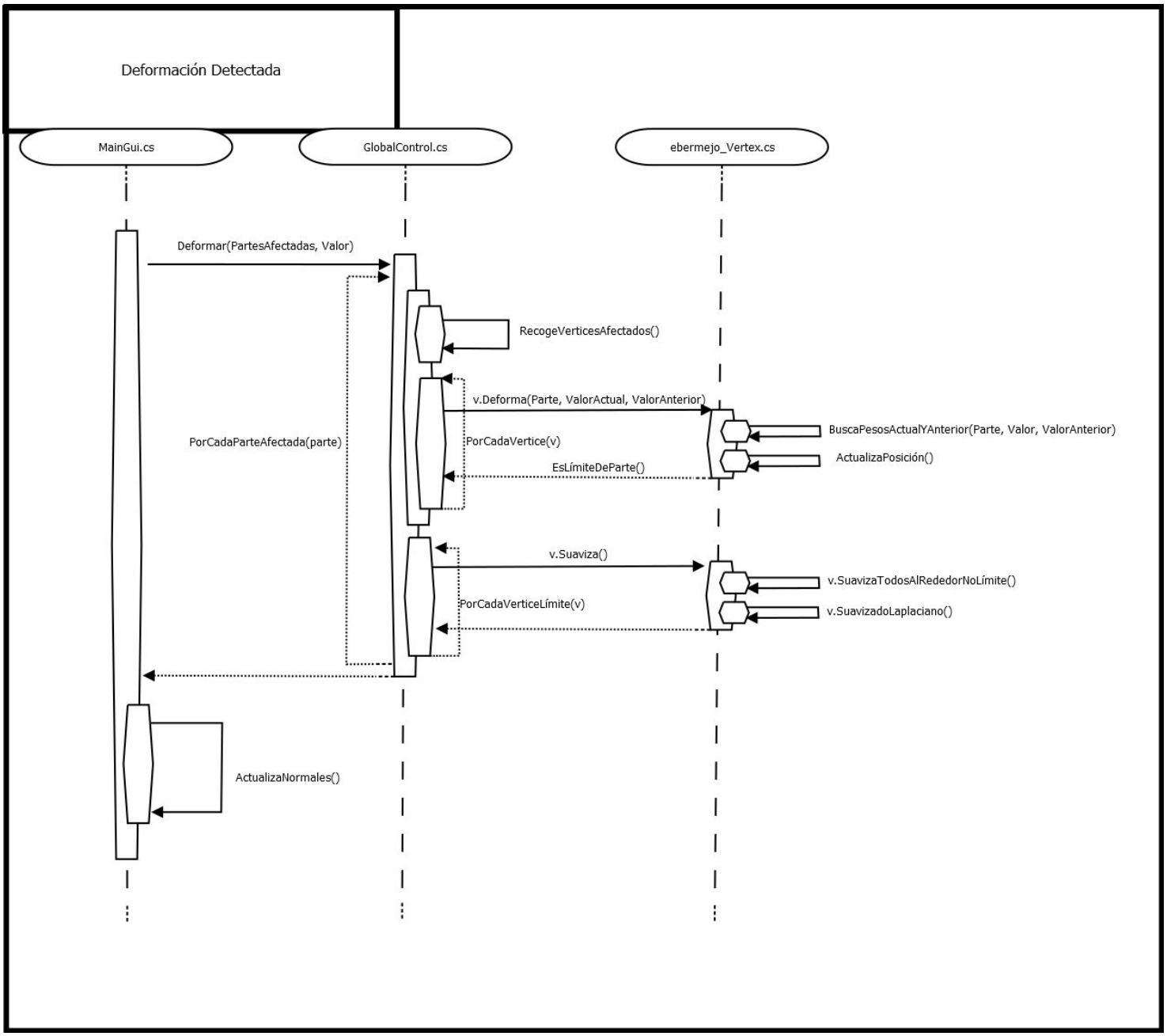


Diagrama de secuencias del proceso de deformación por slide. Creado con [15].

El momento que Unity[9] nos devuelve el control por causa de que un slider se desplaza se ejecuta la función adjunta al susodicho slider, que a su vez, llama a la función que abstrae todas las deformaciones: ***Deformar(slide, valor)*** en esta función, se le cambia el valor del género activo en el momento al slide interno en nuestro código(part_shape_value), en el caso de que el cambio de valor sea exitoso (no ha superado el valor máximo permitido ni ha descendido del mínimo), se habrá cambiado el valor anterior por el actual y el actual por el que le pasamos y además, nos dejará continuar con la deformación hacia la clase *GlobalControl.cs* pasándole como argumento el género al que afecta y el slide al que afecta.
En este momento, repetimos el siguiente proceso por cada parte que se vea afectada, haciendo así deformaciones individuales por partes (como ya hemos dicho, hay deformaciones que pueden afectar a más de una parte simultáneamente). Dentro del ciclo principal, se recogen los vértices que son afectados por la parte que está siendo procesada y a cada uno de ellos, pasándole el valor actual y anterior del slider (actualizados) se llama a la función ***Deformar(parte del cuerpo, valor actual, valor anterior)***

El proceso de deformación dentro del vértice es interesante. Dado que los pesos que cargamos desde los ficheros "*.shadif"* son relativos al modelo estándar y no podemos asegurar que en ese momento el estado de la malla sea ese, deberemos llevar el vértice a la posición original y volverle a aplicar los nuevos pesos, (este es el uso del argumento del valor anterior en los sliders). Por otra parte, para obtener los pesos del valor anterior y actual, deberemos interpolar los valores de los pesos anterior y siguientes al valor anterior y actual de modo que:

$$(ValorActual + Slider.valorMínimo \vee * 10)\ \ eAnteriorActual \leftarrow Floor\ indic$$
$$indicePosteriorActual \leftarrow indiceAnteriorActual + 1 \blacksquare$$

$$(ValorAnterior + Slider.valorMínimo \vee * 10)\ \ eAnteriorAnterior \leftarrow Floor\ indic$$
$$indicePosteriorAnterio \leftarrow indiceAnteriorAnterior + 1 \blacksquare$$

Método de obtención de los índices de los pesos en los que estamos interesados.

Siendo la función *Floor()* una función que redondea el valor entre paréntesis al entero anterior al valor.

p.e: Floor(5.7) = 5

Teniendo los índices de los pesos en la lista, tendremos que utilizar una interpolación lineal para encontrar exáctamente el valor que se le aplicó en la última iteración a ese vértice.

$$W\ \ indiceAnterior * \ indicePosterior *- W\ \ \alpha\leftarrow$$
$$indicePosterior * (ValorAnterior * 10) - \frac{indiceAnterior*}{}\ \beta\leftarrow$$
$$indiceAnterior * \ P_a\leftarrow(\alpha\times\beta) + W$$

Fórmula usada para realizar la interpolación entre los pesos

Teniendo como W_γ el valor de los pesos en el índice γ, * como el peso interpolado que estemos calculando $(\frac{Actual}{Anterior})$ y P_a como los pesos finales interpolados.

Tras realizar la interpolación, sólo nos queda dividir la posición actual del vértice por los pesos anteriores y multiplicarla después por los actuales

$$v.Pos\leftarrow\frac{v.Pos}{P_{anterior}}$$
$$v.Pos\leftarrow v.Pos\times P_{actual}$$

Operación final para obtener la nueva posición del vértice.

De esta manera, no el resto de transformaciones llevadas a cabo con anterioridad a la hora de calcular la posición nueva del vértice, ya que podríamos simplemente recuperar la posición que tiene en el modelo estándar y aplicar los pesos correspondientes.

Trás el proceso de deformación de los vértices sólamente devolvemos si el deformado formaba parte del límite del modelo (para después suavizar, de lo que hablaremos en el siguiente punto), y actualizamos la posición del vértice en la malla activa con los valores calculados.

De esta manera, parte por parte, y vértice por vértice calculamos todas las deformaciones llevadas a cabo al cambiar el valor de un slider.

Suavizado después de una deformación.

Tal y como se dice en [1] trás una deformación, es crucial un proceso de suavizado de la malla, dado que de otro modo, se producen artefactos escalonados.

Imagen de una versión temprana de nuestra aplicación donde el suavizado no estaba implementado.

Durante gran parte del desarrollo de la aplicación el algoritmo de suavizado fué diferente del actual, por lo que consideramos relevante agregar el anterior a esta memoria.

Ambas técnicas se usaban técnicas de suavizado laplaciano[18] de esta manera

3 Laplacian Smoothing

Laplacian Smoothing is a very simple PDE-based smoothing approach formulated as follows [8]:

$$v_i \leftarrow v_i + \sum_{v_j \in v_i^*} \left(\frac{v_j - v_i}{d_i} \right) \tag{4}$$

This process can be done repeatedly to correct the location of each vertex to the geometric center of its neighboring vertices. This approach is simple and fast, however, it produces an oversmoothing result after few iterations.

Note that the *Laplacian Smoothing* is just a special case of the *Weighted Laplacian Filter* [11], also called *Local Averaging*:

$$v_i \leftarrow v_i + \frac{1}{\sum_{v_j \in v_i^*} w_{ij}} \sum_{v_j \in v_i^*} w_{ij}(v_j - v_i) \tag{5}$$

Where w_{ij} are the weights defined as :

$$w_{ij} = \begin{cases} > 0 & if\ v_j \in v_i^* \\ 0 & otherwise \end{cases}$$

For $w_{ij} = 1$ if $v_j \in v_i^*$ we retrieve the *Laplacian Smoothing* update rule, $\sum_{v_j \in v_i^*} w_{ij} = d_i$.

Extracto de [18] página 4(79), donde se habla del suavizado laplaciano y de su predecesor "Weighted Laplacian Filter"

Con la técnica del suavizado laplaciano[18] tratamos de hallar el centro geométrico de los vecinos del vértice siendo suavizado, produciendo muy buenos resultados para nuestro caso, en los que el "ruido" producido por las deformaciones es realmente alto.

En la primera implementación del suavizado en nuestra aplicación, los cambios generados por el suavizado no se aplicaban a los datos del árbol, si no que se aplicaban solamente a la malla activa en el momento, por lo que generaba problemas ya que los datos debían de ser calculados antes de que hubiese acabado la deformación del modelo dado que de otra manera, perderíamos la información de suavizado generada, esto provocaba problemas graves en las juntas de las partes del cuerpo como los siguientes.

Capturas de pantalla de la aplicación antes del arreglo de suavizado donde se advierte una deformación muy clara en los hombros del modelo producto del suavizado mal planteado.

De manera que decidimos probar qué pasaría si el suavizado lo hiciéramos después de toda la deformación del modelo pero que la información modificada persistiera en el tiempo, aunque esto significase que las nuevas posiciones podrían interferir en los pesos originales obtenidos desde SMPL[7]. Por lo que cambiamos el algoritmo para que en lugar de devolver una lista con todos los vértices modificados, devolviese simplemente una variable booleana(con valores true / false) indicando si el vértice es borde de la parte modificada o no, de tal manera que sólo estos como sus vértices vecinos son suavizados después de la deformación.

Los cambios fueron evidentes y mejoraron el resultado final, por lo que este último algoritmo fue el definitivo en nuestra aplicación.

Capturas de pantalla de la aplicación final con el nuevo algoritmo de deformación.

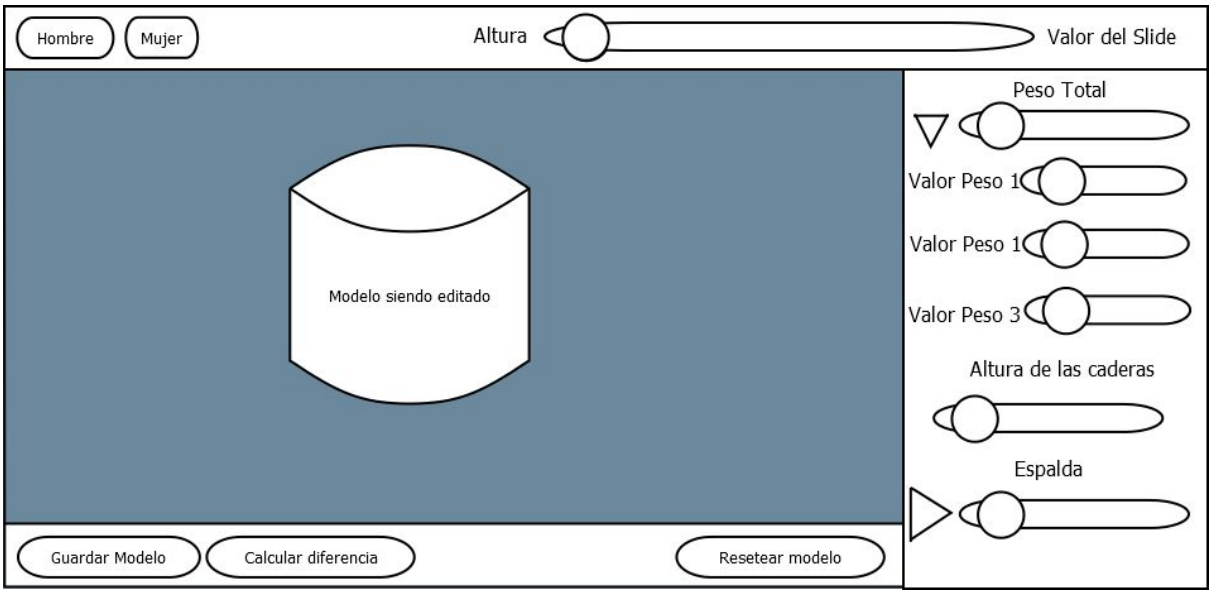
Aunque, por otro lado, podemos observar cómo se generan bordes más marcados de esta manera.

Diseño de la Interfaz de Usuario.

El diseño de la interfaz de usuario de nuestra aplicación es un proceso crítico, esto es debido a que la aplicación está dirigida al público general, y estos no tienen porqué tener conocimientos avanzados de informática, por lo tanto, es un factor clave en el cumplimiento del requisito **N13.**

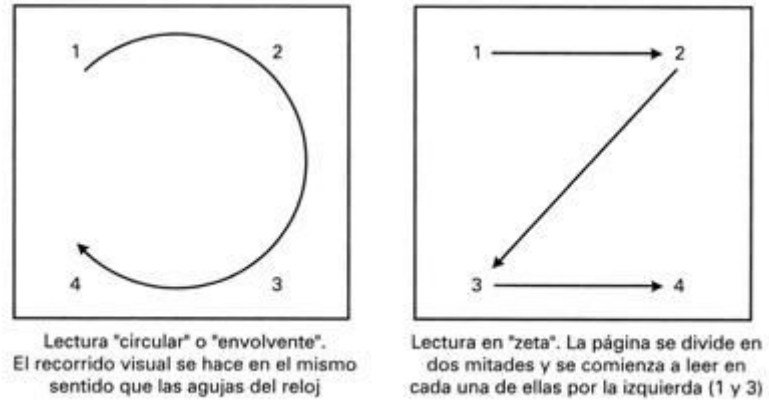
Como primera decisión importante, decidimos que la aplicación debería seguir un modelo de "single-page" lo que significa que todo el trabajo debe realizarse sin un cambio de la interfaz.

La interfaz de usuario (UI) de nuestra aplicación, debe de ser lo más intuitiva posible, a la par de que debe estar organizada de la mejor manera posible. En la siguiente figura se muestran unos prototipos de interfaz dibujados mediante el uso de un programa generador de diagramas[15], el cual nos sirvió de guía para generar nuestra UI.



Prototipo de interfaz (En un primer momento estaba dibujado en papel, aunque lo hemos digitalizado para esta memoria)

Este prototipo (que fue finalmente implementado) está basado en estudios sobre el recorrido visual[19], estos informes, señalan que la lectura que realizamos normalmente al recibir información visual se ordena, usualmente de una de las siguientes formas:



Extracto de [19] donde se muestran ambas formas de lectura.

El prototipo anterior está diseñado basándonos en ambos métodos de lectura, para así adaptar el rápido aprendizaje de la aplicación al mayor público posible de manera que:

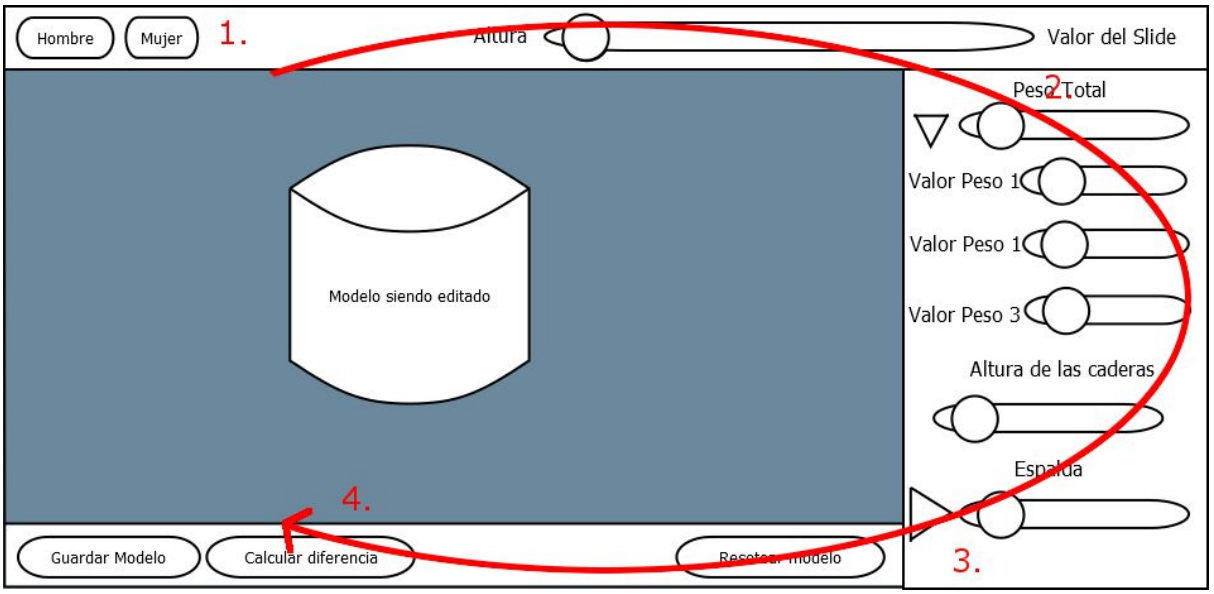
Lectura circular de la aplicación.

"Una teoría sostiene que la lectura de la primera página es circular, comenzado en el ángulo superior izquierdo y siguiendo el sentido de las agujas del reloj. Por ello, la noticia principal se coloca en el ángulo superior izquierdo, en lo que se denomina región o área óptica primaria."

Fragmento de[19] en el que nos explican la teoría de lectura circular

En el caso de que el usuario recorra visualmente la aplicación de manera circular, lo primero que observará será la elección de género y el slider de altura, lo que decidimos que fueran los primeros parámetros en editarse, ya que son los parámetros más estáticos para cada persona, seguido del panel de lateral, el cual es la principal herramienta de edición de los modelos y donde más tiempo pasará el usuario, y como último, la barra inferior, que además, se sitúa a la izquierda de del panel lateral, los elementos de la cual, son los últimos en ser utilizados en cada edición del modelado (Salvar malla, calcular diferencia y reiniciar la malla)

Toda esta lectura, circuncida la parte principal de la aplicación, la ventana donde se muestra el modelo que se está actualmente editando.



Esquema donde mostramos el recorrido teórico que sigue un lector al usar nuestra aplicación.

Lectura en "zeta" de nuestra aplicación.

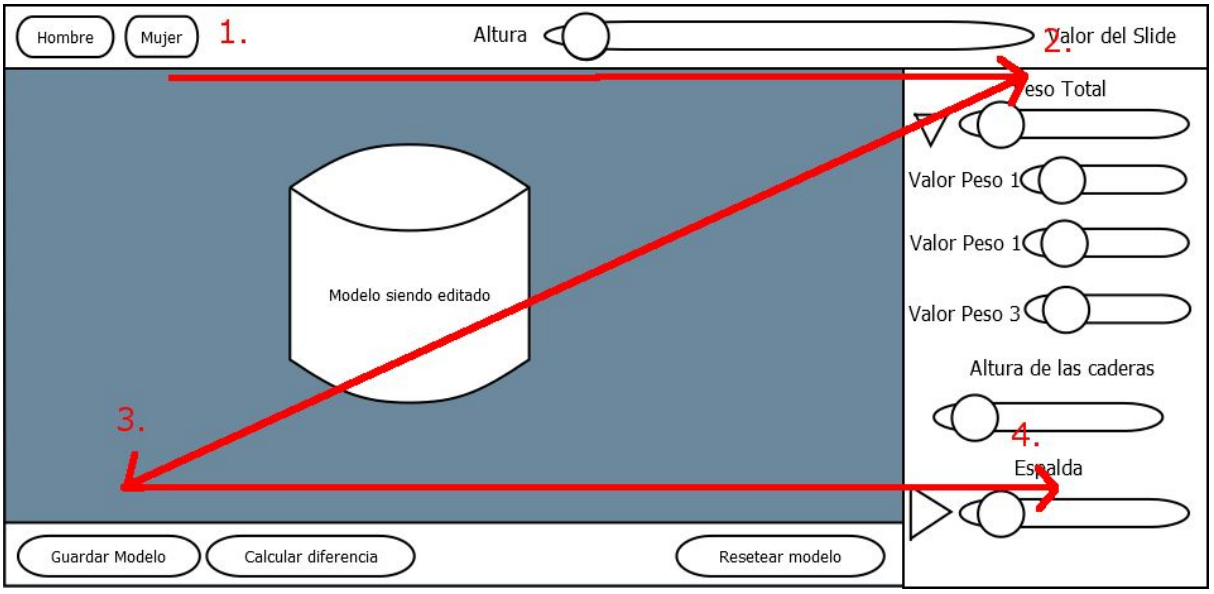
Una segunda teoría divide la página en dos mitades horizontales (superior e inferior) y dos mitades verticales (izquierda y derecha). Considera que la parte superior tiene mayor valor que la inferior y que la izquierda tiene más valor que la derecha. Queda, pues, dividida la plana en cuadrantes, de los cuales el más importante será el superior izquierdo y el menos, el inferior derecho. En consecuencia, según esta teoría la vista hace una lectura siguiendo el trazado de la letra zeta (Z).

Fragmento de[19] en el que nos explican la teoría de lectura circular

Dado que no se ha demostrado ninguna teoría de lectura por encima de la otra, debíamos diseñar para ambos casos para de esta manera todo el mundo tuviese una buena experiencia de aprendizaje en nuestra aplicación.

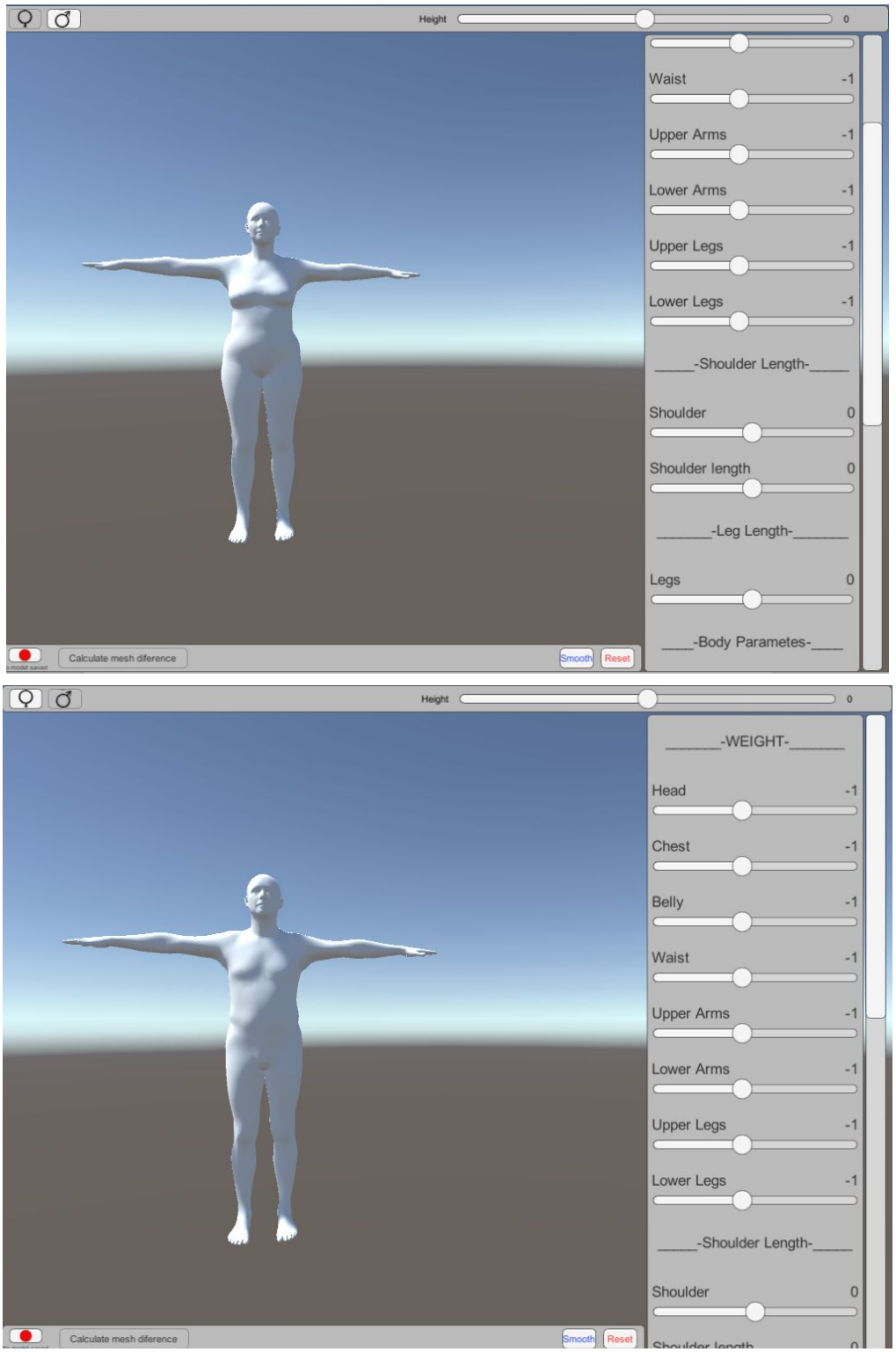
De esta manera, el usuario continuará observando los botones de cambio de género y el slider de cambio de altura antes que cualquier otra cosa, indicando así el orden de uso. Tras esto, la vista

recorrerá de arriba a abajo y de izquierda a derecha la pantalla, observando así el área de trabajo, como el área de sliders que deforman el modelo, como en el modelo circular, los valores que deben de ser los últimos en ser utilizados son los últimos en ser leídos, obteniendo así una UI de fácil aprendizaje y lectura para todos.



Esquema donde mostramos el recorrido teórico que sigue un lector al usar nuestra aplicación.

Cambios del prototipo a la interfaz final.



Capturas de pantalla de nuestra aplicación final junto con la UI terminada.

El primer cambio más notable que advertimos entre el prototipo de UI y la interfaz final es el cambio de idioma. Usamos el inglés dado que es el lenguaje más extendido y que más gente puede entender. Podríamos haber usado técnicas de internacionalización, pero por falta de tiempo no pudimos. Por otra parte, en lugar de un solo botón en la barra inferior en la parte derecha como teníamos en el prototipo, encontramos dos, estos botones son el que teníamos antes de reiniciar y un botón donde se lee "smooth" (suavizar en inglés) que aplica el suavizado laplaciano[18] que hemos explicado anteriormente en el modelo al completo para usarse cuando el modelo, al haber sido deformado muchas veces, tenga partes o artefactos con formas indeseadas, poder suavizarlo y obtener un modelo realista como es nuestro objetivo en el requerimiento **F1.** La última diferencia apreciable es como en la barra lateral del prototipo se sitúan desplegables según el tipo de deformación que se establece, estos no aparecen en la interfaz final, esto no es debido a una decisión de diseño, si no a que Unity no los permite fácilmente en una lista con scroll, en el apartado "*Mejoras*" se habla de esto, en su lugar, hemos colocado "títulos" que indican qué tipo de deformaciones son las que sitúan debajo del mismo.

Explicación de la interfaz Gráfica

- Barra superior
    - Botones de cambio de género. El botón correspondiente al género del modelo activo, está siempre deshabilitado, por lo que sólo podemos clicar en el género contrario.
    - Slider "Height" Modifica la altura del modelo, se sitúa en la barra superior ya que es uno de los atributos más importantes y que menos debe de ser modificado.
- Barra lateral, se reúnen todos los slides de deformación excepto el de altura.
    - "Weight" Deformaciones sobre el peso del modelo.
    - "Shoulder Length" Deformaciones sobre la anchura de la espalda.

  - "Leg Length" Deformaciones sobre la altura de las piernas
  - "Body Parameters" Deformaciones de varios tipos sobre el tronco del modelo.
- Barra Inferior
  - Botón con círculo rojo, Salva la mall actual para calcular la diferencia entre este y otro más tarde.
  - Botón con "Calculate Mesh Difference" Calcula la diferencia entre la malla guardada y la actual, el botón está desactivado si no se ha pulsado antes el botón con círculo rojo. Como lo hace se cubre en el siguiente punto.
  - Botón "Smooth" aplica el suavizado Laplaciano[18] a todo el modelo.
  - Botón "Reset" Reinicia el modelo actual y los sliders a los valores por defecto.

Cálculo de la diferencia entre dos modelos.

Para calcular la diferencia entre los modelos generados por el usuario, (el guardado y el actual) usamos un algoritmo realmente básico y sencillo.

Recorremos todos los vértices de ambos modelos simultáneamente, acumulando la diferencia entre ambos en una variable, el resultado obtenido, lo redondeamos al valor entero más bajo.

$$p.e\ Redondeo(5.8) = 5$$

La fórmula que usamos para llevar a cabo esto es:

$$(v_a[i] - v_s[i]) \sum_{i=0}^{\gamma} \Theta \leftarrow Floor$$

Siendo Floor() la función de redondeo antes explicada, γ es el número total de vértices de nuestro modelo, v_a y v_s son, respectivamente los arrays de vértices del modelo actual y guardado.

Revisión de los requerimientos de la aplicación.

Cuando "terminamos" de desarrollar la aplicación es el momento de revisar si cumplimos o no los requerimientos de la aplicación que definimos previa implementación y diseño de la misma, para ello usaremos los siguientes símbolos y en ocasiones una breve explicación de porqué está o no cumplido el requerimiento.

- ✔ Requerimiento cumplido satisfactoriamente.
- ✖ Requerimiento no cumplido.
- ❓ Requerimiento cumplido / no cumplido, pero con matices.

Funcionales.

1. ❓ *La aplicación debe de ser capaz de modificar un modelo humano 3D por partes de manera lo suficientemente realista como para poder identificarse con el resultado.*
   Explicación: En condiciones normales, este requerimiento estaría cumplido, pero en ocasiones, se forman artefactos extraños en el modelo que deben de ser suavizados con el botón "Smooth", se habla más del tema en el apartado "*Mejoras*" ❓

No funcionales.

1. ❓ *La aplicación debe de tener un tiempo de carga inferior a 5 minutos.* **Explicación:** En algunos casos, según la potencia del procesador del ordenador en el que se ejecute la aplicación, el tiempo de carga puede ir desde los 2 minutos hasta los 10 minutos. ❓
2. ✔ *La ejecución de la aplicación debe de ser ininterrumpida y fluida.* ✔
3. ✔ *Las deformaciones realizadas deben de de ser totalmente fluidas (Obtener una velocidad mínima de refresco de pantalla durante la deformación de 30 imágenes por segundo)* ✔

4. ✔ *La aplicación debe de poder tener una rápida distribución* **Explicación:** Gracias al potente mecanismo de despliegue que nos ofrece Unity esto es posible sin ningún problema. ✔
5. ✔ *La aplicación debe de ser operable sin acceso a internet.* ✔
6. ✔ *Trás el trabajo realizado no debe de quedar rastro de él/los modelos generados para asegurar que esta quede privada entre paciente - doctor.* ✔
7. ✔ *La aplicación debe de ser capaz de guardar un modelo en memoria para, sin modificar este, empezar a modificar un segundo.*✔
8. ✔ *La aplicación debe de permitir calcular la distancia entre un modelo anterior y el que está siendo modificado actualmente.* ✔
9. ✔ *La aplicación debe de permitir el cambio entre sexos.* ✔
10. ✔ *La aplicación debe de tener una amplia gama de deformaciones a las distintas partes del cuerpo.* ✔
11. ✔ *La aplicación debe de contener un botón de reinicio en el caso de que el usuario no está de acuerdo con el resultado producido.* ✔
12. *La aplicación debe de cumplir con la legislación española sobre pruebas de software con humanos.*
13. ✔ *La aplicación debe de ser fácilmente puesta a prueba y usada por individuos sin conocimientos informáticos.* **Explicación:** En ninguna de las diversas pruebas con terceros, alguno de estos ha tenido problema al usarla además de alguna barrera idiomática en algunos de ellos. ✔

Manual de Usuario.

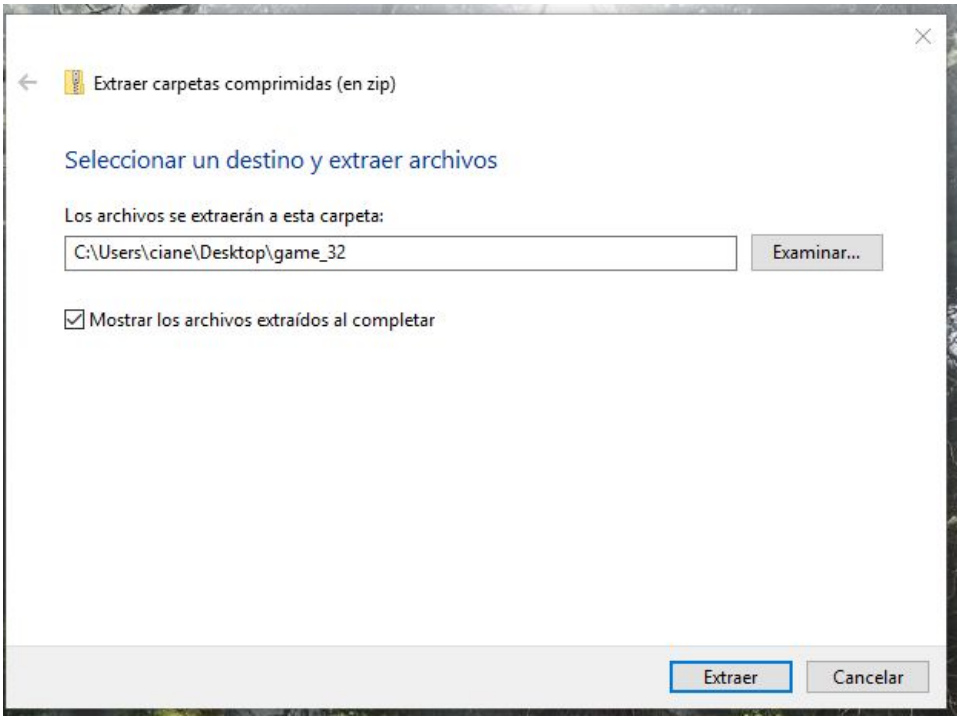
El uso de esta aplicación es realmente simple, dado que es una aplicación destinada al público general, esta debe de ser fácilmente usable para además cumplir con el requerimiento **N13**.

Instalación del software.

La instalación del software es realmente sencilla, para proceder a esta, es necesario descargar la versión compatible con la arquitectura del ordenador donde se va a ejecutar.

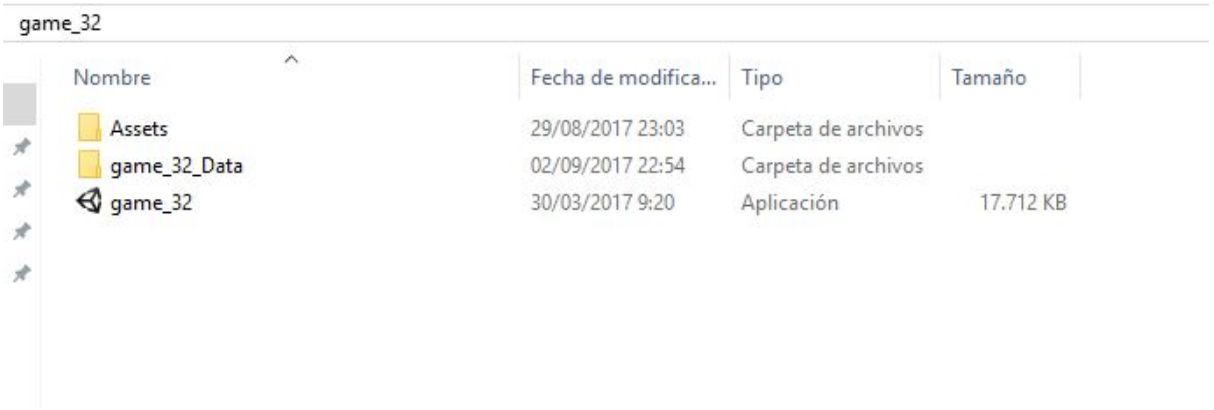
- Versión 64bit.
- Versión 32bits.

Al descargar el software sólo tenemos que descomprimir la carpeta comprimida descargada.



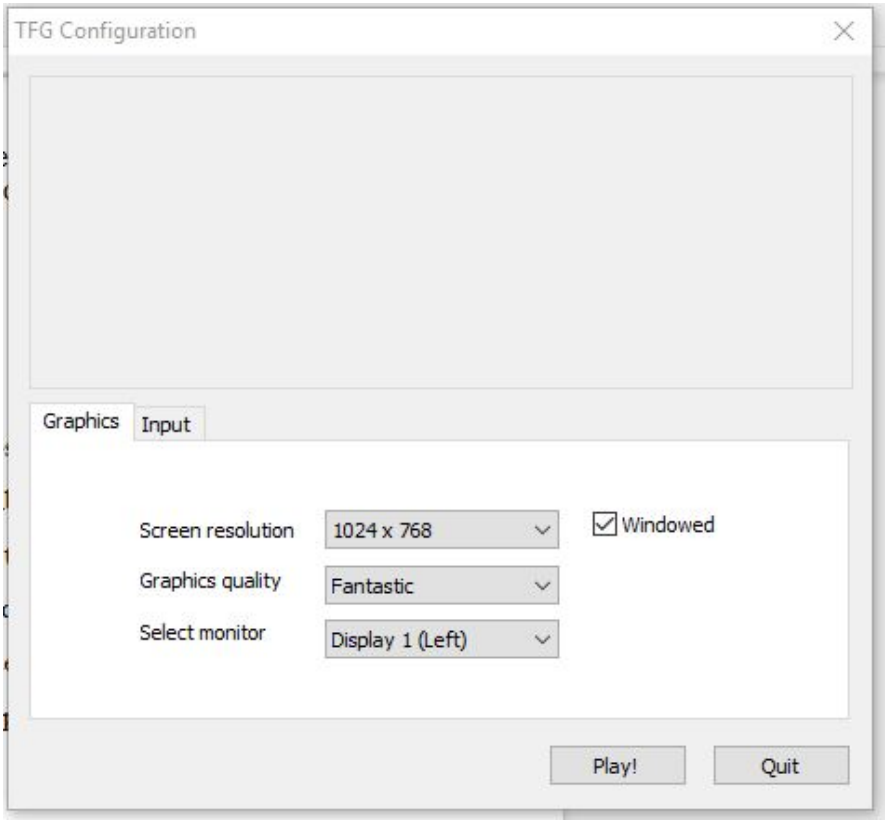
Tras la descompresión, obtendremos una carpeta, con todos los datos necesarios para la ejecución del mismo. Esta carpeta contiene un archivo ejecutable .exe, haciendo doble click sobre él se abre la aplicación. En la carpeta Data se encuentra todos los componentes necesarios para la ejecución

de la base del software, mientras que en la carpeta de assets se encuentran todos los archivos necesarios para la carga de modelos y demás, cualquiera de estos datos es crítico, por lo que no se debería borrar ninguno para que la aplicación funcione correctamente.



Ejecución de la aplicación.

Desde el momento en el que el software queda descomprimido, es posible ejecutar la aplicación, para ello, sólo es necesario hacer doble clic sobre el archivo ejecutable .exe, en ese momento aparece la siguiente pantalla que permite una mínima configuración inicial.



En este punto, recomendamos ejecutar la aplicación con la casilla de “Windowed” marcada, y una resolución inferior a la de la pantalla en la que estamos ejecutándola, tras esto, con clicar en “Play!” el software empezará a ejecutarse.

La ventana de la aplicación deberá teñirse de verde tras la presentación de Unity[9], tiempo en el que la aplicación estará cargando los datos de la misma, este proceso puede llevar hasta 10 minutos y procesadores lentos.

Cuando finalice el proceso de carga, la aplicación estará lista para trabajar, presentándose la siguiente interfaz:

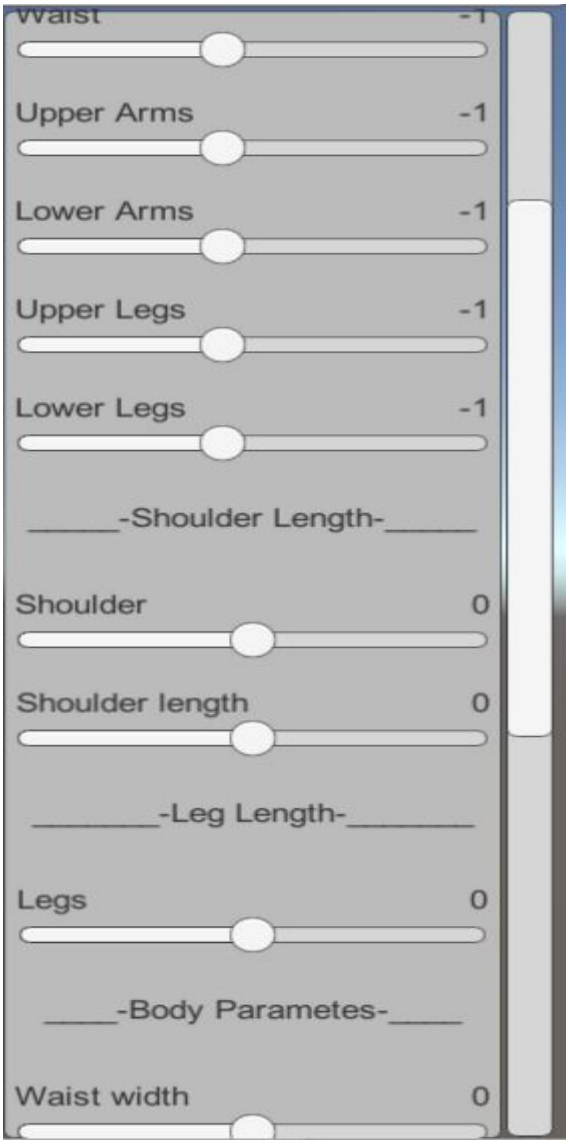
Uso de la aplicación

El uso de la aplicación es realmente sencillo. Lo primero que debemos hacer es seleccionar el sexo del modelo que deformaremos, esto se lleva a cabo con los botones en la parte superior de la pantalla.

Estos botones presentados con los símbolos masculinos y femeninos controlan el género del modelo, de manera que el botón correspondiente al modelo activo está desactivado y sólo se puede usar el botón del género contrario para cambiar a el.

Después de la elección de género, deberíamos elegir la estatura que queremos, esto lo realizamos mediante la barra en la parte superior de la pantalla “Height”

Tras la elección de género y la altura que deseamos, debemos usar el panel lateral para modelar la malla central a nuestro gusto.



En este panel disponemos de multitud de maneras de deformar el modelo.

Los diferentes slides están distribuidos en cuatro grupos principales:

- "Weight" agrupa todas las deformaciones relacionadas con el peso de la persona representada por el modelo.
- "Shoulder Length" agrupa las modificaciones sobre la espalda
- "Leg Length" modifica la longitud de las piernas del modelo.
- "Body Parameters" Agrupa diferentes deformaciones que afectan al torso del modelo

En la siguiente lista expresamos todas las posibles deformaciones de nuestro modelo y a que y cómo afecta cada una.

- Height - Altura del modelo.
- **__WEIGHT__**
- Head - Papada.
- Chest - Grasa en el pecho.
- Belly - Grasa en el vientre.

- Waist - Grasa en la cader.
- Upper Arms - Grosor de la parte superior de los brazos.
- Lower Arms - Grosor de la parte inferior de los brazos.
- Upper Legs - Grosor de los muslos.
- Lower Legs - Grosor de las pantorrillas.
- **__SHOULDER LENGTH__**
- Shoulder - Longitud del torso
- Shoulder Length - Anchura de espalda
- **__LEG LENGTH__**
- Legs - Longitud de piernas
- **__BODY PARAMETERS__**
- Waist Width - Anchura de la cintura
- Chest Height - Altura del pecho
- Neck Height - Largura del cuello.
- Chest Fall - Caída del pecho.
- Waist Fall - Caída de la cadera.
- Chest Front - Forma del pecho
- Chest projection - Proyección del pecho hacia delante.

Nota al usuario: Para evitar problemas, desplace los sliders, en lugar de clicar en la posición donde quiera colocarlos, de otras maneras se pueden generar formas indeseables.

Después de terminar de editar el modelo es muy posible que se hayan generado formas poco deseables, por lo que antes de continuar, recomendamos pulsar el botón “Smooth” en la barra inferior de la aplicación.

Esto suavizará todo el modelo generando resultados más realistas.

Si llega el momento en el que no nos guste una malla deformada y no podamos volver a un estado que nos guste, deberemos pulsar el botón de reinicio “Reset” situado en la barra inferior de la aplicación, que reiniciará el modelo a su estado inicial. Esta operación **NO** modificará la malla guardada de ninguna manera, si es que hay alguna

Cuando queramos guardar el modelo, deberemos usar el botón con un círculo rojo, al pulsarlo, un texto por debajo del mismo cambiará a indicarnos que se ha guardado correctamente y que hay una malla en la memoria.

Cuando creamos que estamos listos para calcular la diferencia entre el modelo guardado y el actual, deberemos pulsar el botón que se sitúa al lado del anterior, este botón estará inhabilitado si no hay ninguna malla guardada.

Al pulsar este botón, el resultado de la operación aparecerá al lado del mismo con número entero de puntos de diferencia.

Por último, para cerrar el programa, deberemos pulsar la X en la ventana de Windows.

Como última guía para el uso de esta aplicación, debemos advertir, de que puede modificar el ángulo de visión con el que vé el modelo que está deformando, usando:

- La tecla ‘R’: Rota hacia la izquierda.

- La tecla ‘W’: Inclina la cámara hacia arriba.
- La tecla ‘S’: Inclina la cámara hacia abajo.
-

Legislación relativa a este software.

Dado que la legislación europea que nos concierne no está publicada, este apartado tratará sobre la legislación a nivel del estado español.

En el **REAL DECRETO 1591/2009, DE 16 DE OCTUBRE, POR EL QUE SE REGULAN LOS PRODUCTOS SANITARIOS** [20] se habla de producto sanitario como:

«Producto sanitario»: cualquier instrumento, dispositivo, equipo, programa informático, material u otro artículo, utilizado solo o en combinación, incluidos los programas informáticos destinados por su fabricante a finalidades específicas de diagnóstico y/o terapia y que intervengan en su buen funcionamiento, destinado por el fabricante a ser utilizado en seres humanos con fines de:
1º Diagnóstico, prevención, control, tratamiento o alivio de una enfermedad,
2º diagnóstico, control, tratamiento, alivio o compensación de una lesión o de una deficiencia,
3º investigación, sustitución o modificación de la anatomía o de un proceso fisiológico,
4º regulación de la concepción, y que no ejerza la acción principal que se desee obtener en el interior o en la superficie del cuerpo humano por medios farmacológicos, inmunológicos, ni metabólicos, pero a cuya función puedan contribuir tales medios.
Artículo 2. Definiciones.

Destacamos las líneas que hablan de programa informático con fines de Diagnóstico y/o prevención:

“«Producto sanitario»: cualquier instrumento, dispositivo, equipo, programa informático...”
“a ser utilizado en seres humanos con fines de:”

" 1º Diagnóstico, prevención, control, tratamiento o alivio de una enfermedad,"

Gracias a estas líneas tan precisas, podemos decir que nuestro software entra dentro de esta regulación[20].

Podemos asegurar que se cumplen las *Garantías sanitarias de los productos* establecidas en el **artículo 4** de (el decreto ley)[20] dado que:

- Al ser un programa informático se cumplen los requisitos esenciales del **artículo 5** de [20]
- Dado que este software es libre, toda la información relativa a él es pública y de fácil acceso.
- A este proyecto no se le atribuyen funciones que no pueda realizar, induciendo así a error.

En el **artículo 7** de [20] se habla sobre la aseguración de confidencialidad para todas las partes que tomen acción en la aplicación. Dado que esta aplicación comple los **requerimientos NF5 y NF6**, por los que se restringe el acceso a internet de la aplicación, así como el guardar cualquier tipo de dato producto de la sesión de uso de la misma, podemos decir que la confidencialidad está asegurada en nuestra aplicación.

Dado que por ahora, no podemos disponer de un equipo que soporte la aplicación, ni una infraestructura donde trabajar con ella, ninguna de las cláusulas del **artículo 10** de [20] se cumplen, las cuales aseguran un soporte continuado de la aplicación, lo que provoca que por el momento nos sea imposible conseguir una licencia previa de funcionamiento.

Según el **anexo IX** de [20] nuestro producto se clasificará como de clase I, ya que es un producto de uso pasajero. Esto significa, que tendremos restricciones mínimas para conseguir la aprobación de distribución y el sello CE (comunidad europea).

"Uso pasajero: Destinados normalmente a utilizarse de forma continua durante menos de sesenta minutos."

Propiedad Intelectual

Según [21] donde se expresa resumidamente qué y cuáles son los derechos de la propiedad intelectual:

"La propiedad intelectual es el conjunto de derechos que corresponden a los autores y a otros titulares (artistas, productores, organismos de radiodifusión...) respecto de las obras y prestaciones fruto de su creación."

Esto influye directamente sobre nuestro trabajo, dado que hemos usado la propiedad intelectual de un tercero para llevarlo a cabo, hablamos de SMPL[7], y para asegurarnos de que nos dan su consentimiento para el uso de su software debemos revisar la licencia que publican en su página web [7]

"Max-Planck grants you a non-exclusive, non-transferable, free of charge right:

- *To download the Model and use it on computers owned, leased or otherwise controlled by you and/or your organisation;*
- *To use the Model for the sole purpose of performing non-commercial scientific research, non-commercial education, or non-commercial artistic projects.*

Any other use, in particular any use for commercial purposes, is prohibited."

"Max-Planck le concede una licencia libre de cargo intransferible y no exclusiva:

- ***Para descargar el modelo y usarlo en ordenadores en propiedad, prestados o de cualquier otra manera controlado por usted y/o su organización***
- ***Para usar el modelo para el sólo propósito de desarrollar investigación científica, educación no comercial, o proyectos artísticos no comerciales. Cualquier otro uso, en particular usos comerciales, está prohibido."***

Esto nos indica que mientras no comercializeamos nuestro software, se le podrá considerar como investigación científica y, por tanto, libre de todo cargo.

Evaluación (legislativa) del software.

Existe un procedimiento para la evaluación de cualquier producto sanitario especificado en el **Anexo X** de [20] donde las pruebas que llevemos a cabo sobre este software deben basarse. Estos son los apartados que nos afectan de alguna manera.

"1.1 ter. La evaluación clínica y sus resultados deberán documentarse. Esta documentación y/o sus referencias completas se incluirán en la documentación técnica del producto."

"1.2. Todos los datos deberán considerarse confidenciales con arreglo a lo dispuesto en el artículo 7."

"2.1. Objetivos.-Los objetivos de las investigaciones clínicas serán: Verificar que, en condiciones normales de utilización, las prestaciones de los productos son conformes a las contempladas en el apartado 3 del anexo I; y determinar los posibles efectos secundarios indeseables en condiciones normales de utilización y evaluar si éstos constituyen riesgos en relación con las prestaciones atribuidas al producto."

De esta manera nos disponemos a llevar a cabo pruebas del software sobre sujetos humanos.

Pruebas sobre la aplicación.

En este apartado llevamos a cabo una serie de entrevistas anónimas a diversas personas que han descargado, instalado y usado nuestra aplicación.

Varón, 21 años, residente en la provincia de Barcelona

Comentario:

Falta de rotación inversa.

Falta de zoom sobre el modelo.

Velocidad de rotación lenta.

Me ha parecido un buen programa diseñado para la detección de personas que pueden ser bulímicas y/o anoréxicas.

Si al programa se le pudiese dedicar más tiempo y recursos se podría pulir mucho más. Es un programa que puede mejorarse.

El programa cumple con su función

P: ¿Te han parecido realistas los resultados finales?

R: Sí, Obviando la definición de musculatura.

P: ¿Te ha parecido intuitiva la interfaz?

R: Sí, Pero añadiría una indicador que indica hacia qué lado el slider hace la parte del cuerpo más ancha y más delgada.

P: ¿Te has podido ver reflejado en el modelo de alguna manera?

R: Sí, ha llegado a reflejar mi figura

Mujer, 22 años, residente en la Comunidad Valenciana

Comentario:

Es un programa que será realmente útil cuando pueda ser mejorado.

P: ¿Te han parecido realistas los resultados?

R: Sí, pero sólo tras clicar en smooth

P: ¿Te ha parecido intuitiva la interfaz?

R: Sí

P: ¿Te has podido ver reflejado en el modelo de alguna manera?

R: Sí

Mujer, 53 años, residente en la provincia de Jaén

Comentario:

Le falta una referencia a la altura, de esta manera sería más sencillo calcular las proporciones de la persona.
Ni las manos ni los pies pueden ser modificados.

Es una buena aplicación que te permite verte como eres desde otra perspectiva

P: ¿Te han parecido realistas los resultados?

R: Sí

P: ¿Te ha parecido intuitiva la interfaz?

R: Sí

P: ¿Te has podido ver reflejada en el modelo de alguna modelo?

R: Sí

Varón, 19 años, residente en la provincia de Barcelona.

Comentario:

Me parece bien la aplicación, aunque hacen falta indicaciones en los sliders sobre hacia que lado se hace más ancho. o más fino.
No se puede introducir el número que se quiere en los sliders de forma directa.
La interfaz me parece buena.

P: ¿Te han parecido realistas los resultados?
R: Sí.

P: ¿Te ha parecido intuitiva la interfaz?
R: Sí.

P: ¿Te has podido ver reflejado en el modelo de alguna modelo?
R: Sí.

Tras el proceso de prueba de la aplicación, descubrimos varios factores que los usuarios echan de menos de forma general, Como la falta de indicaciones o una falta de intuitividad a la hora de saber cómo se va a deformar cada parte al interactuar con cada slider.

Ninguno de ellos ha tenido ningún problema para usar la interfaz y aprender como funciona de forma rápida y efectiva.

A todos los sujetos que han probado la interfaz, les han parecido realistas los resultados obtenidos.

Todos los sujetos han sido capaces de verse reflejados en el modelo que estaban editando de una forma sencilla.

Mejoras.

Nuestro software tiene errores que pueden mejorarse de alguna u otra manera, en esta sección hablamos de estos errores y de posibles ideas que no hemos podido implementar por falta de tiempo y/u otras razones.

Problema en el proceso de carga

Durante el proceso de carga de nuestra aplicación se cargan muchos ficheros y algunos de ellos son masivos, esto provoca que según la potencia del procesador donde se esté ejecutando la aplicación, el tiempo de carga inicial llegue a ser muy alto, esto es algo muy desagradable para el usuario final y además, provoca que no cumplamos totalmente con el requerimiento **N1**.

Tenemos una idea que puede reducir el tiempo de carga a un tercio de lo que llega a tardar ahora, esta es, implementar la carga de datos mediante multi-hilos. Dado que el proceso que más tiempo lleva no es la carga del fichero a memoria, si no el tratamiento del mismo para recuperar información, usar cuatro hilos para el proceso de tratamiento de archivos ".shadif" puede reducir el tiempo empleado en la carga hasta en un 60% (Teniendo en cuenta que hay muchos más proceso menores que no se verían afectados por este cambio).

Esta solución no fué implementada, dado que Unity[9] no permite un fácil uso de hilos y hay que tener un gran cuidado usándolos porque puede interferir con el proceso "core" (núcleo) del propio motor, por lo que deberíamos cortar por completo la ejecución del mismo para ejecutar nuestro código. El diseño de este código y el funcionamiento del mismo sin problema, llevaría demasiado tiempo del que no disponíamos en el momento del desarrollo.

Otra posible solución que aceleraría el tratamiento de archivos en la carga de la aplicación sería almacenar estos de forma comprimida, de manera que se cargen en memoria de forma compresa y con librerías de terceros, leer la información desde este archivo más ligero.

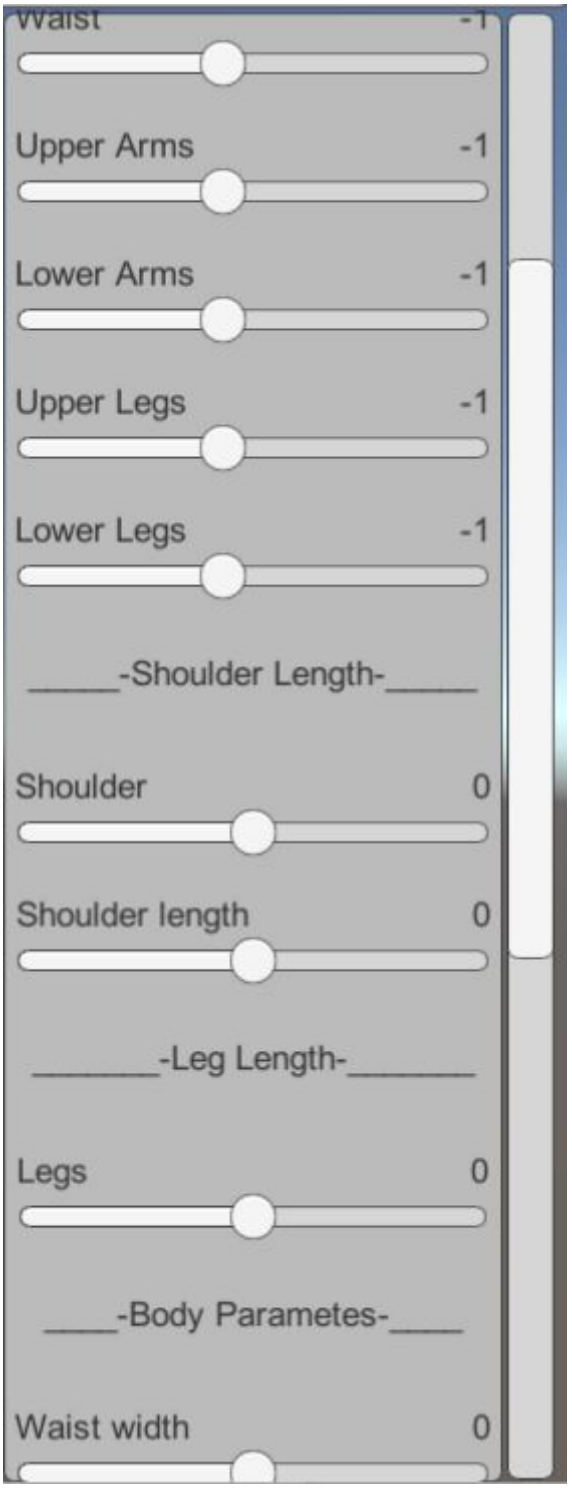
Falta de resultado final de la aplicación.

En este momento, como resultado final de la aplicación, disponemos de un simple número que nos indica la diferencia existente entre un modelo y otro, pero esto no es muy intuitivo de cara al paciente, por lo tanto, sería óptimo, investigar y hallar una manera de convertir esta aplicación en un autotest el cuál, recibiendo como argumento la diferencia entre ambos modelos (obtenida con un método obtenido de una investigación previa, no con el actual) pudiera decirnos si el paciente puede ser potencial enfermo de cualquier desorden alimenticio.

Interfaz gráfica sin desplegables.

Tal y como está diseñada nuestra interfaz gráfica, agregar menús desplegables sería una gran mejora, facilitando la navegación por los sliders y compactando los que no estemos usando.

Este problema es debido a que el sistema que nos facilita Unity[9], aún siendo usable, no nos permite la utilización de desplegables, ni de listas como las que estamos usando en este caso fácilmente, debido a esto, combinar ambos elementos, se hace extremadamente complicado.



Aparición de "artefactos" indeseables en el modelo

Esto es debido a errores en la generación de los ficheros ".shadif", pero mayormente, es provocado por ligeros cambios que llevamos a cabo en cada uno de los vértices al suavizarlos por cualquier causa. Este es un problema realmente molesto, y debido a esto, debemos alertar a los usuarios de la aplicación que "arrastren" los sliders en lugar de pulsar en ellos, ya que frente a grandes cambios entre una deformación y otra, estos "artefactos" se generan mucho más grandes y son más complicados de eliminar.

Una posible solución a este problema es realizar un algoritmo de suavizado mucho menos agresivo´cada vez que deformamos, sobre la parte afectada y las vecinas. Podríamos usar, por ejemplo, el algoritmo (al completo) que diseñan en [18] usando una función kernel con la que no suavizamos demasiado, lo cual provoca que los detalles del modelo no se pierdan.

Texturización de los modelos

Es algo que debemos probar con más usuarios, pero es posible que si texturizan el modelo para que este tenga colores, tal vez pelo o cualquier otro rasgo más allá de la forma del cuerpo, esto facilita la identificación del paciente con el modelo que está deformando.

Cambio de posición de los modelos

El hecho de que los modelos sean estáticos y no se visualice en una posición natural puede provocar que los pacientes no puedan identificarse tan fácilmente con el modelo que deforman.

Esto hecho podría ser solucionado mediante el uso de técnicas de rigging[21], proceso por el cual dotaremos a nuestro modelo de esqueleto, con el cuál podríamos fácilmente imitar posturas naturales, aunque, todas las deformaciones se harían en la posición "estándar" del modelo, y trás realizarlas, los vértices serían desplazados mediante las operaciones de traslación, rotación y escalado a la posición que nos diría el esqueleto que deberían tener.

Conclusiones.

El desarrollo de este TFG no ha sido sencillo y ha llevado mucho tiempo. El proceso de investigación llevó mucho tiempo dado que estuvimos mucho tiempo atascados en el. Por otra parte, el diseño de la aplicación fué un ejercicio muy enriquecedor en cuestiones de ingeniería del software y de estructuras de datos.

Aunque los resultados finales de nuestra aplicación tenga varios fallos que pueden llegar a molestar en algunas ocasiones, creemos la solución proporcionada es realmente buena para haber sido desarrollados por tan sólo una persona que además ha diseñado un método realista de deformación del modelo con la ayuda de sus dos tutores.

Hemos logrado diseñar e implementar una aplicación totalmente usable y afectiva, además sin gastar dinero de ninguna forma, no hemos hecho reporte de gastos dado que no lo hemos visto necesario por esta razón, todo el código usado usa licencia de código libre y gratuito, además de Unity, que al usar la licencia personal, nos obliga a mostrar su logo al iniciar la aplicación, pero no a pagar por él, siempre y cuando no comercialicemos la aplicación, en cuyo caso deberemos pagar al motor un porcentaje de los ingresos generados.

Acabamos orgullosos con el resultado obtenido así como con el trabajo realizado, tanto con la aplicación como con esta memoria, trabajo con el cual esperamos ratificar el título de ingeniería informática.

Bibliografía.

[1] Parameterized human body model for real-time applications - **Mustafa Kasap & Nadia Magnenat-Thalmann, Oct. 2007**

[2] Modeling Variability in Torso Shape for Chair and Seat Design - **Matthew P Reed & Matthew B. Parkinson, January 2008**

[3] Parameterized Human Body Modeling - **Hyewon SEO, 2004**

[4] Intro to Metaballs - **Paul Heckbert comp.graphics, 9 Nov 92**

[5] Videoclip de "Black or White" - **Michael Jackson, 1991**

[6] Manual de referencia de Python - **Versión 2.7.14rc1**

[7] MPI - SMPL - **Matthew Loper, Naureen Mahmood, Javier Romero, Gerard Pons-Moll, MIchael J. Black - 2015**

[8] CAESAR - **SAE International**

[9] Unity3d - **Motor gráfico, Versión 2017.1**

[10] Maya - **Autodesk, programa de edición y modelado 3D.**

[11] Blender - **Blender Network, Version 2.78c, programa de edición y modelado 3D.**

[12] MPI -DMPL - **Matthew Loper, Naureen Mahmood, Javier Romero, Gerard Pons-Moll, MIchael J. Black - 2015**

[13] Unreal Engine - **Motor gráfico, Versión 4.17**

[14] Especificación de Requerimientos, Diseño de bases de datos - **Universidad de Granada.**

[15] Dia Diagram Editor - **The Dia Developers. Editor de diagramas para varios propósitos.**

[16] Fundamentos de Informática y Programación - **Gregorio Martín Quetglás, Francisco Toledo Lobo & Vicente Cerverón Lleó, Septiembre 1995.**

[17] Efficient AVL Tree in C# - **Keith Wood, Febrero 2012.**

[18] Kernel-Based Laplacian Smoothing Method for 3D Mesh Denoising - **Hicham Badri, Mohammed El Hassouni, Driss Aboutajdine, Diciembre 2012**

[19] La prensa, un recurso para el aula, Módulo 1.4. Análisis de prensa. Estructura - **Ministerio de Educación, Cultura y Deporte, Gobierno de España.**

[20] REAL DECRETO 1591/2009, DE 16 DE OCTUBRE, POR EL QUE SE REGULAN LOS PRODUCTOS SANITARIOS - **BOE núm. 268, de 6 noviembre [RCL 2009, 2105]**

[21] Blender: 3D en la educación, Mecánica y Cinemática - **Ministerio de Educación, Cultura y Deporte, Gobierno de España.**

[22] La propiedad Intelectual - **Ministerio de Educación, Cultura y Deporte, Gobierno de España.**