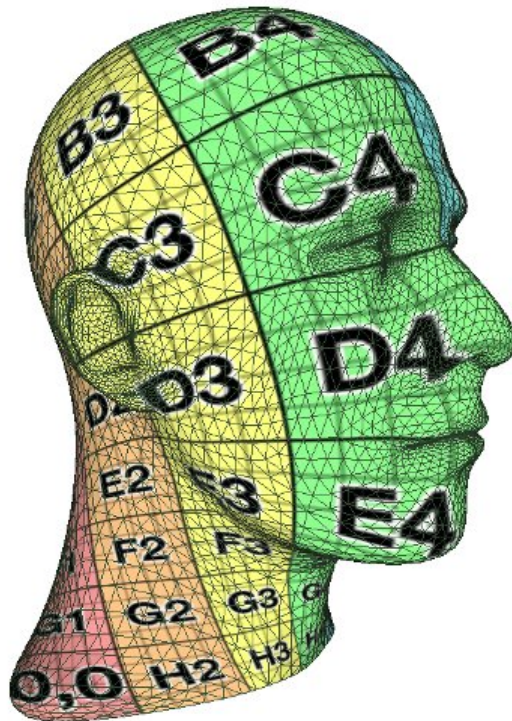


Guión de prácticas del módulo

Estructuras y Algoritmos con Mallas de Triángulos



*Máster Universitario en Informática Gráfica, Juegos y Realidad Virtual de la
Universidad Rey Juan Carlos.*

Objetivos y normativa

El objetivo de la práctica es realizar una serie de ejercicios de diseño e implementación de algoritmos que trabajan sobre mallas de triángulos. La práctica se divide en tres ejercicios independientes.

La implementación se realizará en C++, dejando libre elección del sistema operativo y compilador a emplear para la misma. Se incluyen ficheros de proyecto para compilar usando los entornos Visual Studio y Cmake.

Para aprobar la práctica será necesario implementar correctamente la parte obligatoria de los tres ejercicios propuestos. Además, se propondrá la realización de parte optativa para subir la nota obtenida.

La entrega consistirá en un archivo .zip con los archivos de código fuente modificados por el alumno, enviado mediante el módulo correspondiente en la sección calificación de la web aulavirtual. Para evitar que el archivo de entrega crezca demasiado, es suficiente enviar los ficheros .cpp y .hpp, no siendo necesario enviar los ejecutables del programa. De modo orientativo, el fichero .zip entregado no debería ocupar más de 5 Mb. Recuerda rellenar con tu nombre y email la cabecera de cada fichero fuente que modifiques (busca la palabra TODO: en cada fichero).

El fichero de entrega deberá contener además del código solicitado, una breve memoria en formato pdf conteniendo la salida por pantalla y una captura del resultado para al menos las mallas mannequin2.ply (ejercicios 1 y 2) y mask2.off (ejercicio 1, 2, 3). No se aceptarán entregas sin la memoria correspondiente.

Todo el mundo es responsable de la custodia de su propio código. La detección de copias supondrá suspender toda la asignatura a todos los implicados en la copia.

La versión definitiva de la práctica deberá entregarse antes de las 23:59 del día 30 de noviembre.

Los alumnos que no entreguen a tiempo o suspendan la práctica en primera convocatoria, podrán presentarla en segunda convocatoria hasta el 1 de junio.

Material de apoyo

Para reducir el tiempo dedicado a implementación se ofrece un material de apoyo que puede ser descargado desde la página web de la asignatura en el aula virtual.

El material de apoyo consta de:

- Meshlab: Es un visualizador de ficheros 3D bastante potente, que soporta muchos formatos de fichero y realiza una variedad de algoritmos geométricos. Puede descargarse desde <http://meshlab.net/>. Se recomienda instalarlo en el directorio C:\Meshlab.
- Fmfig-prac2021.rar: Contiene el código fuente de apoyo, para reducir el tiempo de desarrollo total de la práctica.

Uso de Meshlab como visualizador externo en nuestros programas

El programa meshlab se ofrece como una forma de evitar la implementación de métodos de renderizado de la malla. Puede ser sustituido por cualquier otro programa capaz de visualizar

ficheros en formato .PLY o .OBJ, como mview¹, QtViewer², 3D Studio Max, Blender3D, Unity, etc...

Meshlab admite como argumento un nombre de fichero 3D a visualizar. Para usarlo desde la interfaz gráfica de windows hay que arrastrar el fichero 3D a visualizar sobre el icono de meshlab, o bien asociar la extensiones .off, .obj, .ply con el programa.

Desde nuestros programas, podemos lanzar meshlab mediante la función system() de C++. Al final de varios de los ficheros de código de ejemplo se puede encontrar como implementar la llamada.

```
//Visualize the output file using an external viewer
string viewcmd = "C:/meshlab/meshlab.exe";
string cmd = viewcmd + " " + filename;
cout << "Executing external command: " << cmd << endl;
system(cmd.c_str());
```

El código anterior contiene la ruta completa al ejecutable de meshlab. Se puede modificar el valor inicial de viewcmd para apuntar a otro ejecutable diferente.³

Código de apoyo

El código de apoyo se encuentra en el fichero FMFIG-prac2021.rar, y se divide en dos partes: código de las clases auxiliares ya listas para usar, y esqueletos para los ejercicios a realizar.

Los ficheros con clases auxiliares que se entregan son:

- vec3.hpp : Contiene la implementación de una clase para almacenar puntos y vectores en 3D y las operaciones más comunes: suma, resta, distancia, productos escalar (*) y vectorial (^), normalización, etc...
- SimpleMesh.hpp : Contiene la implementación de la clase *SimpleMesh* y las clases auxiliares *SimpleEdge* y *SimpleTriangle*. Es un ejemplo de implementación de malla de triángulos que contiene una lista de vértices compartidos y una lista de triángulos sin información de vecindad entre triángulos.
- ColorMesh.hpp : Contiene una extensión de la clase SimpleMesh para incluir un campo capaz de almacenar un color RGB por cada vértice.

La clase SimpleMesh contiene métodos para acceder a coordenadas y triángulos, calcular algunos parámetros como área y centroide de triángulos y lectura y escritura de ficheros en los formatos .OFF, .OBJ y .PLY

1 <http://mview.sourceforge.net/>

2 <http://www.openmesh.org/>

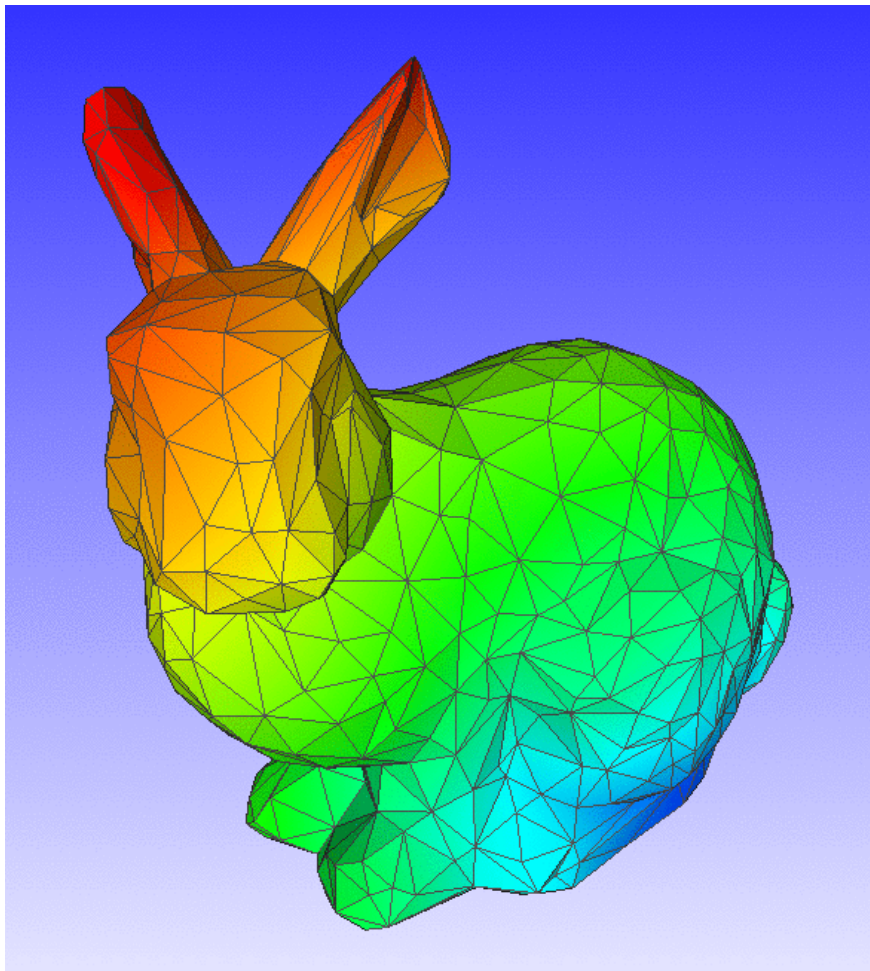
3 Se recomienda que la ruta no contenga espacios para no tener que introducir códigos de escape de C++ en la cadena. En caso de haber instalado meshlab en su directorio original, probad la ruta:
string viewcmd = "\"C:\\Program Files (x86)\\VCG\\MeshLab\\meshlab.exe\"";

Ejercicio 0. Ejemplo de carga y visualización de una malla.

El fichero **meshColor.cpp** contiene un programa de ejemplo que realiza los siguientes pasos:

1. Carga una malla en formato OFF, OBJ o PLY (por defecto del fichero bunny.off , pero puede modificarse pasando un argumento al programa)
2. Calcula la distancia Euclídea desde cada vértice al primer vértice de la malla y su máximo.
3. Asigna colores a los vértices basándose en la distancia Euclídea de cada vértice al primer vértice de la malla. Para ello se usa el método `RGBColor::setTemperature(d, dmin, dmax)`, que asigna un color basándose en la posición de `d` dentro del intervalo `[dmin..dmax]`. Los vértices más cercanos aparecerán en azul y los más alejados en rojo.
4. Salva la malla en formato .PLY con colores y la visualiza con un programa externo (por defecto meshlab, pero puede modificarse por cualquier otro)⁴.

No es necesario hacer nada sobre el código, salvo tal vez modificar la ruta del programa externo empleado como visualizador al final del program. Solamente se entrega como un ejemplo de uso.



Salida del programa meshColor, visualizado con meshlab.

⁴ Es posible que tengas que modificar el código para incluir la ruta completa al programa de visualización usado. Procura que la ruta completa no incluya espacios.

Ejercicio 1. Cálculo de medidas estadísticas sobre una malla.

El fichero **meshStatistic.cpp** contiene un programa de ejemplo que realiza los siguientes pasos:

1. Carga una malla tipo SimpleMesh (por defecto del fichero mannequin2.ply)
2. Imprime algunas estadísticas sobre la malla (número de vértices, triángulos, etc...)

Se pide completar la implementación de las siguientes tareas:

- TODO 1.1 : Calcular y mostrar por pantalla los siguientes parámetros:
 - Área mínima, máxima y promedio de las facetas de la malla. Área total de la malla.
 - Mínimo y máximo de los ángulos internos a los triángulos de la malla.
 - Mínimo, máximo y promedio de las longitudes de arista de la malla.
 - Mínimo, máximo y promedio de los factores de forma de los triángulos de la malla.
Como factor de forma se usara la fórmula:
$$\frac{\text{arista mínima}}{\text{circumradio} \cdot \sqrt{3}}$$

Se espera que el factor de forma tome su valor máximo (1.0) en los triángulos equiláteros, y valor mínimo (0.0) en los triángulos degenerados de la malla.
- TODO 1.2 : Identificar y escribir por pantalla los índices de los triángulos cuyo factor de forma sea menor que un umbral dado en la variable shapeFactorTh (por defecto $1/(4 \cdot \sqrt{3})$).
- TODO 1.3 : Salvar una malla con colores en los vértices llamada "output_meshStatistic.ply", que asigne el color rojo (RGBColor(1,0,0)) a los vértices de la malla que pertenezcan a algún triángulo con factor de forma inferior a shapeFactorTh, y color blanco a todos los demás vértices.

Nota: No se aceptarán como válidos los ejercicios que tarden más de 10 segundos en calcular las estadísticas sobre cualquiera de las mallas.

Ejercicio 1. Parte optativa: Cálculo de áreas mixtas por vértice.

Como parte optativa del programa meshStatistic.cpp, se pide calcular el área de la celda mixta de Voronoi para cada vértice, definida como sigue:

$$A_{mix} = 0$$

Para cada triángulo T vecino del vértice v_i :

Si T es obtuso: //Evitamos uso de Voronoi

Si el ángulo de T en vértice v_i es obtuso:

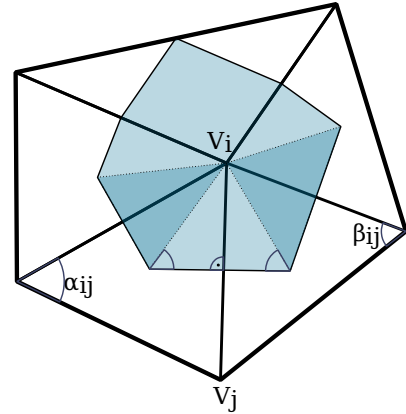
$$A_{mix} += \text{Area}(T)/2$$

Else:

$$A_{mix} += \text{Area}(T)/4$$

Else: //Voronoi es seguro

$$A_{mix} += \frac{1}{8} \cdot \sum_{j \text{ vecino } i} (\cot \alpha_{ij} + \cot \beta_{ij}) \cdot \|\vec{V}_j - \vec{V}_i\|^2$$



Generalmente, es más sencillo calcular el resultado de la ecuación anterior reordenando los términos para hacer un cálculo fuera de orden, asociando los términos por triángulos en lugar de por vértices vecinos

$$A_{mix} = 0$$

Para cada triángulo T vecino del vértice v_i :

Si T es obtuso: //Evitamos uso de Voronoi

Si el ángulo de T en vértice v_i es obtuso:

$$A_{mix} += \text{Area}(T)/2$$

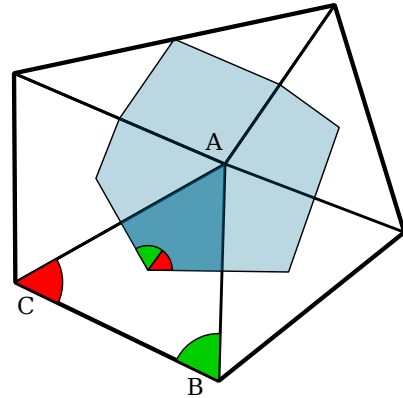
Else:

$$A_{mix} += \text{Area}(T)/4$$

Else: //Voronoi es seguro

Para cada vértice de T :

$$A_{mix} += \frac{1}{8} \cdot (\|AB\|^2 \cdot \cot \hat{C} + \|AC\|^2 \cdot \cot \hat{B})$$



Nota: Podemos comprobar si las áreas de los vértices calculadas son correctas calculando la superficie total de la malla sumando las áreas por vértice y comparando su valor con la superficie total de la malla que obtuvimos en el punto 1.

Por ejemplo, la salida esperada para el fichero de entrada mallas/mannequin.ply es :

```
Loading file mallas/mannequin.ply
```

```
Num vertex: 6743 Num triangles: 13424 Unreferenced vertex: 0
```

```
Area    min: 0.0013 max: 0.7581 average: 0.1213 total: 1628.8569
```

```
Angle   min: 0.1189 max: 2.7538
```

```
EdgeLen min: 0.0382 max: 1.8653 average: 0.4967
```

```
ShapeF  min: 0.1370 max: 0.9978 average: 0.7195
```

```
VertexA min: 0.0045 max: 1.3521 average: 0.2416 total: 1628.8569
```

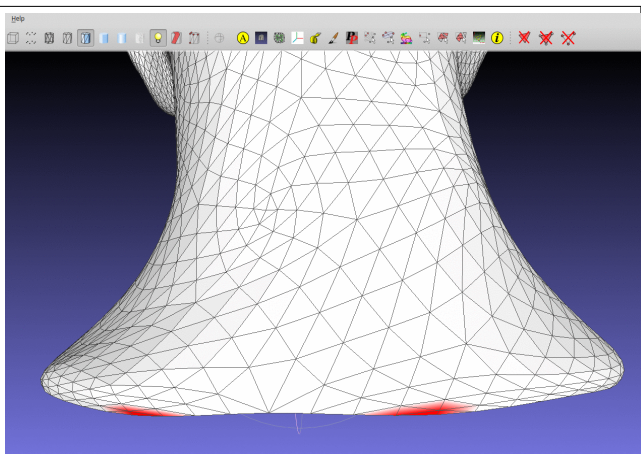
```
3 triangles with ShapeFactor < 0.1443
```

```
Triangle 2108 has ShapeFactor 0.1441
```

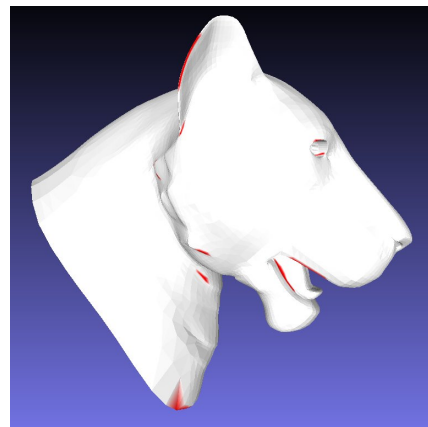
```
Triangle 3592 has ShapeFactor 0.1370
```

```
Triangle 3856 has ShapeFactor 0.1420
```

```
Saving output to output_meshStatistic.ply
```



Triángulos degenerados en mannequin.ply



Triángulos degenerados en lion.ply

El directorio outputs contiene las estadísticas y mallas de salida esperadas para varias mallas de ejemplo.

Ejercicio 2. Búsqueda de aristas frontera de una malla.

El fichero **meshBoundary.cpp** contiene un programa de ejemplo que realiza los siguientes pasos

1. Carga una malla del tipo SimpleMesh (por defecto, el fichero 16Triangles.off) e imprime algunas estadísticas sobre el mismo (número de vértices, triángulos, etc...)
2. TODO 2.1: Llama al método *updateEdgeLists()*, esperando que la función actualice la lista de aristas frontera (aquellas que pertenecen a un triángulo que no tienen un triángulo vecino) y la lista de aristas interiores a la malla. Un algoritmo que realiza esta función se explicó el día 4 de la asignatura. Reordenar la lista de aristas frontera de forma que forme un bucle ordenado, de forma que:
 - El vértice “a” de la primera arista será el de menor coordenada X de todos los vértices de frontera. En caso de empate, aquel de ellos que tenga menor coordenada Y. En caso de empate, aquel de ellos que tenga menor coordenada Z.
 - El resto de aristas forman una (o varias) listas circulares, es decir, el índice “a” de cada arista corresponderá con el índice “b” de la arista anterior (ver ejemplo más adelante)
3. Escribe por pantalla el número de aristas frontera (boundary edges) y aristas interiores (internal edges) de la malla. Escribe la lista de aristas frontera. Escribe la longitud lineal de la frontera y la característica de Euler de la malla.
4. TODO 2.2: Para cada vértice de la malla, calcular la distancia superficial al vértice de la frontera más cercano. La distancia superficial se medirá mediante caminos formados por aristas internas de la malla. El resultado se almacenará en un vector llamado “boundDist”, y el índice del vértice más alejado de la frontera se almacenará en la variable “deepestVertex”.
5. TODO 2.3: Visualizar gráficamente el resultado anterior. Emplea una malla de tipo ColorMesh y modifica el color de los vértices usando una paleta de temperatura, de forma que el color pase de rojo a azul conforme los vértices se alejan de la frontera.⁵ Salvar la malla de colores en un fichero llamado “*output_boundary.obj*” y visualizar el fichero con meshlab.⁶

Por ejemplo, la salida esperada para el fichero de entrada **16Triangles.off** es :

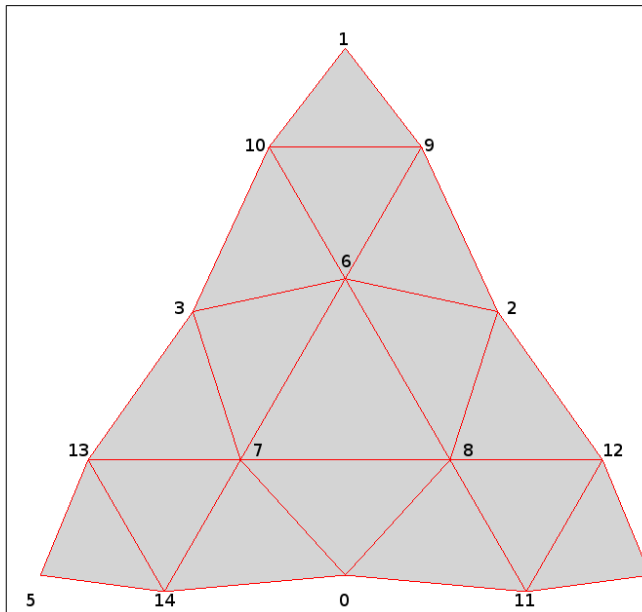
```
Loading file mallas/16Triangles.off
Num vertex: 15 Num triangles: 16 Unreferenced vertex: 0
Done updateEdgeLists() 1.3e-05 seconds
12 boundary edges
18 internal edges
Edges in the boundary:
[5->14] [14->0] [0->11] [11->4] [4->12] [12->2] [2->9] [9->1] [1->10]
[10->3] [3->13] [13->5]
Boundary length: 9.85725
Euler Characteristic: 1
Done boundDist() 4.3e-05 seconds
maxDistance: 0.816499 at vertex: 7
Done colorize by distance to boundary 5.7e-05 seconds
Saving output to file: output_boundary.obj
```

⁵ Se puede ver un ejemplo sencillo de asignación de colores en el fichero de ejemplo **meshColor.cpp**.

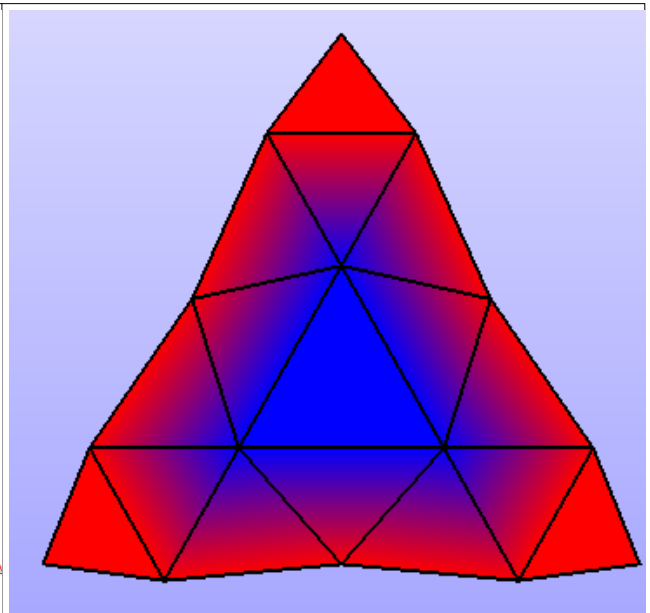
⁶ Tu resultado debe ser similar al filtro de meshLab **Filters**→**Color creation**→**Colorize by border distance**

Nótese como las aristas están ordenadas, siendo el segundo vértice de cada arista igual al primer vértice de la arista siguiente. La unión ordenada de todas las aristas proporciona uno (o varios) ciclos cerrados.

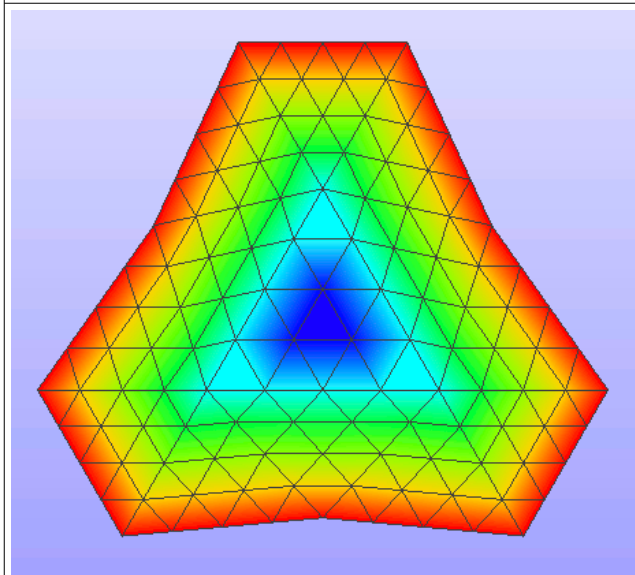
$5 \rightarrow 14 \rightarrow 0 \rightarrow 11 \rightarrow 4 \rightarrow 12 \rightarrow 2 \rightarrow \dots$



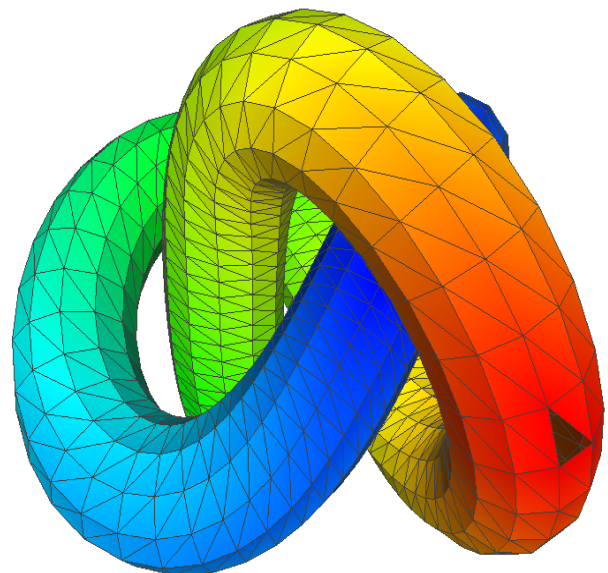
Mapa de vértices de la malla 16Triangles



Salida esperada para 16Triangles.
Rojo = Frontera. Azul = Interior




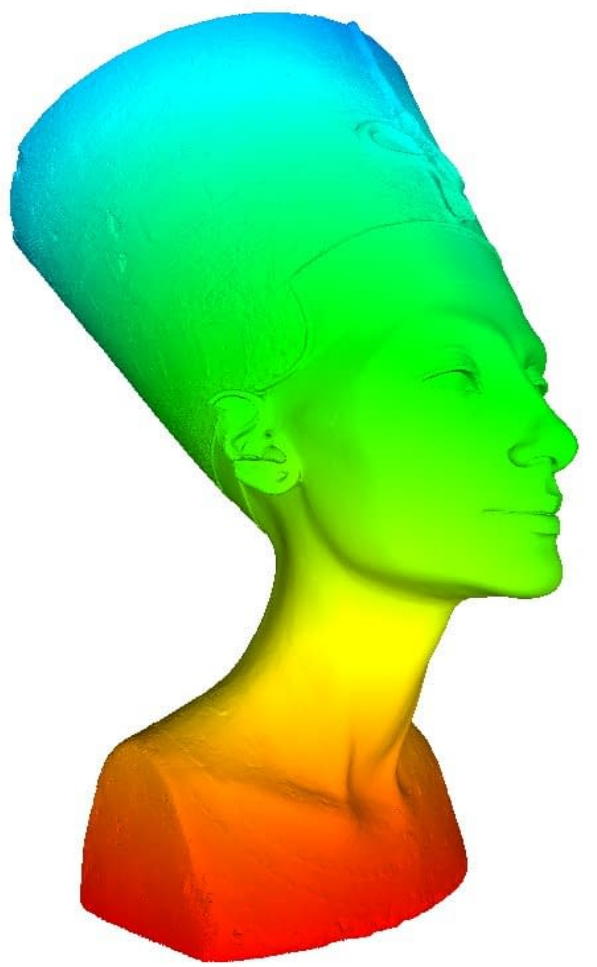
Salida esperada para hexagon2
maxDistance: 1.019 at vertex: 38



Salida esperada para knot-hole
maxDistance: 395.75 at vertex: 1374

Ejercicio 2: Parte optativa. Implementación eficiente

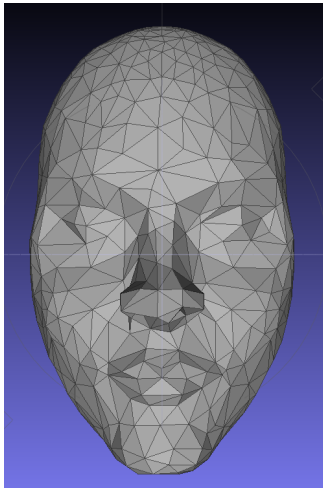
1. Optimiza tu solución del ejercicio 2 hasta el punto de que sea capaz de calcular **en menos de cinco segundos** la frontera y la distancia superficial mínima de cada vértice interior hasta el vértice más próximo de la frontera, para la malla **Nefertiti.990kv.ply** (incluida en el fichero **mallas-extra.zip**).
Los tiempos deben medirse compilando el programa en modo Release (para activar las optimizaciones) y se tomaran del mensaje en consola “Done colorize by distance to boundary xxxxx seconds”, para no tener en cuenta el tiempo de escribir el fichero de salida.
2. Comprueba que tu programa admite mallas con múltiples fronteras, como el fichero **angel_kneeling.150kv.ply** incluido en el directorio de mallas extra. El coloreado debe seguir siendo rojo alrededor de las fronteras y azul conforme se aleja de las mismas. Comprueba que tu programa también admite mallas cerradas sin fronteras externas, en cuyo caso todos los vértices deben pintarse en color blanco.

	
Salida esperada para angel_kneeling.150kv.ply maxDistance: 13.7901 at vertex: 44431	Salida esperada para Nefertiti.990kv.ply maxDistance: 628.777 at vertex: 429856

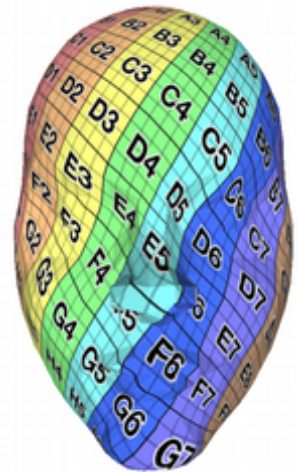
El directorio outputs contiene las distancias y mallas de salida esperadas para varias mallas de ejemplo.

Ejercicio 3. Parametrización (cálculo de coordenadas UV) de una malla

El objetivo del ejercicio es calcular una parametrización sencilla de la malla de entrada, asignando a cada vértice de la misma una coordenadas UV que permita mapear la superficie 3D sobre un cuadrado.



0,1	A2	A3	A4	A5	A6	A7	1,1
B1	B2	B3	B4	B5	B6	B7	B8
C1	C2	C3	C4	C5	C6	C7	C8
D1	D2	D3	D4	D5	D6	D7	D8
E1	E2	E3	E4	E5	E6	E7	E8
F1	F2	F3	F4	F5	F6	F7	F8
G1	G2	G3	G4	G5	G6	G7	G8
0,0	H2	H3	H4	H5	H6	H7	1,0



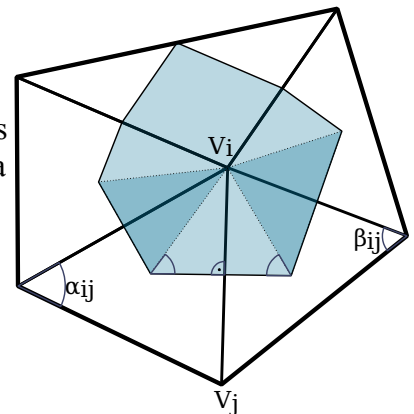
Para ello, vamos a implementar un método de Barycentric Mapping similar al sugerido en la sección 5.3 del libro “Polygon Mesh Processing”, 2010. El procedimiento se repasará en la clase de práctica, y debemos implementarlo usando **matrices densas** de Eigen sobre el fichero **meshTexture-Dense.cpp**.

El algoritmo que seguiremos será el siguiente:

1. Partimos de una malla que debe cumplir la condición de ser topológicamente equivalente a un disco, es decir, *tener una frontera externa no vacía cuyos vértices puedan ser ordenados formando un ciclo único y cerrado*. En nuestro directorio *mallas* podremos encontrar varios ejemplos, como *16Triangles.off*, *Mask2.off*, *Mask3.off*, *Mannequin.off*, ...
2. Calculamos la lista de vértices que forma parte de la frontera, y los ordenamos de forma que formen un ciclo cerrado (debería estar hecho como parte del ejercicio *meshBoundary*).
3. Calculamos los elementos fuera de la diagonal de la matriz Laplaciana como:

$$\vec{L}_{ij} = \begin{cases} \cot \alpha_{ij} + \cot \beta_{ij} & \text{Si vértice } i \text{ vecino de vértice } j \\ 0 & \text{Si vértice } i \text{ no vecino de vértice } j \end{cases}$$

Los ángulos α_{ij} y β_{ij} son los correspondientes a los ángulos opuestos de los dos triángulos de la malla que tienen la arista V_j-V_i como frontera, según puede verse en la ilustración.



De forma orientativa, estos son los valores esperados para la matriz correspondiente al fichero 16Triangles.off (los números se han truncado a dos decimales)

0	0	0	0	0	0	0	1.61	1.61	0	0	0.32	0	0	0.32
0	0	0	0	0	0	0	0	0	0.77	0.77	0	0	0	0
0	0	0	0	0	0	1.61	0	1.61	0.32	0	0	0.32	0	0
0	0	0	0	0	0	1.61	1.61	0	0	0.32	0	0	0.32	0
0	0	0	0	0	0	0	0	0	0	0	0.77	0.77	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0.77	0.77
0	0	1.61	1.61	0	0	0	0.67	0.67	1.33	1.33	0	0	0	0
1.61	0	0	1.61	0	0	0.67	0	0.67	0	0	0	0	1.33	1.33
1.61	0	1.61	0	0	0	0.67	0.67	0	0	0	1.33	1.33	0	0
0	0.77	0.32	0	0	0	1.33	0	0	0	0.84	0	0	0	0
0	0.77	0	0.32	0	0	1.33	0	0	0.84	0	0	0	0	0
0.32	0	0	0	0.77	0	0	0	1.33	0	0	0	0.84	0	0
0	0	0.32	0	0.77	0	0	0	1.33	0	0	0.84	0	0	0
0	0	0	0.32	0	0.77	0	1.33	0	0	0	0	0	0	0.84
0.32	0	0	0	0	0.77	0	1.33	0	0	0	0	0	0.84	0

4. Terminamos de calcular la matriz Laplaciana, calculando los elementos de la diagonal de forma que la suma de los elementos por cada fila de la matriz valga cero.

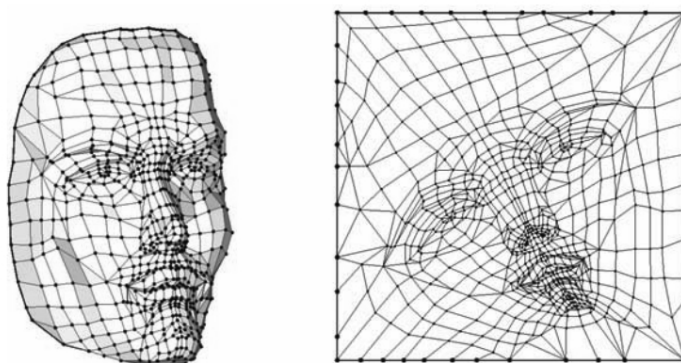
$$\vec{\vec{L}}_{ij} = \begin{cases} L_{ij} & Si\ i \neq j \\ -\sum_{j=1..N} L_{ij} & Si\ i=j \end{cases}$$

5. Emplearemos esta matriz como base para plantear un sistema de ecuaciones, para ello :

1. Calculamos una *matriz del sistema*, parcheando las filas de la matriz laplaciana que corresponden a vértices de la frontera exterior de la malla. Puesto que para esos vértices vamos a asignar un valor de UV mediante heurística en el siguiente paso.

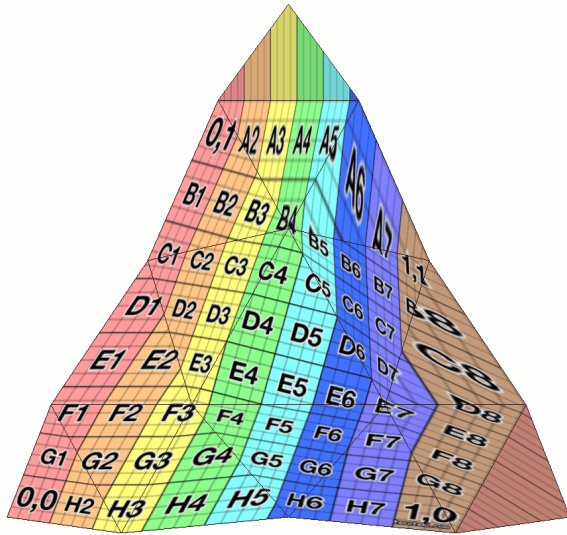
$$\vec{\vec{M}}_{ij} = \begin{cases} L_{ij} & Si\ i \notin Frontera \\ 0 & Si\ i \in Frontera, i \neq j \\ 1 & Si\ i \in Frontera, i = j \end{cases}$$

2. Asignamos valores para \overline{UV}_0 a los vértices externos. Por ejemplo, podemos redistribuirlos de forma equidistante a lo largo de la frontera de un cuadrado de lado unidad, según se sugiere en el *método de Floater*. Los valores \overline{UV}_0 para vértices internos lo dejamos como (0,0).

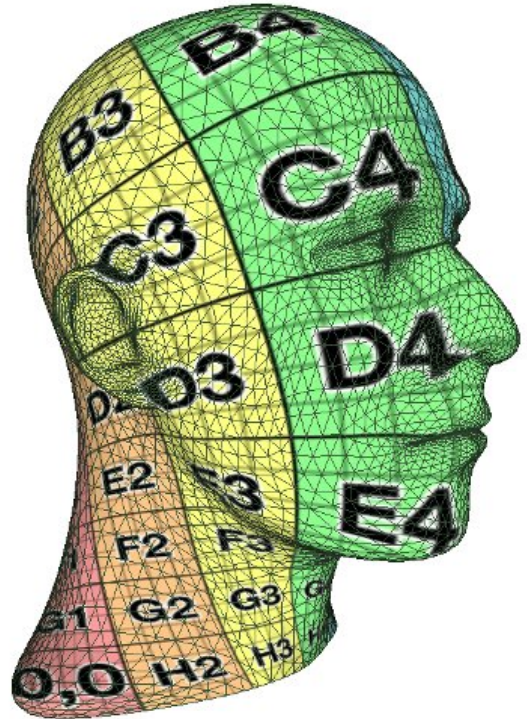


6. Resolvemos el sistema de ecuaciones $\overline{M} * \overline{UV} = \overline{UV}_0$. La sección 6 del código de ejemplo incluye los comandos necesarios para resolver el sistema de ecuaciones usando Eigen. Será suficiente para mallas pequeñas en un tiempo de cálculo razonable.

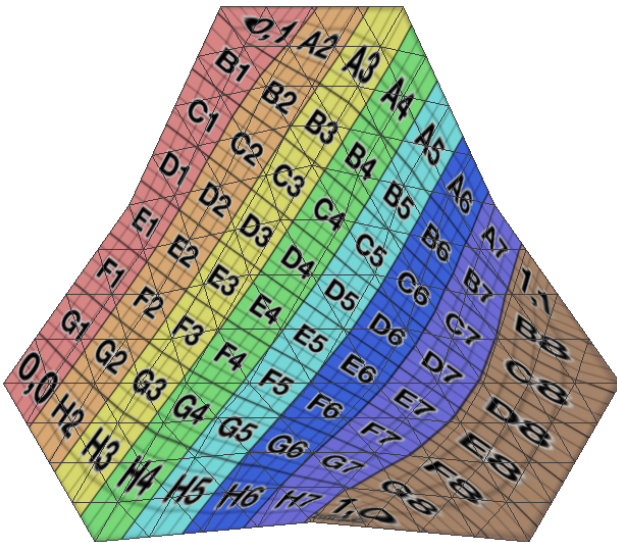
Las siguientes imágenes contienen el resultado esperado para varios ficheros de entrada.



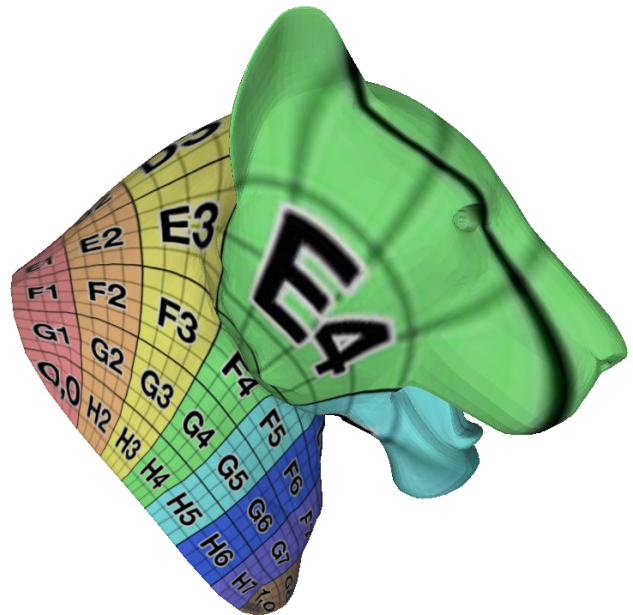
16Triangles.off



Mannequin2.ply



Hexagon2.off



Lion.ply

Parte optativa del problema 3.

El punto débil de la implementación del ejercicio anterior es que la matriz densa *meshMatrix* puede fácilmente agotar la memoria del computador porque crece muy rápidamente conforme crece el numero de vértices de la malla. Por ello, se pide:

- Reescribir el algoritmo que hemos descrito en el ejercicio anterior empleando esta vez el fichero **meshTexture-Sparse.cpp**, y empleando esta vez **matrices dispersas** de Eigen⁷ para almacenar la matriz Laplaciana y resolver el sistema de ecuaciones. De forma orientativa, la malla del fichero *mannequin2.ply* debería procesarse ahora en menos de un segundo en modo Release. Asegúrate de desactivar el volcado de las matrices intermedias a fichero poniendo la variable *dumpMatrix* a *false*.
- Optimizar tu código hasta el punto de conseguir procesar cualquier malla con menos de 50.000 vértices (por ejemplo, la malla *maxplanck.45kv.ply*) en **menos de dos segundos**.



Requisitos y notas:

- Todos los ficheros fuente modificados por el alumno deberán llevar el nombre y email del autor y la fecha en el espacio reservado para ello en el comentario de cabecera de cada fichero.
- Realiza limpieza del código que hayas escrito para depurar tus programas antes de la entrega. Se penalizará con 0.5 puntos cada uno de los siguientes problemas:
 - La entrega de ficheros fuente sin los datos del autor en la cabecera.
 - La aparición por pantalla de mensajes no esperados, introducidos por el alumno.
 - La escritura de ficheros no solicitados, incluyendo volcados de las matrices de meshTexture o volcados de mallas con nombres diferentes a los solicitados.
 - Los warnings del compilador indicando que una variable no ha sido utilizada.
- Todo tu código debería quedar entre las etiquetas “//TODO” y “//END TODO” que encontrarás a lo largo del código. También puedes añadir funciones o estructuras de datos auxiliares al principio del fichero si lo crees necesario. No toques el código destinado a calcular los temporizadores. No cambies los nombres de los ficheros de salida.
- Los tiempos para evaluar la parte optativa se medirán en el PC del profesor (un intel i7-4790, con 8 Gb de RAM), compilando en modo Release y sin tener en cuenta los tiempos de lectura y escritura de los ficheros de malla.
- Con el objetivo de ayudarte en la depuración, se incluyen varias imágenes con los ángulos y aristas de la malla 16Triangles.off en el directorio mallas.
- Las mallas hexagon0.off, hexagon1.off, hexagon2.off y hexagon3.off son especialmente adecuadas para depurar los ejercicios meshBoundary y meshTexture, pues incrementan el número de vértices poco a poco.
- No se aceptará la entrega sin la memoria conteniendo las capturas de pantalla de la salida de cada uno de los ejercicios en el PC del alumno.
- Procura no dejar tu trabajo para el último día. El plazo de entrega es inamovible.
- En caso de detectarse copias entre alumnos, todas las partes serán evaluadas como 0.0