Capítulo 3.Arquitectura general y procesamiento de archivos.

Diseño general del software.

El software se encuentra dividida en 4 niveles jerárquicos y uno de acceso global.:

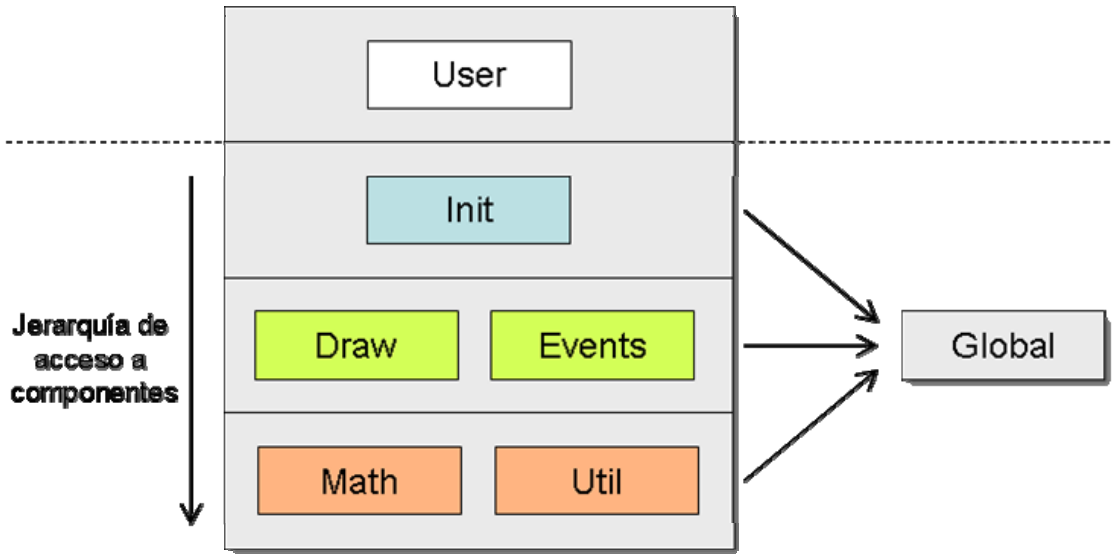


Fig 2.1.- Diseño general del software

1er. Nivel jerárquico (User).- Es el nivel jerárquico correspondiente a la capa de usuario en la cual se encuentran los únicos componentes a los cuales el usuario puede acceder. Provee al usuario un acceso indirecto a ciertos componentes de los niveles jerárquicos más bajos dándole así la funcionalidad del software de manera sencilla y segura.

Los siguientes tres niveles jerárquicos que se encuentran bajo el primer nivel así como el componente de acceso global, están restringidos al usuario por razones de seguridad ya que estos contienen el código interno referente a la implementación de cada funcionalidad del software. Estos tres niveles están ordenados de modo que un nivel puede acceder a componentes del mismo nivel o cualquier otro que este debajo de el pero no se puede acceder a componentes que estén en niveles superiores.

2º. nivel jerárquico (Init).- En éste nivel se encuentran los procedimientos que llevan a cabo la inicialización del dispositivo de video, la superficie a ser ocupada por OpenGL y la puesta en marcha de la captura de eventos. Los tres aspectos anteriores se realizan bajo la supervisión de la librería SDL. Contiene también los procedimientos que llevan a cabo las indicaciones del *programador final* las cuales son llevadas a cabo al principio de la ejecución del software.

3er. nivel jerárquico (Draw/Events).- Es en éste nivel en donde se encuentran las clases que albergan los mecanismos que modifican los parámetros de la cámara según los eventos que ocurran así cómo también las clases que se encargan de construir e interpretar la información sobre los objetos 3D para poder ser desplegados en pantalla.

4º. nivel jerárquico (Math/Util).- En este nivel se albergan componentes matemáticos y de aplicación diversa que pueden ser requeridos por componentes de los dos niveles anteriores y por componentes del mismo nivel.

Nivel de acceso global.- Almacena los elementos que pueden ser accedidos y modificados en cualquier momento por los componentes localizados en alguno de los niveles jerárquicos a excepción del nivel de usuario.

Arquitectura general del software.

A continuación se muestra un diagrama que muestra la secuencia de los procesos realizados por el software:

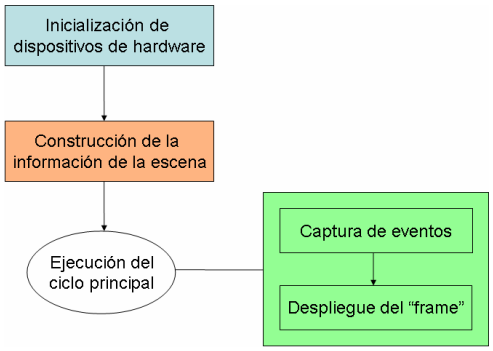


Fig. 2.1.- Diagrama de secuencia.

La arquitectura básica que fue utilizada para el diseño del software se muestra en el siguiente diagrama de clases:

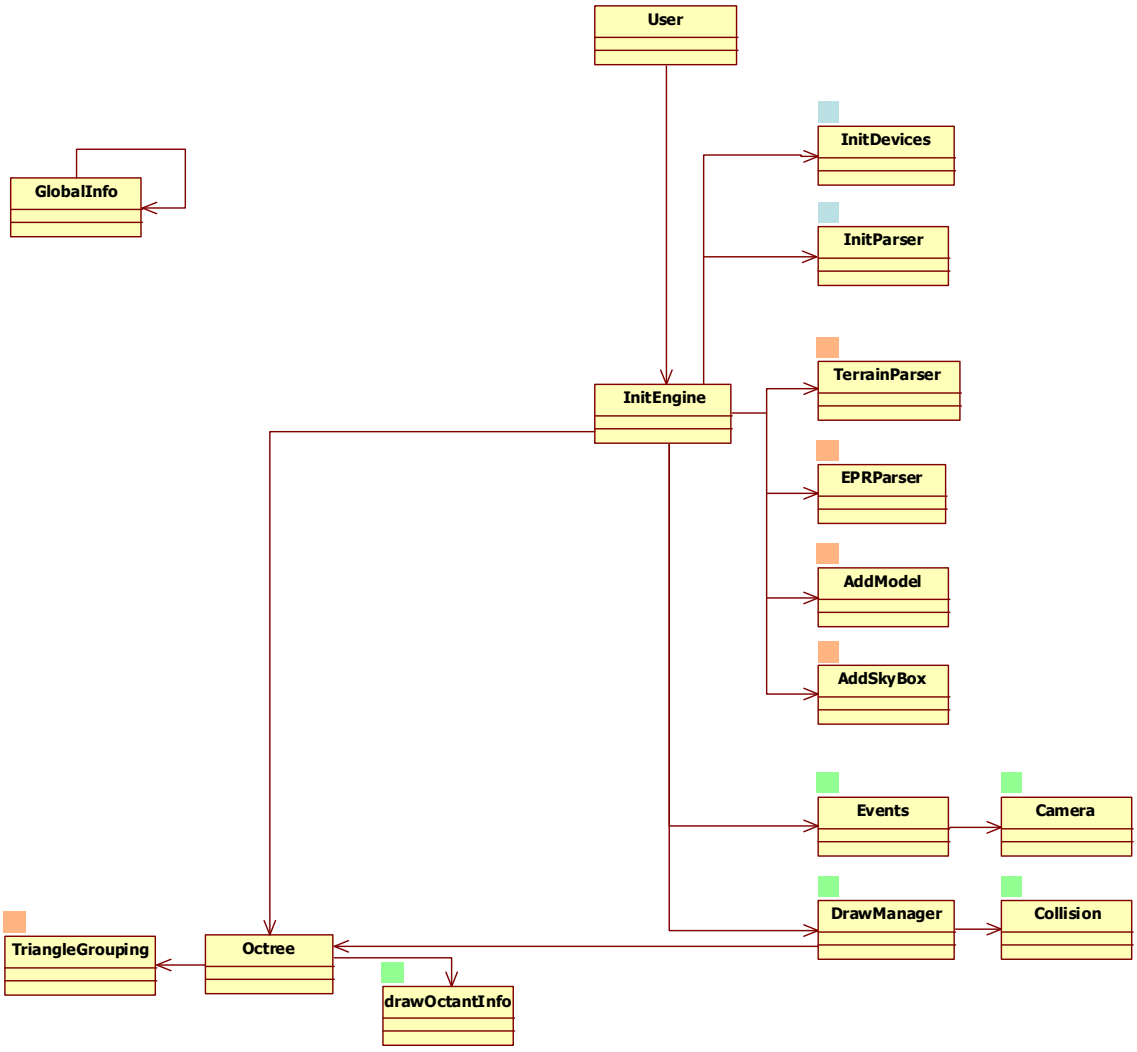


Fig. 2.2.- Diagrama de clases general.

Cada clase está marcada con un cuadro del color correspondiente a la etapa a la cual pertenece en el diagrama de secuencia. La etapa de "inicialización de dispositivos de hardware" es el primer procedimiento que se realiza al ejecutar el software.

Básicamente consiste en habilitar y configurar el hardware de video y en inicializa la captura de eventos del teclado y mouse (clase *InitDevices*). También se extrae la información correspondiente a las propiedades iniciales para la ejecución del software (posición de cámara, velocidad de movimiento de cámara, etc.), que se encuentra en un archivo de inicialización (clase *InitParser*).

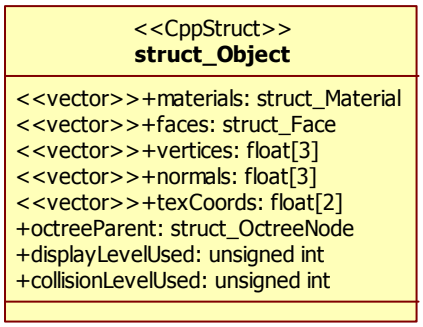
Como su nombre lo indica, en la etapa "construcción de la información de la escena" se construyen los objetos y estructuras que definen los elementos del mundo 3D. Esto se realiza con la información obtenida en base a los comandos ingresados por el usuario que en el diagrama son todas las clases marcadas con color naranja a excepción de *TriangleGrouping* que representa los procedimientos internos del software para particionar la información del mundo 3D.

En la "ejecución del ciclo principal" se llevan a cabo los procesos que se realizaran repetitivamente durante todo el ciclo de ejecución del software que básicamente son dos: 1) Captura de eventos, 2) Despliegue del "frame". En la "captura de eventos" se verifica si el usuario a dado alguna orden a travez de los dispositivos de entrada utilizados (clase *Events*). De ser así se realizan los cambios a las propiedades de cámara correspondientes (clase *Camera*). En la etapa "despliegue de un frame" (clase *DrawOctantInfo*), se procede a evaluar las regiones del mundo 3D que serán desplegadas considerando las propiedades de la cámara y la posible colisión con la geometría de la escena (clase *Collision*).

En el **Apéndice B** se muestra un diagrama de clases que describe de manera más completa la arquitectura del software.

Estructura básica de un objeto 3D.

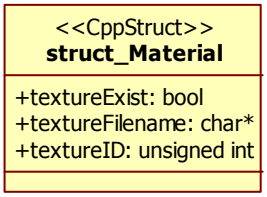
La estructura con la información básica inicial que debe tener cada objeto 3D en la escena esta definida como sigue:



Los contenedores de tipo vector utilizados en el software pertenecen a la librería estándar de C++. Cuando se requiere saber el tamaño de un contenedor de tipo vector, se devuelve un dato de tipo "*unsigned int"* el cual esta formado por 32 bits por lo cual podemos representar números comprendidos en el rango entre 0 y 4,294,967,295. Esto significa que un contenedor de tipo vector de la librería estándar de C++ puede almacenar hasta 4,294,967,295 elementos y por consiguiente una estructura de tipo "*struct_Object"* puede almacenar un máximo de 4,294,967,295 vértices, caras, normales, coordenadas de textura y materiales.

En términos generales explicaremos a continuación la función de cada uno de los atributos de la estructura anterior aunque los conceptos referentes a "*octrees"* serán explicados a profundidad en temas posteriores.

materials: Vector de estructuras tipo "*struct_Material"* que contiene las propiedades de todos los materiales utilizados para el objeto 3D. Los atributos básicos de la estructura de cada material son los siguientes:



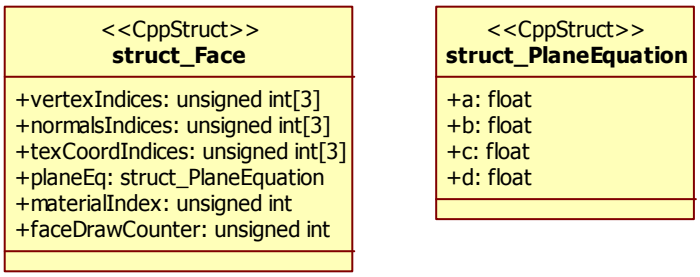
textureExist: Puede ser "true" o "false" si es que existe o no una textura asignada a este meterial.

textureFilename: Almacena el nombre y ruta del archivo con el mapa de bits correspondiente a la textura.

textureID: Cuando la textura es registrada en OpenGL, le es asignado este identificado al cual se hace referencia cada vez que se desea utilizar la textura.

Cabe mencionar que en los materiales hemos omitido el uso de características como luz ambiental, difusa, especular, emisiva y brillo del material. Uno de los objetivos específicos del proyecto consiste en garantizar al programador final un aprendizaje total de la librería en menos de un día. El correcto uso de estas características puede requerir de cierta habilidad del programador final para manejar las fuentes de luz en una escena. La manipulación incorrecta de las fuentes de luz puede dar resultados no deseados en el aspecto de la escena final. Es por esto que con el afán de evitarle al programador final resultados que frustren sus intentos por utilizar el software, se tomo la decisión de omitir el uso de la iluminación basada en las características antes mencionadas sobre la superficie de los materiales.

faces: Vector de estructuras tipo "*struct_Face"* con la información correspondiente a todos los triángulos que conforman el objeto 3D.



vertexIndices: Vector que contiene los índices que hacen referencia a las coordenadas de los tres vértices del triángulo. Los índices corresponden a la posición de los elementos del vector *vértices* que se encuentran dentro de la estructura "*struct_Object"*.

normalsIndices: Vector que contiene los índices que hacen referencia a las normales de los tres vértices del triángulo. Los índices corresponden a la posición de los elementos del vector "*normals"* que se encuentran dentro de la estructura "*struct_Object"*.

texCoordIndices: Vector que contiene los índices que hacen referencia a las coordenadas de textura de los tres vértices del triángulo. Los índices corresponden a la posición de los elementos del vector *vértices* que se encuentran dentro de la estructura "*struct_Object"*.

Cabe mencionar que la normal "*normalsIndices[i]"* y la coordenada "*texCoordIndices[i]"* corresponden al vértice "*vertexIndices[i]"*.

planeEq: Contiene los coeficientes de la ecuación del plano del triángulo ($ax+by+cz+d = 0$).

materialIndex: Contiene el número del material utilizado para el triángulo. Este número corresponde al índice de algún elemento del vector "*materials"* que se encuentran dentro de la estructura "*struct_Object"*.

numberOfDraws: Sirve para registrar si el triángulo ya a sido dibujado en el "*frame"* actual. Como veremos más adelante, existen casos en los que una cara podría ser dibujada más de una vez si no se lleva un control adecuado. Este atributo es ocupado por el algoritmo que lleva a cabo este control.

vertices: Vector de elementos tipo "*float[3]"* que almacena las coordenadas cartesianas de los vértices que conforman los triángulos del objeto 3D.

normals: Vector de elementos tipo "*float[3]"* que almacena las coordenadas de las normales utilizadas para los vértice del objeto 3D.

texCoords: Vector de elementos tipo "*float[2]"* que almacena las coordenadas de textura utilizadas para los vértice del objeto 3D.

octreeParent: Nodo padre del "*octree"* que administra al objeto 3D.

displayLevelUsed: Nivel de profundidad dentro del octree en donde se encuentran los octantes con la información geométrica y de texturas del mundo 3D.

collisionLevelUsed: Nivel de profundidad dentro del octree en donde se encuentran los octantes utilizados para realizar la detección de colisiones.

Obtención de la información de un modelo: Archivos OBJ.

Los archivos OBJ contienen información que define la geometría de un objeto 3D. Para definir las propiedades de los materiales utilizados, se vale de una archivo por separado con extensión MTL. Ambos archivos se encuentran en formato "ASCII" . Los archivos OBJ pueden guardar la información tanto de objetos poligonales como aquellos de tipo "*free form*". Los objetos poligonales utilizan puntos, líneas y caras para ser representados mientras que los objetos de tipo "*free form"* usan curvas y superficies. En este proyecto de tesis será extraída la información únicamente los objetos poligonales.

Los siguientes datos son extraídos de los archivos OBJ:

- *Coordenadas cartesianas de los vértices*: Se identifica con la letra *v* dentro del archivo y nos indica que en los siguientes tres "*tokens"* se encuentra la información referente a las coordenadas cartesianas de un vértice. Estos tres "tokens" representan tres números de tipo flotante con las coordenadas cartesianas (*x,y,z*) del vértice. Por ejemplo:

```
v  7.636161 43.826744 4.425876
```

  en donde estamos representando un vértice con coordenadas *v*(7.636161, 43.826744, 4.425876).

- *Coordenadas de textura de los vértices*: Se identifica con la palabra *vt* dentro del archivo y nos indica que en los siguientes tres "*tokens"* se encuentra la información referente a una coordenada de textura. Estos tres "tokens" representan tres número de tipo flotante con las coordenadas de textura (*s,t,u*) cuyo rango está entre 0 y 1. Omitiendo la tercera componente, la coordenada *t*(0,0) representa la esquina inferior izquierda de la imagen y la coordenada *t*(1,1) la esquina superior derecha. Veamos un ejemplo ejemplo:

```
vt 0.286574 0.151111 0.0
```

  en donde estamos representando una coordenada de textura *t*(0.286574,0.151111,0.0). Ya que en el software solo procesaremos texturas bidimensionales, solo serán tomados en cuenta los dos primeros elementos *t*(*s*,*t*), es decir, *t*(0.286574, 0.151111).

- *Normales de los vértices*: Se identifica con la palabra *vn* dentro del archivo y nos indica que en los siguientes tres "*tokens"* se encuentra la información referente a las componentes de una normal. Estos tres "tokens" representan tres número de tipo flotante con las coordenadas componentes (x,*y,z*) de una normal. Por ejemplo:

```
vn  -0.826805 0.300613 0.475422
```

en donde estamos representando una normal con componentes *n*(-0.826805 0.300613, 0.475422).

- Material utilizado para cada triángulo: Se identifica con la palabra `usemtl` dentro del archivo y nos indica que el siguiente "token" contiene la cadena que identifica al material que será utilizado para los siguientes triángulos. Por ejemplo:

```
usemtl __axe_bmp
```

en donde se indica que el material que será utilizado en las próximas caras se identifica con el nombre *__axe_bmp* dentro del archivo de materiales MTL.

- Índices de los vértices que conforman cada triángulo: Se identifica con la letra *f* dentro del archivo y nos indica que en los siguientes tres "*tokens"* se encuentra la información referente a las coordenadas cartesianas, coordenadas de textura y normales de los tres vértices que conforman una cara.. Por ejemplo:

```
f 865/2/85 863/5/81 864/9/16
```

significa que la cara está compuesta por tres vértices *v* con índices *865*, *2* y *85*, cada uno de los cuales tienen las coordenadas de textura *vt* con índices *863*, *5* y *81* respectivamente para cada uno y normales *vn* con índices *864*, *9* y *16* respectivamente para cada uno.

Los elementos anteriormente descritos siempre aparecen dentro del archivo en el siguiente orden aunque entre ellos pueden existir otro tipo de etiquetas que no manejaremos:

1. *v*
2. *vt*
3. *vn*
4. *usemtl*
5. *f*

Hemos visto que un elemento *f* que describe las propiedades geométricas de un triángulo se describen usando índices. Esto significa que cada vez que encontremos un elemento de tipo *v*, *vt* o *vn* debemos insertarlo inmediatamente en su vector correspondiente y ese será el orden correcto que debe existir para obtener índices válidos. Cabe mencionar que cuando trabajamos con C++, a cada índice hay que restarle *-1* para obtener la posición correcta ya que el formato OBJ toma como primer índice el número *1* y no el *0*.

Para terminar la extracción de la información de un modelo 3D debemos finalmente extraer la información del archivo de materiales MTL. De este archivo lo único que ocuparemos será el nombre del archivo de textura. Se describen también las propiedades de luz ambiental, difusa, especular, emisión, exponente especular y valor alfa que aunque no serán ocupadas en el software se han extraído para uso futuro. Los siguientes datos son extraídos del archivo MTL:

- *Nombre de identificación del material*: Se identifica con la palabra *newmtl* seguido de la cadena que identifica al material. Esta cadena es usada para asignar un material dentro de un archivo OBJ (mediante la palabra *usemtl*). Por ejemplo:

  ```
  newmtl __axe_bmp
  ```

  en donde podemos ver que el nombre que identifica al material cuyas propiedades serán definidas a continuación es *__axe_bmp*.

- *Exponente especular (shininess)*: Se identifica con la palabra *Ns* dentro del archivo e indica que el siguiente "*token"* es una cadena que representa un valor numérico cuyo rango normalmente varía entre 0 y 1000. Por ejemplo:

```
Ns 32
```

- Valor alpha: Se identifica con la letra *d* dentro del archivo e indica que el siguiente "*token"* es una cadena que representa un número flotante entre 0.0 y 1.0 con el valor alpha del material. Un valor de 1.0 indica que el material es completamente opaco y un valor de 0.0 indica que es completamente transparente. Por ejemplo:

```
d 0.6
```

- Color difuso del material (*diffuse*): Se identifica con la palabra *Kd* dentro del archivo e indica que los siguientes tres "*tokens"* corresponden a tres valores de tipo flotante (entre 0 y 1) que definen las componentes RGB del color difuso del material. Por ejemplo:

```
Kd 0.4 0.4 0.4
```

- *Color especular del material* (*specular)*: Se identifica con la palabra *Ks* dentro del archivo e indica que los siguientes tres "*tokens"* corresponden a tres valores de tipo flotante (entre 0 y 1) que definen las componentes RGB del color especular del material. Por ejemplo:

```
Ks 0.7 0.7 0.7
```

- Color de la luz ambiental (*ambient*): Se identifica con la palabra *Ka* dentro del archivo e indica que los siguientes tres "*tokens"* corresponden a tres valores de tipo flotante (entre 0 y 1) que definen las componentes RGB del color de la luz ambiental que será reflejado en el material. Por ejemplo:

```
Ka 0.3 0.3 0.3
```

Obtención de la información de una escena: Editor T.ED.

Sería sumamente laborioso describir una escena compleja sin tener una referencia visual de lo que queremos desplegar con el software creado en este proyecto de tesis. Es por esta razón que se optó por utilizar una herramienta de edición de escenas en conjunto con el software desarrollado. Los requerimientos buscados en esta herramienta fueron los siguientes:

- Fácil de utilizar y aprender.
- Importación de algún formato 3D de gran difusión.
- Exportar la información de los elementos de la escena en un formato claro y entendible.
- Capacidad para manejar escenas en exteriores de gran tamaño.
- Gratuito o de bajo costo.

Se probó una gran variedad de herramientas de modelado así como editores de niveles para juegos pero resultaron ser difíciles de utilizar, no producían la información requerida y eran incapaces de manejar escenas en exteriores de gran dimensión. Solo el editor de escenas T.ED pudo cumplir los requerimientos buscados para interactuar con relativa facilidad con el software desarrollado. No es una herramienta gratuita pero su costo muy bajo ($30dls aproximadamente). Esta herramienta esta diseñada para crear escenas en exteriores de gran dimensión y se pueden importar los siguientes formatos 3D: .X(DirectX), .3DS (3D Studio Max) y .B3D (Blitz 3D). Este editor no maneja el formato .OBJ utilizado en el software desarrollado por lo cual se han tenido que utilizar las herramientas de conversión BitTurn y PolyTrans mencionadas en el capítulo 1. Para trabajar con el editor necesitamos realizar la conversión de los modelos 3D a alguno de los formatos mencionados anteriormente. El software desarrollado utilizará solo archivos OBJ y es por esto que cuando lea el archivo creado por el editor T.ED, a los archivos 3D referenciados en este siempre les reemplazara su extensión por OBJ.

Básicamente lo que haremos será crear visualmente mediante el editor T.ED una escena en 3D. Una vez construida la escena el editor generará un archivo de texto en donde se defina en que parte del mundo 3D estará localizado cada objeto así como su rotación y

escalamiento. Este archivo de texto está compuesto básicamente por dos tipos de objetos: Fuentes e instancias.

Los objetos fuente hacen referencia a los archivos que contienen los distintos modelos 3D utilizados en la escena. Los objetos de tipo instancia hacen referencia a los objetos fuente ya sea una o más veces, depende que tantas ocasiones se utilice un mismo modelo en la escena como por ejemplo en el caso en que el modelo de un arbusto se utilice en varias locaciones de la escena. Cada instancia pose atributos de translación, rotación y escalamiento de la fuente a la cual hacen referencia. Finalmente todas las variantes geométricas descritas por las instancias de los modelos fuente son guardadas en una estructura de tipo "*structObject"*, es decir, toda la geometría de la escena se almacena en una estructura de este tipo.

Los datos que buscaremos en un archivo de texto con extensión EPR generado por el editor T.ED son los siguientes:

Para los objetos fuente:

- `[SRC]` : Este "*token"* nos indica que a continuación se presenta la información referente a un objeto fuente.

- `TED_Id` : Si un "*token*" de una fuente inicia con esta subcadena significa que en el se encuentra una cadena de identificación con la cual las instancias referencían a este objeto fuente. Por ejemplo:

  ```
  TED_Id=183945-170697
  ```

  en donde *183945-170697* es la cadena que identifica a la fuente actual.

- `TED_Filename` : Si un "*token"* de una fuente inicia con esta subcadena significa que contiene el nombre y ruta del archivo con el modelo 3D del objeto fuente. Por ejemplo:

```
TED_Filename=C:\Ricardo\Meshes\grass.3ds
```

En donde el nombre y ruta del archivo que contiene el modelo 3D es C:\Ricardo\Meshes\grass.3ds . Es muy importante mencionar que el software desarrollado solo tomara el nombre de archivo y se descartara la ruta y la extensión de este. La ruta es remplazada por aquella en donde se encuentra el archivo EPR por lo cual es en ese mismo lugar deben de encontrarse los archivos con los modelos 3D así como las texturas que ocupa. Esto es con el fin de unificar las rutas de todos archivos con los contenidos de una escena. Así mismo, la extensión del archivo 3D es reemplazado por la extensión OBJ que es el formato que es capas de interpretar el software desarrollado.

- `[/SRC]` : Indica el final de la información de la fuente actual.

Para las instancias:

- `[INSTANCES]` : Este "*token"* nos indica que a continuación se presenta la información referente a un objeto fuente.

- `TED_Id` : Si un "*token"* de una instancia inicia con esta subcadena significa que en el se encuentra una cadena de identificación que hace referencia a la fuente que se quiere instanciar. Por ejemplo:

```
TED_Id=183945-170697
```

- `TED_Pos_XYZ` : Si un "*token"* de una instancia inicia con esta subcadena significa que contiene las coordenadas de traslación con la cual se quiere transformar la información geométrica de la fuente. Por ejemplo:

```
TED_Pos_XYZ=265.844,45.0164,328.865
```

Significa que la instancia será un duplicado del modelo 3D que contiene la

fuente TED_Id especificada y en el cual se quiere trasladar a las coordenadas (*265.844, 45.0164, 328.865*) del mundo 3D.

- `TED_Rot_XYZ` : Si un "*token"* de una instancia inicia con esta subcadena significa que contiene las rotaciones alrededor de los ejes *x*, *y*, y *z* del modelo y en grados, con las cuales se quiere transformar la información geométrica de la fuente. Por ejemplo:

  ```
  TED_Rot_XYZ=37.6166,0.0,5.20757
  ```

  Significa que la instancia será un duplicado del modelo 3D que contiene la fuente TED_Id especificada y en el cual se quiere rotar el modelo *265.844* grados alrededor del eje *x*, *45.0164* grados alrededor del eje *y,* y *328.865* grados alrededor del eje *z* del modelo 3D.

- `TED_Sca_XYZ` : Si un "*token"* de una instancia inicia con esta subcadena significa que contiene los escalamientos a lo largo de los ejes *x*, *y*, y *z* del modelo, que se quieren realizar en la información geométrica de la fuente. Por ejemplo:

  ```
  TED_Rot_XYZ=37.6166,0.0,5.20757
  ```

  Significa que la instancia será un duplicado del modelo 3D que contiene la fuente TED_Id especificada y en el cual se quiere realizar un escalamiento del *%*150 sobre el eje x, *%*100 sobre el eje y, y *%*60 sobre el eje z del modelo 3D.

- `[/INS]` : Indica el final de la información de la instancia actual.

Como mencionado anteriormente, realizamos las transformaciones descritas en las instancias a los modelos 3D indicados dentro de las mismas y la información geométrica obtenida es guardada en una estructura de tipo "*struct_Objec"t* que al final contendrá la geometría de toda la escena descrita en el archivo EPR.