Pontificia Universidad Javeriana
Facultad de ingeniería
Estructuras de datos
Alejandro Mayorga
Diego Guevara
Documento entrega 1

En el presente informe se realizara la descripción de el sistema para generar polígonos a partir de archivos de texto y su posterior manipulación.

TADS:

Vértice:
Este describe los puntos donde 2 o más líneas se unen en un polígono siendo así un punto en 2 o 3 dimensiones.

variables:
x, flotante, describe la coordenada en el plano x del vértice (ancho).
y, flotante, describe la coordenada en el plano y del vértice (largo).
z, flotante, describe la coordenada en el plano z del vértice (alto).

Cara:
Esta describe un superficie entre aristas(línea entre 2 vértices) que denota alguna cara del polígono

variables:
tamaño cara, float, describe las dimensiones de superficie de la cara (a*b).
vértice j esimo, tipo vértice, es el vértice j esimo de la cara.

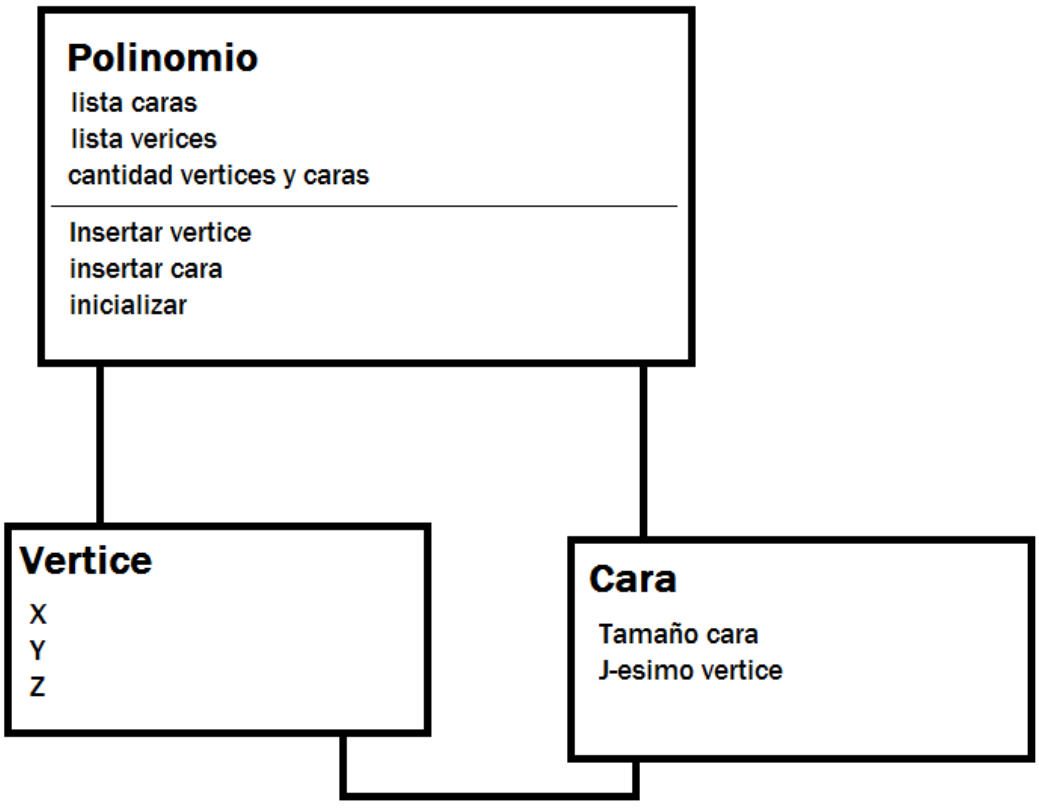
Polígono:
Es la unión de varias caras en un espacio 3D para formar una figura.

Variables:
nombre, string, es el identificador único de cada polígono.
cantidad de vértices, entero, es la cantidad de vértices que el polígono contiene.
Lista de Vértices, es una lista de cada vértice con sus respectivas propiedades
Lista de Caras, es una lista con las respectivas caras del polígono y sus valores
Cantidad de Caras, entero, es el número de caras contenidas por el polígono

Funciones:

(se asumen los gets y sets para cada variable del polígono)
inicializarPoligono(), inicializa el polígono con valores neutros para evitar el manejo de basura
insertarVertice(vértice val), agrega un vértice a la lista de ves del polígono.
insertarCara(cara val), agrega una cara la lista de caras del polígono.

Diagrama de relación:



Descripción del funcionamiento general:

Para presentar esta primera entrega se decidió hacer una interfaz con las distintas opciones que puede tomar el usuario y a continuación se abordara cada una y como fueron implementadas.

1. Cargar polígono a memoria
2. Listar polígonos en memoria
3. Envolver objeto
4. Envolver objetos en memoria
5. Descargar memoria
6. Guardar objetos
7. Terminar programa

Cargar en memoria:
Función para crear un nuevo polígono en base a los datos en un archivo de texto.

Representación en código:
poligono cargarPoligono(string val, list<poligono> &poligonosMemoria);

Entradas: una variable de tipo String que denota el nombre del archivo del cual se extraerán los datos. Con este valor se buscara la existencia o no de dicho archivo.

Salidas: Un polígono creado con toda la información en el archivo de texto.

Pasos a seguir:

1. Se abre el archivo con el nombre dado en val
2. Se consigue la primera y segunda línea de dicho archivo
3. La primera línea contiene el nombre de la malla y se guarda en el nombre de un polígono auxiliar (el de salida) y la segunda como la cantidad de vértices que este contiene (también guardado en el polígono auxiliar)
4. A continuación, como ya se sabe que siguen los vértices se tomara la línea siguiente del archivo y empezara a buscar los valores antes de los espacios.
5. Al primer valor antes de un espacio lo asignara a la variable x de un vértice auxiliar
6. A partir de donde se encontró el primer espacio, buscara el numero antes del segundo para formar la variable Y y agregarla al vértice auxiliar en su coordenada correspondiente.
7. Finalmente a partir de donde estaba el segundo espacio, se buscara el 3er número que denote la variable Z.
8. Una vez conseguido el trio de coordenadas en el vértice auxiliar, se agregara al polígono auxiliar.
9. Este proceso se repite hasta que el número de tríos armados y agregados al polígono, sea igual al cantidad de veticas recolectada en el punto 3.

10. Nota: el procedimiento está hecho para hacer estos tríos de valores incluso cuando en una línea se tienen más de 3 números
11. Una vez llenados los vértices se recorrerán las líneas restantes del archivo para formar las caras
12. El primer número en la línea corresponderá al tamaño de la cara
13. De nuevo como en el paso anterior, se formaran tríos con los números restantes para formar un vértice y agregarlo al vértice de una cara auxiliar
14. Una vez obtenidos el vértice y el tamaño de la cara auxiliar, se agregaran a la lista de caras del polinomio auxiliar (el mismo de salida).
15. Este proceso se repetirá de formar caras se repetirá hasta que la línea extraída del archivo de texto sea un -1. Ahí se para y se cierra el stream al archivo.

Listar polígonos:
Lista cada polígono en memoria con sus propiedades

Representación en código:
void listapoligonos(list<poligono> &poligonosMemoria).

Entradas: poligonosMemoria , apuntador a las lista de polígonos que se desea enlistar
Salidas: ninguna.

Pasos:
1. Revisar que la lista no este vacía, en caso de que si, terminar la función.
2. Mediante un iterador, recorrer la lista y por cada valor imprimir su nombre, su cantidad de vértices, la cantidad de caras y aristas

Envolvente objeto:
Crea un envolvente sobe un polígono dado

Representación en código:
poligono envolvente(string nombreObjeto, list<poligono> listIn).

Entradas: string nombreObjeto, nombre del objeto que se quiere envolver.
list<poligono> listIn, lista que contiene los polígonos en memoria
Salidas: el envoltorio del polígono.

Pasos:
1. Revisar si existe un polígono en la listaIn con el nombre nombreObjeto.
2. Si existe se busca por todos los vértices, el máximo en la dimensión x, el mínimo en la dimensión x, el máximo en la dimensión y, el mínimo en la dimensión y, el máximo en la dimensión z y el mínimo en la dimensión z.
3. Se guarda los máximos y mínimos en 2 vértices, auxmax y auxmin.
4. Se crean 8 vértices como las 8 combinaciones de auxmax X auxmin.
5. Se agregan a la lista de vértices de un polinomio de salida.
6. Se crean las 6 caras con sus tamaños calculados y se agregan a la listas de caras del polinomio de salida.

7. Se retorna el polinomio.

Descargar polígono:
Elimina un polígono en específico de la memoria.

Representación en código:
void descargarmemoria(list<poligono> &poligonosMemoria, string nombre_objeto)

Entradas: list<poligono> &poligonosMemoria, lista de polígonos de donde se quiere borrar uno en específico. string nombre_objeto, nombre del objeto que se quiere borrar.
Salidas: ninguna

Pasos:
1. Mediante un iterador recorrer la lista de polígonos.
2. Si en algún momento, el nombre del polígono que está en la posición del iterador es igual a la entrada nombre_objeto, se borrara dicho elemento y se colocara una bandera borrado como true.
3. Al salir de la búsqueda, si la bandera borrado es verdadera, se indicara que se borró con éxito, de otro modo se indicara que no se encontró el objeto en memoria.

Guardar polígono:
Función para guardar un polígono de memoria, en un archivo determinado

Representación en código:
bool guardarPoligono(string nombreObjeto, string nombreArchivo, list<poligono> listIn)

Entradas: String nombre objeto, nombre del polígono que se quiere guardar.
String nombreArchivo, nombre del archivo donde se quiere guardar el polígono.
list<poligono> listIn, lista de polígonos para buscar la existencia del polígono que se quiere guardar.

Salidas: un booleano, dependiendo si se pudo hacer el guardado se manda verdadero, sino falso.

Pasos:
1. Revisar si existe el polinomio con el nombre de entrada en la lista mediante buscarPoligono(listIn, nombreObjeto).
2. Si existe se continua extrayéndolo en un polinomio auxiliar mediante buscarPoligono2(listIn, nombreObjeto).
3. Se abre el stream de salida.
4. Se crea un nuevo archivo de texto con el nombre dado en las entradas.

5. Se agrega el nombre del polígono y el tamaño respectivamente, cada uno en una línea diferente.
6. Se realiza un for para extraer cada vértice de la lista de vértices del polinomio auxiliar.
7. Se realiza un for para extraer cada cara de la lista de caras del polinomio auxiliar.
8. Se agrega un -1 final.
9. Se cierra el stream.

Terminar programa:
Acaba con la ejecución de la aplicación.

Funciones extra:
bool buscarPoligono(list<poligono> listIn, string nombreIn): busca la existencia de un polígono con nombre nombreIn en la listaIn, si existe retorna true, sino false.
poligono buscarPoligono2(list<poligono> listIn, string nombreIn): busca la existencia de un polígono con nombre nombreIn en la listaIn, si existe retorna lo retorna, sino, retorna un polígono NULL.

Entrega 2:

En el presente informe se realizara la descripción de la implementación de los comandos necesarios para la entrega 2 del proyecto.
Para esta entrega se realizaron 3 comandos básicos cuyo objetivo es el cálculo de distancias entre vértices con la finalidad de obtener mayor información de las mayas 3D.

Comandos:

v_cercano px py pz nombre_objeto:
Pretende calcular la distancia entre un punto dado con coordenadas XYZ y un vértice existente en un polígono cuya única restricción es que sea el más cercano al punto dado.

Representación en código:
float* componente2Punto1(string nombrePoligono, list<poligono> poligonosMemoria, float x, float y, float z, bool on);

Entradas:
px, un numero racional positivo que denota la variable X (ancho) en un espacio de 3 dimensiones.
py, un numero racional positivo que denota la variable Y (largo) en un espacio de 3 dimensiones.
pz, un numero racional positivo que denota la variable Z (alto) en un espacio de 3 dimensiones.
nombre_objeto: nombre que denota un polígono cargado previamente en memoria para ser buscado y procesado.

Salidas:
La distancia a la que se el punto dado y el vértice más cercano al polígono, sin embargo indirectamente también da información de cuáles son las coordenadas del vértice del polígono y su índice mediante impresiones por pantalla.

Paso a Paso:

1. Busca el nombre del polígono en la memoria de polígonos insertados.
2. Si no está, termina la ejecución.
3. Extrae del polígono encontrado su lista de vértices.
4. Recorre la lista de vértices.
5. Para cada elemento, extrae las coordenadas x, y yz y llama una función de cálculo de distancia euclidiana.
6. Ingresa la distancia resultante en un árbol
7. Si el parámetro ON de entrada se encuentra como true, llamara la función del árbol que retorna el valor más a la izquierda de este (ósea el mínimo) y retornara dicho valor
8. Si el parámetro On se encuentra como falso, volverá a recorrer la lista de vértices y volverá a hacer el cálculo de distancias por cada uno
9. Si la en dicho vértice la distancia euclidiana es igual, imprimirá los las coordenadas de el vértice, el nombre del objeto y acabara la ejecución de la función.

v_cercano px py pz:
Busca entre todos los objetos cargados en memoria, el que tenga el vértice de menor distancia entre un punto dado e imprimirá su información.

Representación en código:
void componente2Punto2(list<poligono> poligonosMemoria, float x, float y, float z);

Entradas:
px, un numero racional positivo que denota la variable X (ancho) en un espacio de 3 dimensiones.
py, un numero racional positivo que denota la variable Y (largo) en un espacio de 3 dimensiones.
pz, un numero racional positivo que denota la variable Z (alto) en un espacio de 3 dimensiones.

Salida: la información impresa en pantalla sobre el nombre del objeto cuyo vértice esta a menor distancia al punto dado y, las coordenadas de dicho vértice.

Paso a paso:

1. Recorre la lista de polígonos en memoria.
2. Para cada polígono, obtiene su nombre y llama la función componente2Punto1 descrita anteriormente., insertando el valor que retorne (una distancia) en un árbol binario.
3. Una vez haya recorrido toda la lista la vuelve a recorrer.
4. Para cada elemento compara el valor mínimo del árbol binario (el que está más a la izquierda) con un nuevo llamado de la función componente2Punto1 sobre cada polígono.
5. Si las distancias son iguales, imprimirá toda la información de dicho polígono y terminara la ejecución.

v_cercanos_caja nombre_objeto:
Dado el nombre de un objeto, se calculara la envolvente de dicho objeto y a partir de esta y sus 8 esquinas, se buscara los vértices del polígono que encapsula, que estén a menor distancia de las esquinas, así cada esquina, tendrá un vértice correspondiente más cercano.

Representación en código:
void componente2Punto3(string nombrePoligono,list<poligono> poligonosMemoria)

Entradas:
nombre_objeto: nombre que denota un polígono cargado previamente en memoria para ser buscado y procesado.

Salida: la información impresa en pantalla sobre el los 8 vértices que estén más cerca a las 8 esquinas de la envolvente.

Paso a paso:

1. Buscar el nombre del polígono en la lista de polígonos cargados en memoria.
2. Si se encuentra, extraer de este la correspondiente lista de vértices, sino se termina la ejecución.
3. Recorrer la lista de vértices
4. Para cada elemento de la lista, llamar la función componente2Punto1 descrita previamente.
5. Una vez recorridos todos los elementos de la lista se finaliza la ejecución.

Funciones adicional:
calcularDistancia:
Dados dos pares de coordenadas XYZ calcular la distancia euclidiana entre estas.

Representación en código:
float* calcularDistancia(float x, float y, float z, float x2, float y2, float z2)

Esta función solo aplica la fórmula de distancia euclidiana y la retorna.

Formula de distancia euclidiana para coordenadas en 3 dimensiones:

$$d(A,B)=\sqrt{|x_B-x_A|^2+|y_B-y_A|^2+|z_B-z_A|^2}$$

Equivalente en código:

```
abs(sqrt( pow((x2 - x),2) + pow((y2 - y),2) + pow((z2 - z),2)));
```