



## Ejercicios Tema 4 – parte 1 (Shaders de Teselación)

### Material de consulta

Se recomienda repasar los siguientes recursos online antes de comenzar a resolver los ejercicios.

- Diapositivas del tema 4 de la asignatura, diapos: 1-37.
- Videos explicativos de las diapositivas 1-37 del tema 4.  
(<https://aulavirtual.uv.es/mod/kalvidres/view.php?id=570053>)  
(<https://aulavirtual.uv.es/mod/kalvidres/view.php?id=570059>)  
(<https://aulavirtual.uv.es/mod/kalvidres/view.php?id=570063>)  
(<https://aulavirtual.uv.es/mod/kalvidres/view.php?id=570066>)
- Red book OpenGL Programming guide, 8th Edition, pag: 485-508.  
(<https://www.cs.utexas.edu/users/fussell/courses/cs354/handouts/Addison.Wesley.Opacity.Programming.Guide.8th.Edition.Mar.2013.ISBN.0321773039.pdf>)
- Graphics Shaders : Theory and Practice, 2nd Edition, pag: 315-352.  
Disponible online a través de VPN en trobes +, Servei de biblioteques de la UV  
(<https://ebookcentral.proquest.com/lib/univalencia/detail.action?docID=830238>)
- OpenGL 4 Shading Language Cookbook, 2nd Edition, pag: 215-261  
Disponible online a través de VPN en trobes +, Servei de biblioteques de la UV  
(<https://ebookcentral.proquest.com/lib/univalencia/detail.action?docID=1441782>)

### Ejercicios Tema 4 – parte 1

Crea un proyecto en Microsoft Visual Studio e incluye en él los ficheros de la carpeta código. Compila y ejecuta el programa. En este ejemplo, se dibuja un cubo definido como un conjunto de 6 QUADS (uno para cada una de las caras del cubo). El cubo está centrado en el origen de coordenadas del S.R. del mundo y sus lados tienen longitud 2 (es decir, los vértices del cubo están definidos dentro del intervalo  $[-1,-1,-1]$  y  $[1,1,1]$ ). Pulsando la tecla 'W' se activa/desactiva el modo alambre y pulsando la tecla 'L' se activa/desactiva la iluminación.

En los siguientes ejercicios usaremos shaders de teselación para convertir el cubo en una esfera de radio 1, centrada también en el origen de coordenadas (es decir, una esfera inscrita dentro del cubo).

Nota: en una esfera de radio 1, el vector posición de sus vértices está normalizado (tiene longitud 1) y el vector normal para cada vértice es igual al vector posición de dicho vértice.

Para ello, hay que realizar las siguientes tareas:



### 1.1 - Esfera con nivel de teselación fijo

Vamos a convertir el cubo en una esfera usando los shaders de teselación. Para ello, hay que modificar el código de los siguientes ficheros:

*Código de la aplicación:*

- El objeto programa, además de los shaders de vértice y fragmento, debe incluir los dos shaders de teselación (el de control y el de evaluación).
- Hay que definir dentro de la función `initCube` el número de vértices por parche y sustituir la primitiva de dibujo `GL_QUADS` por `GL_PATCHES` (cuando se emplea shaders de teselación la primitiva de dibujo utilizada en la aplicación debe ser `GL_PATCHES`).

*Código del shader de vértice:*

- El valor escrito dentro de la variable `gl_Position` será ahora la posición del vértice sin transformar (`aPosition`, sin multiplicar por la matriz de transformación de modelo-vista-proyección).
- La única variable de salida será `vColor` (las otras dos se crearán en el shader de evaluación a partir de los datos de la esfera)

*Código del shader de control de la teselación:*

- Este shader se ejecuta una vez por cada vértice/punto de control de salida (4 veces para cada parche). Dentro del código hay que recoger las posiciones de los vértices/puntos de control de entrada (dentro de `gl_in[ ]`) y pasárselos al shader de evaluación (copiando las posiciones dentro de `gl_out[ ]`).
- Luego, hay que definir el nivel de teselación del interior y del contorno del parche (la tarea de teselación la realizará la etapa de *Tessellation Primitive Generator*).
- Además, habría que recoger la variable con la información de color proporcionada por el vertex shader (`vColor`) y pasársela al shader de evaluación.

Nota: `vColor` será aquí un array y su tamaño será igual que el de `gl_in`.

Nota: para la primera parte, el nivel de teselación en el contorno y en el interior del parche es fijo e igual a la constante `uTessLevel` definida dentro del código.

*Código del shader de evaluación de la teselación:*

- Este shader se ejecuta una vez por cada vértice de salida (generado por la etapa de *Tessellation Primitive Generator*). Dentro del código hay que transformar las coordenadas paramétricas del vértice al S.R. local del objeto (es el sistema de referencia de los puntos de control) y, luego, convertirlo en un vértice de la esfera (cosa que se consigue simplemente normalizando la posición). La posición resultante, transformada al espacio de la ventana (multiplicada por la matriz de modelo-vista-proyección) se guarda dentro de la variable predefinida `gl_Position`.
- Además, el shader tiene que generar como datos de salida (`out`) toda la información relacionada con el vértice (normal S.R. vista, posición del vértice S.R. vista y color). Estos datos se recogerán como datos de entrada en el shader de fragmento.

Nota: Para convertir las coordenadas paramétricas en coordenadas del S.R. de los puntos de control, se calcula primero los vectores que conecta el punto de control 0



(`gl_in[0].gl_Position`) con los puntos de control 1 y 3 (se obtienen dos vectores, uno representa el lado inferior del parche y el otro el lado izquierdo). Con esos dos vectores se obtiene la posición sumándole a la posición del punto de control 0, el vector del lado inferior multiplicada por la coordenada x, más el vector del lado izquierdo multiplicada por la coordenada y.

*Código del shader de fragmento:*

- Hay que recoger como datos de entrada (`in`) los datos proporcionado por el shader de evaluación e utilizarlos para colorear el fragmento, bien utilizando como color el que se le pasa desde el shader de evaluación, o bien como resultado del proceso de iluminación calculado a partir de la información de la posición y normal pasada por este shader (iluminación por pixel).

### 1.2 - Esfera con nivel de teselación variable, global para toda la esfera

Modifica el código del shader de control para que el nivel de teselación cambie con la distancia de la esfera al punto de vista.

Nota: para obtener el nivel de teselación emplea la función definida en el código (`level`) pasándole como parámetro el centro de la esfera en el S.R. del local al objeto (en este S.R. local al objeto, el centro de la esfera está situado en la posición  $(0, 0, 0)$ )

Nota: Esta opción se activará cuando el usuario pulse la tecla 'T'.

### 1.3 - Esfera con nivel de teselación variable para cada parche

Modifica, de nuevo, el código del shader de control para que el nivel de teselación para cada parche sea función de la distancia del centro del parche al punto de vista.

Nota: Esta opción se activará cuando el usuario pulse de nuevo la tecla 'T'.

Nota: Observa los resultados y comprueba qué pasa cuando dos parches vecinos tienen distinto nivel de teselación.

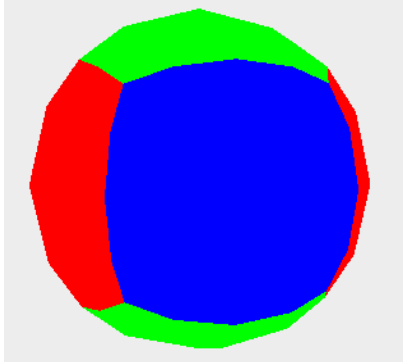
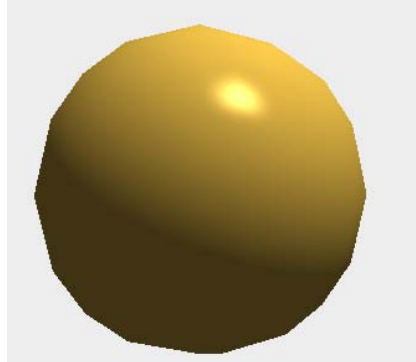
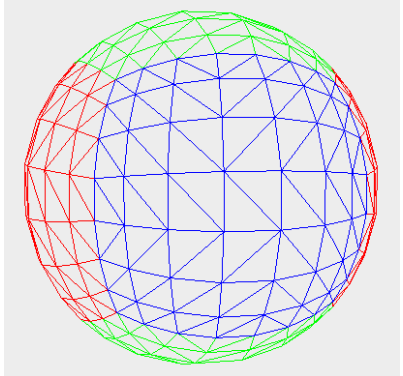
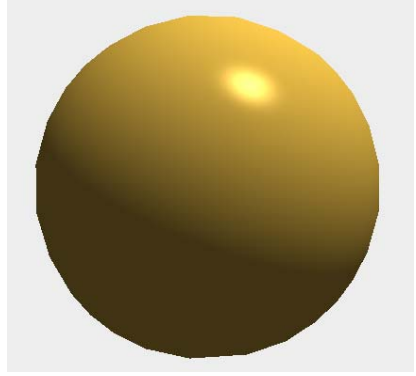
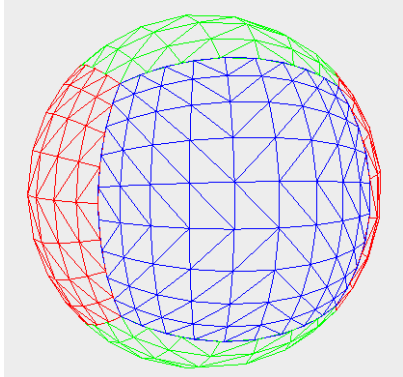
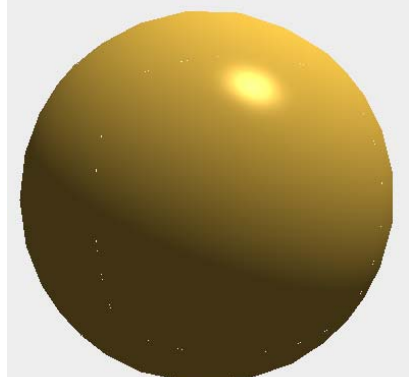
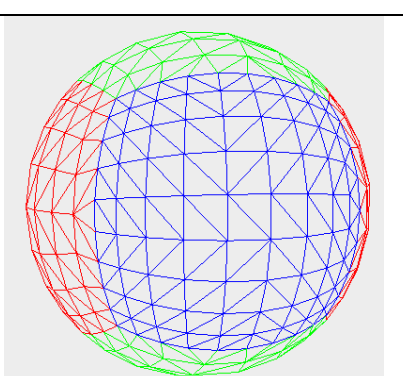
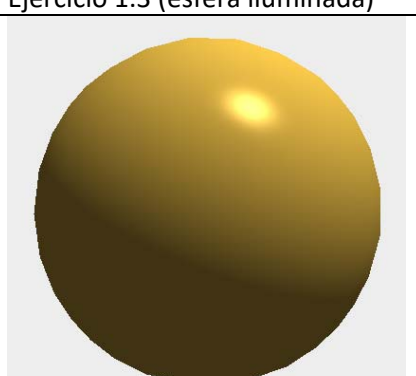
### 1.4 - Esfera con nivel de teselación variable, sin grietas

Para corregir el problema de las grietas (que aparecen en el lado común entre dos parches vecinos con diferente nivel de teselación), debemos calcular de forma independiente (al nivel de teselación del interior del parche) el nivel de teselación de cada lado del parche (fijando éste en función de la distancia del centro del lado al punto de vista).

Nota: Esta opción se activará cuando el usuario pulse de nuevo la tecla 'T'.



## Resultados

	
Ejercicio 1.1 (esfera coloreada)	Ejercicio 1.1 (esfera iluminada)
	
Ejercicio 1.2 (modo alambre)	Ejercicio 1.2 (esfera iluminada)
	
Ejercicio 1.3 (modo alambre)	Ejercicio 1.3 (esfera iluminada)
	
Ejercicio 1.4 (modo alambre)	Ejercicio 1.4 (esfera iluminada)