Informática Gráfica y Visualización
Prácticas
Práctica Nº 3.A. – Representación de Modelos: Mallas de Triángulos

Objetivos de la práctica:

- Creación y visualización de mallas de triángulos en OpenGL
- Utilización de estructuras eficientes para la visualización de mallas de triángulos, como los arrays de vértices y de normales de OpenGL

Sesiones de laboratorio: **dos sesiones**
Evaluación: **0.6 puntos sobre 10**

Material a entregar: fichero **pr3a.zip** con los **ficheros .cpp y .h** proporcionados en la plantilla, modificados tal y como se pide en los diferentes apartados de la práctica

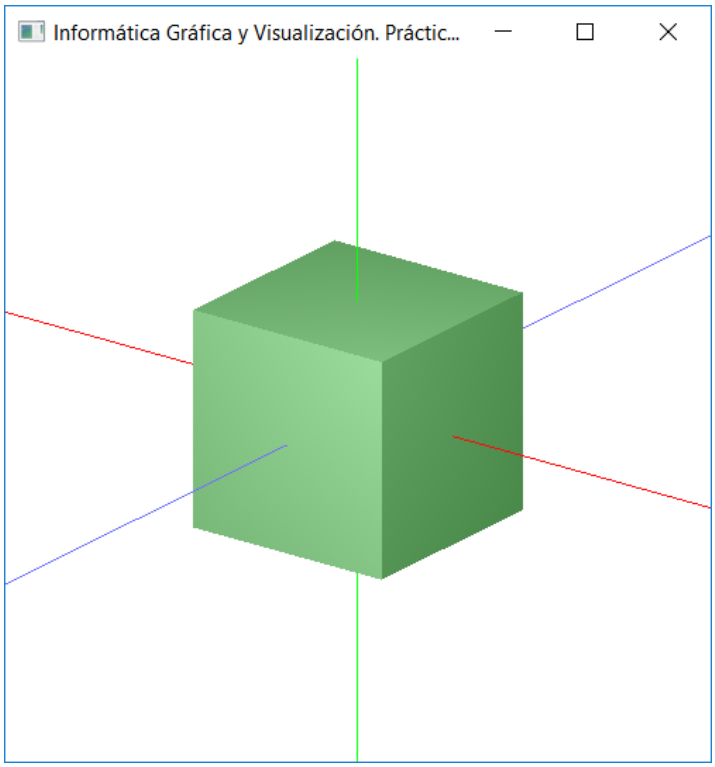
Recuerda llevar al aula lápiz y papel para dibujar geometría y hacer cálculos

PRÁCTICA Nº 3.A

Para la realización de la práctica partimos del código:

- **Pr3a.cpp**: función main() del programa.
- **igvInterfaz.h** e **igvInterfaz.cpp**: especificación e implementación de la clase ***igvInterfaz***, que contiene la funcionalidad básica para crear una ventana de visualización, su configuración y la gestión de los eventos del sistema.
- **igvEscena3D.h e igvEscena3D.cpp**: especificación e implementación de la clase ***igvEscena3D***, que contiene la funcionalidad básica para visualizar una escena.
- **igvPunto3D.h e igvPunto3D.cpp**: especificación e implementación de la clase ***igvPunto3D***, que contiene la funcionalidad para declarar y utilizar objetos de tipo punto y vector.
- **igvCamara.h e igvCamara.cpp**: especificación e implementación de la clase ***igvCamara***, que contiene la funcionalidad básica para crear y manipular cámaras de visión en la aplicación.
- **igvMallaTriangulos.h** e **igvMallaTriangulos.cpp**: especificación e implementación inicial de la clase ***igvMallaTriangulos***, con la funcionalidad básica para crear y visualizar una malla de triángulos.

Si ejecutamos el programa sin realizar ninguna modificación se abre una ventana mostrando lo siguiente:



Se visualiza un cubo de lado 1 centrado en el origen generado con la función `glutSolidCube(1);`

A) (**0.2 puntos**) Construir y visualizar un modelo equivalente al cubo anterior utilizando una malla de triángulos, **visualizando los triángulos uno a uno** (debe ser gen®rico e independiente del modelo):

- Implementa el constructor y el destructor de la clase `igvMallaTriangulos`:
  - `igvMallaTriangulos::igvMallaTriangulos(long int _num_vertices, double *_vertices, long int _num_triangulos, long int *_triangulos);`:
    Devuelve un objeto de la clase `igvMallaTriangulos` conteniendo la descripción de la malla mediante dos arrays, el primero, `_vertices`, almacena los `_num_vertices` vértices de la malla (cada vértice descrito mediante 3 coordenadas `float`); y el segundo, `_triangulos`, almacena los índices (posiciones en el array `_vertices` de los 3 vértices de cada triángulo) para construir la malla de `_num_triangulos` triángulos.

    - `igvMallaTriangulos::~igvMallaTriangulos();`:
    Destructor que libera la memoria asignada a los arrays utilizados para almacenar la malla de triángulos.

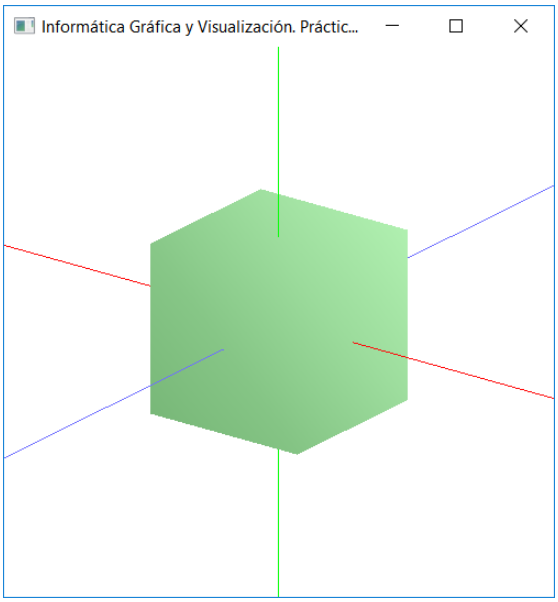
- Implementa el método `void igvMallaTriangulos::visualizar(void)` para visualizar un objeto de la clase `igvMallaTriangulos`:
  - se deberán visualizar **individualmente** los triángulos de la malla utilizando llamadas `glBegin(GL_TRIANGLES); … glEnd();`

- Añade el código necesario al método `igvEscena3D::igvEscena3D()` para crear la malla de triángulos `malla` asociada al objeto `escena` de la clase `igvInterfaz`. Esta malla se debe corresponder con el cubo generado por la función `glutSolidCube(1)`.

- Sustituye la llamada a `glutSolidCube(1)` en el método `igvEscena3D::visualizar()` por la llamada correspondiente al método `igvMallaTriangulos::visualizar()` del objeto `igvMallaTriangulos` creado.

- No es necesario considerar las normales a los triángulos ni a los vértices de momento, se añadirán más adelante en esta práctica.

La visualización que se debe obtener es la mostrada en la siguiente figura:



B) (**0.1 puntos**) Visualizar la malla de triángulos utilizando un **array de vértices de OpenGL**:

- Modifica el método `igvMallaTriangulos::visualizar()` implementado en el apartado anterior para que la visualización de la malla se realice utilizando un array de vértices de OpenGL, no requiriendo por tanto llamadas `glBegin(GL_TRIANGLES); … glEnd();`
- La visualización que se debe obtener es la misma mostrada para el apartado A.

C) (**0.1 puntos**) Para proporcionar una interacción básica con el modelo visualizado, implementar la rotación del modelo utilizando el teclado:

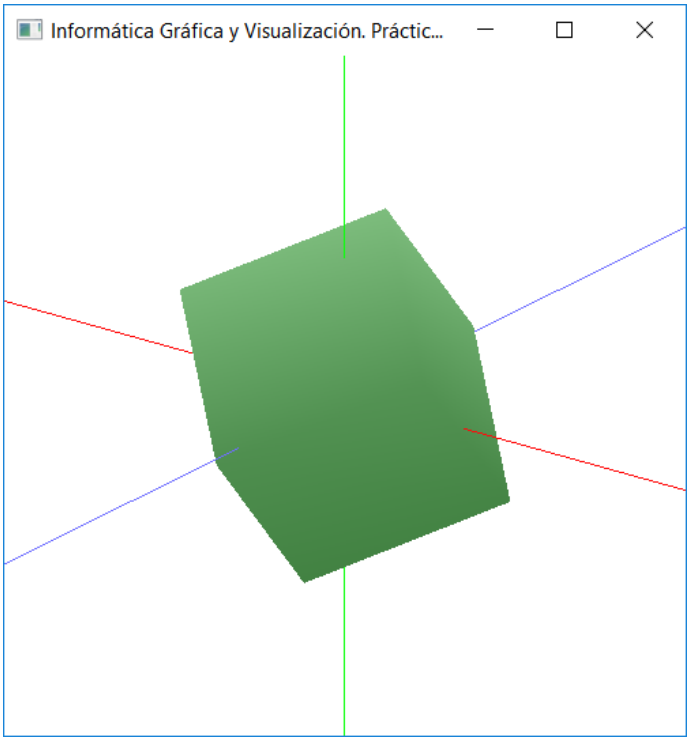
- Añadir tres atributos a la clase `igvMallaTriangulos` para guardar el ángulo de rotación en X, Y y Z que se le va a aplicar a la malla. Por defecto estos atributos tienen que valer 0.

- Añade los métodos correspondientes a la clase para que esos valores se incrementen en un ángulo pasado por parámetro, por ejemplo para la rotación en X: `igvMallaTriangulos::rotarX(float angulo)`. Añade también a la clase los métodos para recuperar los valores de cada ángulo de rotación.

- Añade las transformaciones correspondientes en el método `igvEscena3D::visualizar()` para que las rotaciones respecto a los ángulos almacenados se apliquen a la malla.

- Modifica el código del método `igvInterfaz::set_glutKeyboardFunc()` para que cuando se pulse la tecla **'x'** se incremente en 10 grados la rotación respecto al eje X y cuando se pulse la tecla **'X'** se disminuya en 10 grados (lo equivalente para **'y'**/**'Y'** y **'z'**/**'Z'**)

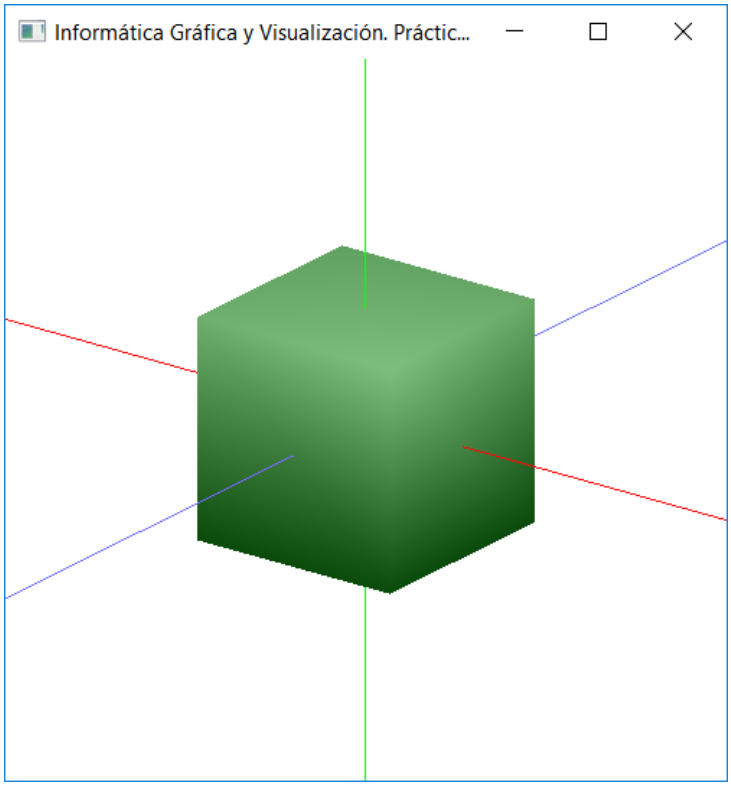
Partiendo del modelo inicialmente visualizado, si se pulsa 4 veces la tecla **'x'** y posteriormente 4 veces la tecla **'y'** se debe obtener la siguiente visualización:



D) (**0.2 puntos**) Para mejorar la visualización de la malla, hay que añadir la normal a cada vértice:

- Añade al código del constructor `igvMallaTriangulos::igvMallaTriangulos()`, implementado en el apartado A, la asignación de las normales. Para ello, hay que calcular la normal a cada triángulo de la malla y asignar esa normal a los tres vértices del triángulo. Ten en cuenta que para los v®rtices compartidos por m§s de un tri§ngulo habr²a que promediar las normales de dichos tri§ngulos. Como no tenemos informaci· n sobre los tri§ngulos que inciden en un v®rtice, cada vez que incluimos la normal en un v®rtice podemos sumar los valores promediando por 1/3 del valor calculado en cada uno de los tri§ngulos.

- Modifica el código del método `igvMallaTriangulos::visualizar()` para visualizar la malla utilizando la visualización **con array de vértices de OpenGL**, añadiéndole el array de normales de OpenGL. La visualización obtenida debe coincidir con la que se muestra en la siguiente figura:



E) Para facilitar la ejecución de todos los apartados de la práctica, añade el código necesario para que al pulsar las teclas que se indican a continuación se visualice el resultado del apartado correspondiente:

Tecla 1 – apartado A
Tecla 2 – apartado B (cambia el color del cubo para distinguir del apartado A)
Tecla 3 – apartado D