Pontificia Universidad Javeriana
Departamento de Ingeniería de Sistemas
Estructuras de Datos
Proyecto del curso, 2024-30

1 Descripción del problema

Una malla (o retícula) poligonal, en el contexto de computación gráfica es un conjunto de vértices, aristas y caras que permiten definir digitalmente la forma de un objeto tridimensional. Los vértices corresponden a puntos en el espacio 3D, es decir, coordenadas con valores reales ($x, y, z \in \mathbb{R}$). Las aristas permiten conectar pares de vértices, representadas como líneas entre los puntos. Finalmente, las caras agrupan series de vértices (y por consiguiente aristas) que conforman un polígono cerrado sencillo (usualmente un triángulo o un cuadrilátero, pero otro tipo de polígonos puede utilizarse también). La figura a continuación muestra un ejemplo de la representación de un cubo a partir de estos elementos constitutivos:

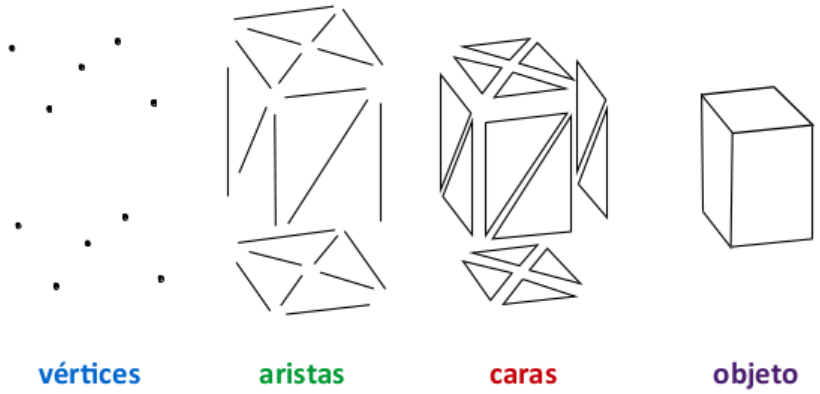


Figure 1: Representación computacional de un objeto 3D (cubo) a partir de vértices, aristas y caras. (Adaptado de `https://en.wikipedia.org/wiki/Polygon_mesh#/media/File:Mesh_overview.svg`)

La combinación de esta información (vértices, aristas y caras) permite describir de forma computacional diferentes estructuras tridimensionales al nivel de detalle que se requiera, para su posterior manipulación con algoritmos de análisis y procesamiento de mallas. De esta forma, las mallas poligonales constituyen la principal estructura de información en aplicaciones como simulación, videojuegos, animación, realidad virtual y aumentada, sistemas CAD (Computer-Aided Design) y muchas otras.

Diferentes operaciones pueden realizarse sobre la malla poligonal, desde consultas sencillas relacionadas con la ubicación de los elementos y sus relaciones locales, hasta transformaciones globales que permiten cambiar formas o generar el paso a paso de animaciones. Para que estas operaciones puedan ser eficientes, la información de vértices, aristas y caras debe estructurarse y almacenarse de forma que facilite la inserción, búsqueda y eliminación de los datos de cada objeto 3D.

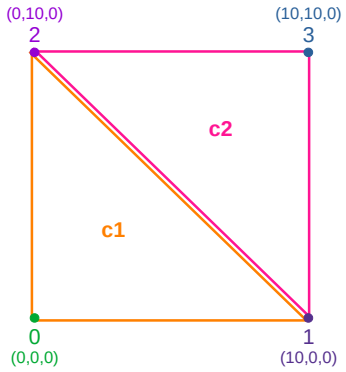
1.1 Descripción de la información de entrada

La información básica de las mallas, que representan diferentes objetos 3D, se encuentra almacenada en un archivo de texto con la siguiente estructura:

```
mesh_name
n
x0 y0 z0
...
xn-1 yn-1 zn-1
c1 i1_1 i1_2 ... i1_c
...
cm im_1 im_2 ... im_c
-1
```

- *mesh_name* es una cadena de caracteres que representa el nombre de la malla u objeto 3D (sin espacios).
- n es la cantidad de puntos (vértices) que definen la malla.
- xi, yi y zi son las coordenadas reales del i-ésimo punto (vértice).
- cj es el tamaño (cantidad de puntos o vértices) de la j-ésima cara.
- ij_k es el índice del k-ésimo punto (vértice) que conforma la j-ésima cara. Note que los puntos (vértices) están indexados desde `0` hasta `n-1`, en el orden de aparición en el archivo.
- `-1` indica el final de la información de la malla.

A continuación se presenta un ejemplo de malla (dos triángulos), y al lado la información correspondiente en el archivo de texto:




```
Mesh_0
4
0 0 0
10 0 0
0 10 0
10 10 0
3 0 1 2
3 1 3 2
-1
```

2 Descripción del proyecto

El objetivo del presente proyecto es construir un sistema que permita algunas manipulaciones sencillas sobre archivos que representan objetos 3D a partir de mallas poligonales. El sistema se implementará como una aplicación que recibe comandos textuales, agrupados en componentes con funcionalidades específicas. A continuación se describen los componentes individuales que conforman el proyecto.

2.1 Componente 1: organización de la información.

Objetivo: Los algoritmos implementados en este componente servirán para gestionar la información básica de los objetos 3D, a partir de los archivos de mallas. Este componente se implementará con las siguientes funciones:

- **comando:** `cargar` *`nombre_archivo`*
  posibles salidas en pantalla:
  (Archivo vacío o incompleto) `El archivo` *`nombre_archivo`* `no contiene un objeto 3D valido.`
  (Archivo no existe) `El archivo` *`nombre_archivo`* `no existe o es ilegible.`
  (Objeto ya existe) `El objeto` *`nombre_objeto`* `ya ha sido cargado en memoria.`
  (Resultado exitoso) `El objeto` *`nombre_objeto`* `ha sido cargado exitosamente desde el archivo` *`nombre_archivo`*`.`
  descripción: Carga en memoria la información del objeto *`nombre_objeto`* contenida en el archivo identificado por *`nombre_archivo`*. El comando debe estructurar la información a partir del archivo de forma que sea fácil recuperar los datos posteriormente.

- **comando:** `listado`
  posibles salidas en pantalla:
  (Memoria vacía) `Ningun objeto ha sido cargado en memoria.`
  (Resultado exitoso) `Hay` *`n`* `objetos en memoria:`
  `nombre_objeto_1` `contiene` *`n_1`* `vertices,` *`a_1`* `aristas y` *`c_1`* `caras.`
  `nombre_objeto_2` `contiene` *`n_2`* `vertices,` *`a_2`* `aristas y` *`c_2`* `caras.`
  `...`
  descripción: Lista los objetos cargados actualmente en memoria, junto con la información básica de cada uno: cantidad de puntos, de aristas y de caras.

- **comando:** `envolvente` *`nombre_objeto`*
  salida en pantalla:
  (Objeto no existe) `El objeto` *`nombre_objeto`* `no ha sido cargado en memoria.`
  (Resultado exitoso) `La caja envolvente del objeto` *`nombre_objeto`* `se ha generado con el nombre env_`*`nombre_objeto`* `y se ha agregado a los objetos en memoria.`
  descripción: Calcula la caja envolvente del objeto identificado por *`nombre_objeto`*. Esta caja envolvente se define a partir de dos puntos, $\mathbf{p}_{min}$ y $\mathbf{p}_{max}$, los cuales determinan los límites de una especie de cuarto donde cabe el objeto completo, es decir, todos los vértices del objeto se encuentran contenidos dentro de la caja (ver figura a continuación). $\mathbf{p}_{min}$ y $\mathbf{p}_{max}$ se calculan como los puntos extremos, en cada dimensión, del conjunto de vértices que define el objeto. La caja envolvente se agrega como un nuevo objeto en memoria, asignándole automáticamente el nombre env_*`nombre_objeto`*, para distinguirla de los demás objetos en memoria.

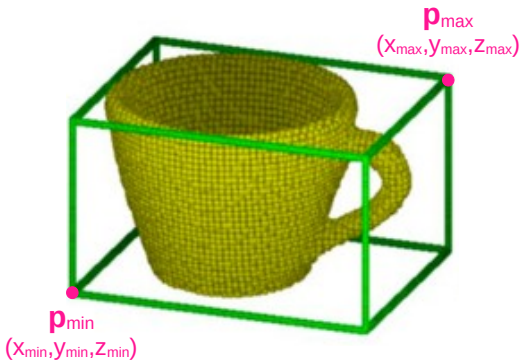


Figure 2: Representación de la caja envolvente de un objeto 3D, a partir de los puntos extremos $\mathbf{p}_{min}$ y $\mathbf{p}_{max}$

- **comando:** `envolvente`
  salida en pantalla:
  (Memoria vacía) `Ningun objeto ha sido cargado en memoria.`
  (Resultado exitoso) `La caja envolvente de los objetos en memoria se ha generado con el nombre` *`env_global`* `y se ha agregado a los objetos en memoria.`
  descripción: Calcula la caja envolvente que incluye a todos los objetos cargados actualmente en memoria. Esta caja envolvente se calcula de la misma forma que en el comando anterior, sólo que teniendo en cuenta todos los vértices de todos los objetos en memoria, para no dejar ninguno por fuera de la caja. La caja envolvente se agrega como un nuevo objeto en memoria, asignándole automáticamente el nombre *`env_global`*, para distinguirla de los demás objetos en memoria.

- **comando:** `descargar` *`nombre_objeto`*
  posibles salidas en pantalla:
  (Objeto no existe) `El objeto` *`nombre_objeto`* `no ha sido cargado en memoria.`
  (Resultado exitoso) `El objeto` *`nombre_objeto`* `ha sido eliminado de la memoria de trabajo.`
  descripción: Descarga de la memoria toda la información básica (vértices, aristas, caras) del objeto identificado por *`nombre_objeto`*.

- **comando:** `guardar` *`nombre_objeto nombre_archivo`*
  posibles salidas en pantalla:
  (Objeto no existe) `El objeto` *`nombre_objeto`* `no ha sido cargado en memoria.`
  (Resultado exitoso) `La información del objeto` *`nombre_objeto`* `ha sido guardada exitosamente en el archivo` *`nombre_archivo`*`.`
  descripción: Escribe en un archivo de texto, identificado por *`nombre_archivo`*, la información básica (vértices y caras) del objeto identificado por *`nombre_objeto`*. El archivo debe seguir el formato presentado en la Subsección 1.1.

- **comando:** `salir`
  posibles salidas en pantalla:
  (No tiene salida por pantalla)
  descripción: Termina la ejecución de la aplicación.

2.2 Componente 2: vértices más cercanos.

Objetivo: Los algoritmos implementados en este componente servirán para identificar puntos (vértices) más cercanos en los objetos. Este componente se implementará con las siguientes funciones:

- **comando:** `v_cercano` *`px py pz nombre_objeto`*
  posibles salidas en pantalla:
  (Objeto no existe) `El objeto` *`nombre_objeto`* `no ha sido cargado en memoria.`
  (Resultado exitoso) `El vertice` *`i`* `(`*`vx`*`,`*`vy`*`,`*`vz`*`) del objeto` *`nombre_objeto`* `es el más cercano al punto (`*`px`*`,`*`py`*`,`*`pz`*`), a una distancia de` *`valor_distancia`*`.`
  descripción: Identifica el vértice del objeto *`nombre_objeto`* más cercano (en términos de la distancia euclidiana) al punto indicado por las coordenadas *`px`*, *`py`* y *`pz`*. Informa en pantalla el índice del vértice, los valores actuales de sus coordenadas, y la distancia a la cual se encuentra del punto dado.

- **comando:** `v_cercano` *`px py pz`*
  posibles salidas en pantalla:
  (Memoria vacía) `Ningun objeto ha sido cargado en memoria.`
  (Resultado exitoso) `El vertice` *`i`* `(`*`vx`*`,`*`vy`*`,`*`vz`*`) del objeto` *`nombre_objeto`* `es el más cercano`

al punto (*px*,*py*,*pz*), a una distancia de *valor_distancia*.
descripción: Identifica, entre todos los objetos cargados en memoria, el vértice más cercano (en términos de la distancia euclidiana) al punto indicado por las coordenadas *px*, *py* y *pz*. Informa en pantalla el objeto al que pertenece, el índice del vértice, los valores actuales de sus coordenadas, y la distancia a la cual se encuentra del punto dado.

- **comando:** v_cercanos_caja *nombre_objeto*
  posibles salidas en pantalla:
  (Objeto no existe) El objeto *nombre_objeto* no ha sido cargado en memoria.
  (Resultado exitoso) Los vertices del objeto *nombre_objeto* más cercanos a las esquinas de su caja envolvente son:

Esquina	Vertice	Distancia
1 (*e1x*,*e1y*,*e1z*)	*i1* (*v1x*,*v1y*,*v1z*)	*valor_distancia_1*
2 (*e2x*,*e2y*,*e2z*)	*i2* (*v2x*,*v2y*,*v2z*)	*valor_distancia_2*
...		
8 (*e8x*,*e8y*,*e8z*)	*i8* (*v8x*,*v8y*,*v8z*)	*valor_distancia_8*

  descripción: Identifica los vértices del objeto *nombre_objeto* más cercanos (en términos de la distancia euclidiana) a los puntos (vértices) que definen la respectiva caja envolvente del objeto. Informa en pantalla, en una tabla, las coordenadas de cada una de las esquinas de la caja envolvente, y para cada una de ellas, el índice del vértice más cercano, los valores actuales de sus coordenadas, y la distancia a la cual se encuentra de la respectiva esquina.

2.3 Componente 3: rutas más cortas.

Objetivo: Los algoritmos implementados en este componente servirán para identificar rutas más cortas que conectan diferentes vértices dentro de los objetos. Este componente se implementará con las siguientes funciones:

- **comando:** ruta_corta *i1 i2 nombre_objeto*
  salida en pantalla:
  (Objeto no existe) El objeto *nombre_objeto* no ha sido cargado en memoria.
  (Índices iguales) Los indices de los vertices dados son iguales.
  (Índices no existen) Algunos de los indices de vertices estan fuera de rango para el objeto *nombre_objeto*.
  (Resultado exitoso) La ruta más corta que conecta los vertices *i1* y *i2* del objeto *nombre_objeto* pasa por: *i1*,*v1*,*v2*,...,*vn*,*i2*; con una longitud de *valor_distancia*.
  descripción: Identifica los índices de los vértices que conforman la ruta más corta entre los vértices dados para el objeto *nombre_objeto*. La distancia entre los vértices está medida en términos de la distancia euclidiana. Informa, además, la distancia total de la ruta calculada.

- **comando:** ruta_corta_centro *i1 nombre_objeto*
  salida en pantalla:
  (Objeto no existe) El objeto *nombre_objeto* no ha sido cargado en memoria.
  (Índice no existe) El indice de vertice esta fuera de rango para el objeto *nombre_objeto*.
  (Resultado exitoso) La ruta más corta que conecta el vertice *i1* con el centro del objeto *nombre_objeto* , ubicado en *ct (ctx, cty, ctz)*, pasa por: *i1*,*v1*,*v2*,...,*ct*; con una longitud de *valor_distancia*.
  descripción: Identifica los índices de los vértices que conforman la ruta más corta entre el vértice dado y el centro del objeto *nombre_objeto*. El vértice centro del objeto se identifica calculando el centroide (coordenadas promedio) de todos los vértices del objeto, este punto es agregado entonces

a la representación del objeto. Luego, se busca el vértice del objeto más cercano a ese centroide, y se conecta con el centroide por medio de una arista. Finalmente, se utiliza el mismo proceso del comando anterior para buscar la ruta más corta entre el centroide (ya conectado dentro del objeto) y el vértice dado. La distancia entre los vértices está medida en términos de la distancia euclidiana. Informa, además, la distancia total de la ruta calculada.

2.4 Interacción con el sistema

La interfaz de la aplicación a construir debe ser una consola interactiva donde los comandos correspondientes a los componentes serán tecleados por el usuario, de la misma forma que se trabaja en la terminal o consola del sistema operativo. El indicador de línea de comando debe ser el caracter $. Se debe incluir el comando `ayuda` para indicar una lista de los comandos disponibles en el momento. Así mismo, para cada comando se debe incluir una ayuda de uso que indique la forma correcta de hacer el llamado, es decir, el comando `ayuda` *`comando`* debe existir.

Cada comando debe presentar en pantalla los mensajes de resultado (éxito o error) especificados antes, además de otros mensajes necesarios que permitan al usuario saber, por un lado, cuando terminó el comando su procesamiento, y por el otro lado, el resultado de ese procesamiento. Los comandos de los diferentes componentes deben ensamblarse en un único sistema (es decir, funcionan todos dentro de un mismo programa, no como programas independientes por componentes).

3 Evaluación

Las entregas se harán en la correspondiente actividad de BrightSpace, hasta la media noche del día anterior al indicado para la sustentación de la entrega. Se debe entregar un archivo comprimido (único formato aceptado: .zip) que contenga dentro de un mismo directorio (**sin estructura de carpetas interna**) los documentos (único formato aceptado: .pdf) y el código fuente (.h, .hxx, .cxx, .cpp). Si la entrega contiene archivos en cualquier otro formato, será descartada y no será evaluada, es decir, la nota definitiva de la entrega será de **0 (cero) sobre 5 (cinco)**.

3.1 Entrega 0: semana 3

La entrega inicial corresponderá únicamente a la interfaz de usuario necesaria para interactuar con el sistema. De esta forma, se verificará el indicador de línea del comando, y que el sistema realice la validación de los comandos permitidos y sus parámetros. (Revisar en particular el numeral 2.4).

3.2 Entrega 1: semana 6

Componente 1 completo y funcional. Esta entrega tendrá una sustentación durante la correspondiente sesión de clase, y se compone de:

- (40%) Documento de diseño. El documento de diseño debe seguir las pautas de ingeniería que usted ya conoce: descripción de entradas, salidas y condiciones para el procedimiento principal y las operaciones auxiliares (comandos). Para la descripción de los TADs utilizados, debe seguirse la plantilla definida en clase. Además, se exigirán esquemáticos (diagramas, gráficos, dibujos) que describan el funcionamiento general de las operaciones (comandos) principales.
- (20%) Plan de pruebas. Adjuntar al documento de diseño un plan de pruebas, que siga las pautas vistas en clase, para el comando `envolvente`.
- (30%) Código fuente compilable en el compilador `gnu-g++` (versión 4.0.0, como mínimo). Este porcentaje de la entrega será un promedio de la evaluación de cada comando.

- (10%) Sustentación (individual) con participación de todos los miembros del grupo.

3.3 Entrega 2: semana 12

Componentes 1 y 2 completos y funcionales. Esta entrega tendrá una sustentación durante la correspondiente sesión de clase, y se compone de:

- (10%) Completar la funcionalidad que aún no haya sido desarrollada de la primera entrega. Se debe generar un acta de evaluación de la entrega anterior (incluirla al principio del documento de diseño) que detalle los comentarios textuales (literales) hechos a la entrega y la forma en la que se corrigieron, arreglaron o completaron para la segunda entrega.
- (30%) Documento de diseño. El documento de diseño debe seguir las pautas de ingeniería que usted ya conoce: descripción de entradas, salidas y condiciones para el procedimiento principal y las operaciones auxiliares (comandos). Para la descripción de los TADs utilizados, debe seguirse la plantilla definida en clase. Además, se exigirán esquemáticos (diagramas, gráficos, dibujos) que describan el funcionamiento general de las operaciones (comandos) principales.
- (20%) Plan de pruebas. Adjuntar al documento de diseño un plan de pruebas, que siga las pautas vistas en clase, para el comando `v_cercanos_caja`.
- (30%) Código fuente compilable en el compilador `gnu-g++` (versión 4.0.0, como mínimo). Este porcentaje de la entrega será un promedio de la evaluación de cada comando.
- (10%) Sustentación (individual) con participación de todos los miembros del grupo.

3.4 Entrega 3: semana 18

Componentes 1, 2 y 3 completos y funcionales. Esta entrega tendrá una sustentación (del proyecto completo) entre las 8am y las 12m del último día de clase de la semana 18, y se compone de:

- (10%) Completar la funcionalidad que aún no haya sido desarrollada de la primera y segunda entregas. SSe debe generar un acta de evaluación de las entregas anteriores (incluirla al principio del documento de diseño) que detalle los comentarios textuales (literales) hechos a las entregas y la forma en la que se corrigieron, arreglaron o completaron para la tercera entrega.
- (30%) Documento de diseño. EEl documento de diseño debe seguir las pautas de ingeniería que usted ya conoce: descripción de entradas, salidas y condiciones para el procedimiento principal y las operaciones auxiliares (comandos). Para la descripción de los TADs utilizados, debe seguirse la plantilla definida en clase. Además, se exigirán esquemáticos (diagramas, gráficos, dibujos) que describan el funcionamiento general de las operaciones (comandos) principales.
- (20%) Plan de pruebas. Adjuntar al documento de diseño un plan de pruebas, que siga las pautas vistas en clase, para el comando `ruta_corta`.
- (30%) Código fuente compilable en el compilador `gnu-g++` (versión 4.0.0, como mínimo). Este porcentaje de la entrega será un promedio de la evaluación de cada comando.
- (10%) Sustentación (individual) con participación de todos los miembros del grupo.