HotCoffee Engine

Documento de Diseño Técnico

Max Alberto Solano Maldonado

Game Scene Graph

Grafo

En su expresión más básica, un grafo es un conjunto de vértices (nodos), que están relacionados entre ellos. Gráficamente dos nodos están conectados por medio de una arista. ***Véases Ilustración: Grafo***

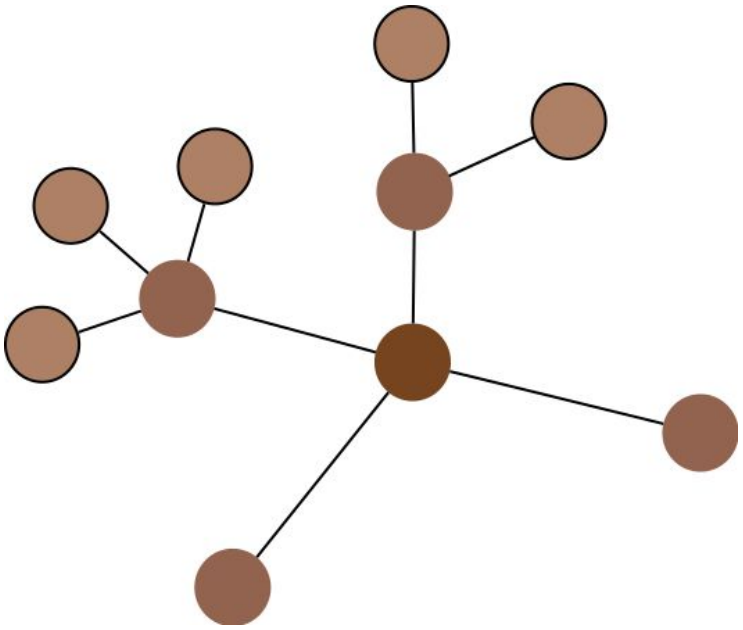


Ilustración: Grafo. *Un grafo está compuesto por un grupo de nodos y aristas.*

Matemáticamente, un grafo no es más que un conjunto de vértices y aristas que representación la relación que hay entre los vértices.

$$G = (V, A)$$

V es el conjunto de vértices. A es el conjunto de aristas. Asimismo "A" no es más que un conjunto de pares definido por:

$$(u, v) \; donde \; v \in V$$

Grafo Conexo y Grafo no Conexo

El grafo conexo es un grafo por el cual todos sus nodos están conectados entre sí. Esto quiere decir, que desde cualquier nodo, es posible llegar hacia otro nodo por medio de una o más rutas [12]. A continuación se ilustra un grafo conexo:

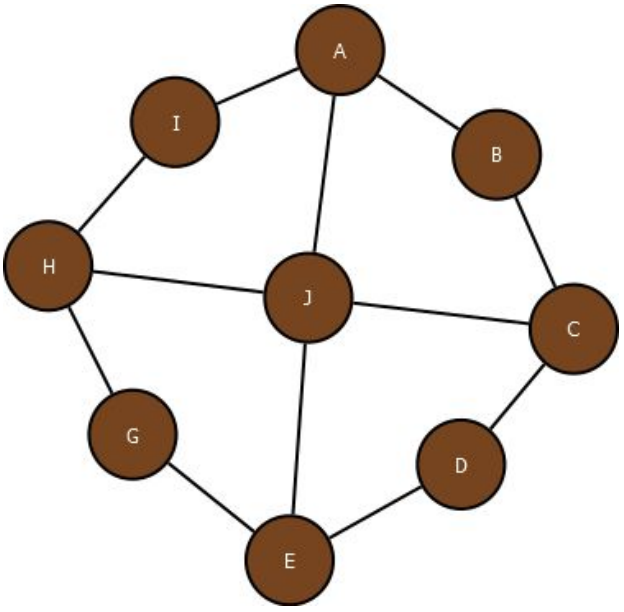


Ilustración: Grafo Conexo.

Por otro lado, tenemos el grafo no conexo, que consiste en un grupo de nodos que no necesariamente deben estar completamente conectados entre sí. Esto quiere decir que no siempre se puede llegar desde un nodo a otro por medio de una ruta [12]. La siguiente ilustración presenta un grafo no conexo:

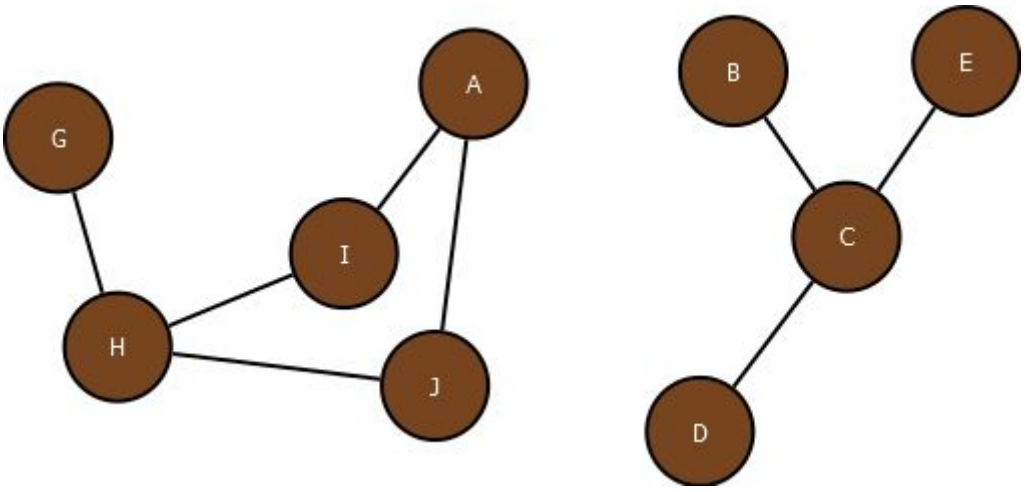


Ilustración: Grafo no Conexo.

Toma en cuenta que el nodo "a" puede llegar al nodo "c" por medio de una ruta, sin embargo, no es posible llegar al nodo "b".

Grafos en los Videojuegos

Los grafos están presentes en una gran parte de la estructura de un juego. Son tan básicos como las matrices y vectores matemáticos. Entre algunos ejemplos podemos encontrar el "Malla de Navegación" de un objeto. La Malla de Navegación no es más que un simple grafo que delimita el área por el cual un objeto puede trasladarse.

En ***Ilustración: Navigation Mesh*** se presenta una malla de navegación generada a partir de una escena. La Malla "Azul" representa el área por el cual todo objeto puede trasladarse.

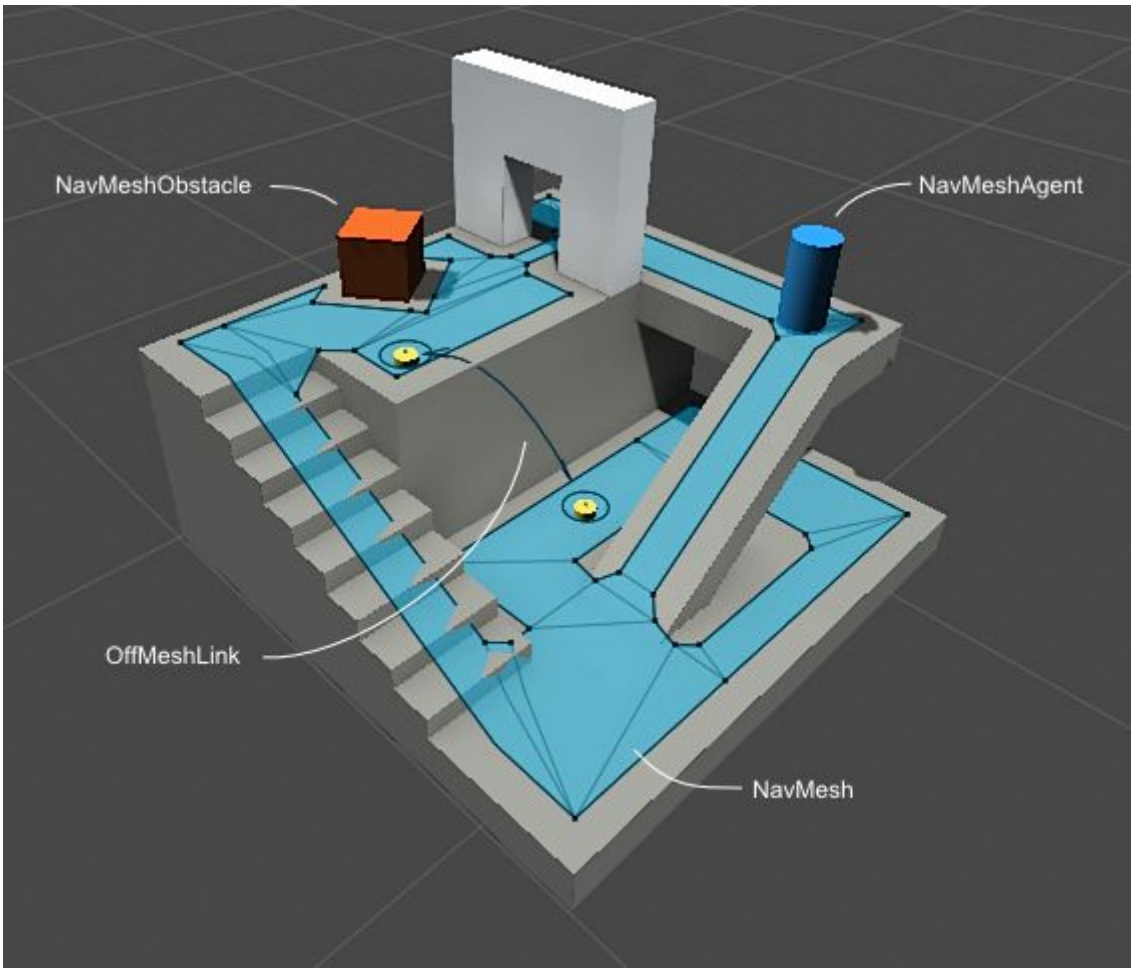


Ilustración: Navigation Mesh. *Enlace: [https://steamcommunity.com/games/906660/announcements/detail/17129468928093 80937] Las mallas de navegación no son más que un grafo que delimita el área de traslación de los objetos dinámicos.*

A partir de este grafo, los objetos utilizan algoritmos de búsqueda de caminos para trasladarse de un punto a otro.

Bajo los conceptos anteriores, se puede decir que un modelo 3D es un grafo. Son un conjunto de vértices que están conectados entre sí en un espacio tridimensional para representar una figura. A continuación se ilustra una malla tridimensional con forma de elefante:

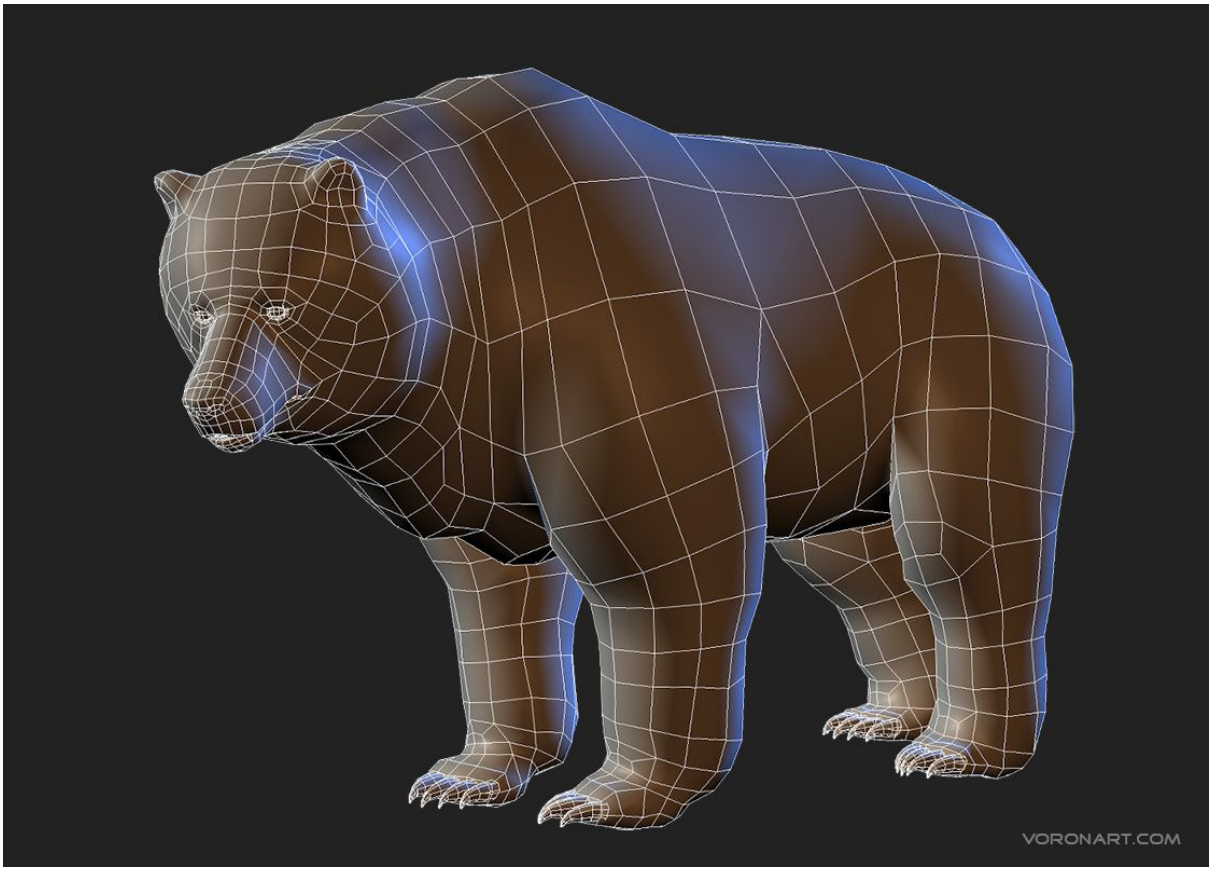
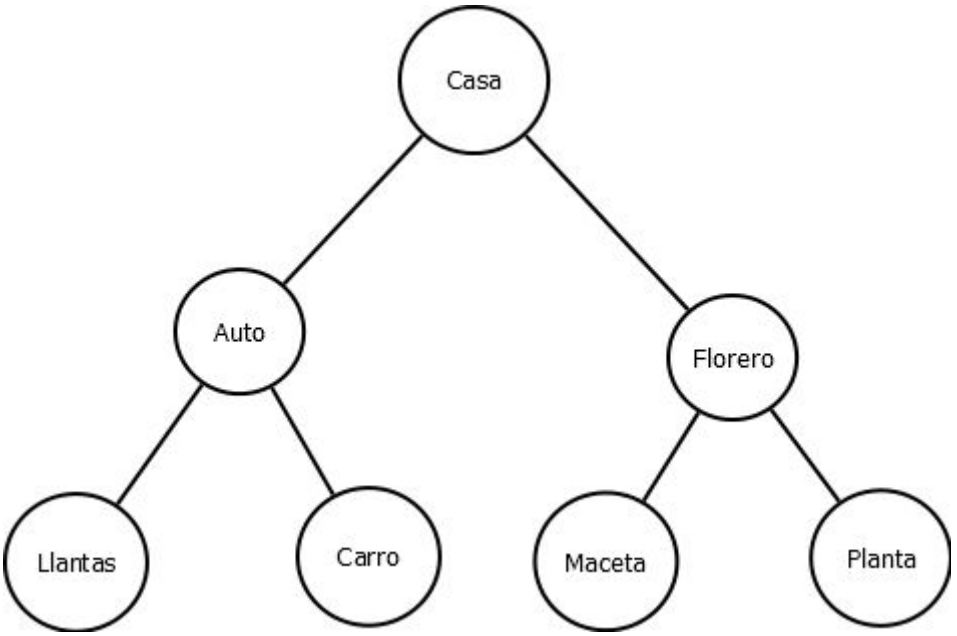


Ilustración: Malla tridimensional de Oso. *Enlace: [https://voronart.com/portfolio-items/animated-brown-bear-3d-character/] La malla tridimensional que forma la figura de un oso, está compuesta por una serie de nodos y aristas.*

El Scene Graph es una de las implementaciones de grafos dentro del mundo de los videojuegos. Un Scene Graph es un grafo cuyo vértices representan los objetos que existen en el mundo y las aristas representan la relación o "herencia" que existen entre los objetos [6]. A continuación se presenta un Scene Graph que relaciona las diferentes partes de un Robot.



El nodo “Auto” y “Florero” son descendientes del nodo “Casa”. Esto quiere decir que cualquier transformación aplicada al nodo “Casa”, se verá reflejada en sus hijos (“Auto” y “Florero), y estos a sus descendientes. En conclusión, las transformaciones de un nodo, tendrán efecto en todos sus descendientes, incluyendo los descendientes de los descendientes.

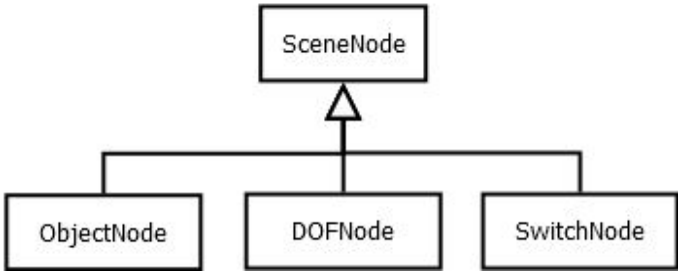
Scene Graph

El Scene Graph de HotCoffee estable la relación que hay entre los objetos del mundo y por consiguiente, el orden de multiplicación de matrices.

Tipos de Nodos

Scene Node	Nodo base de la que derivan los nodos especiales.
Object Node	Nodo que contiene un solo GameObject de la escena.
DOF Node	Nodo que contiene una matriz de transformación
Switch Node	Nodo que permite tomar una sola vertiente de un conjunto de rutas.

SceneNode Heritage

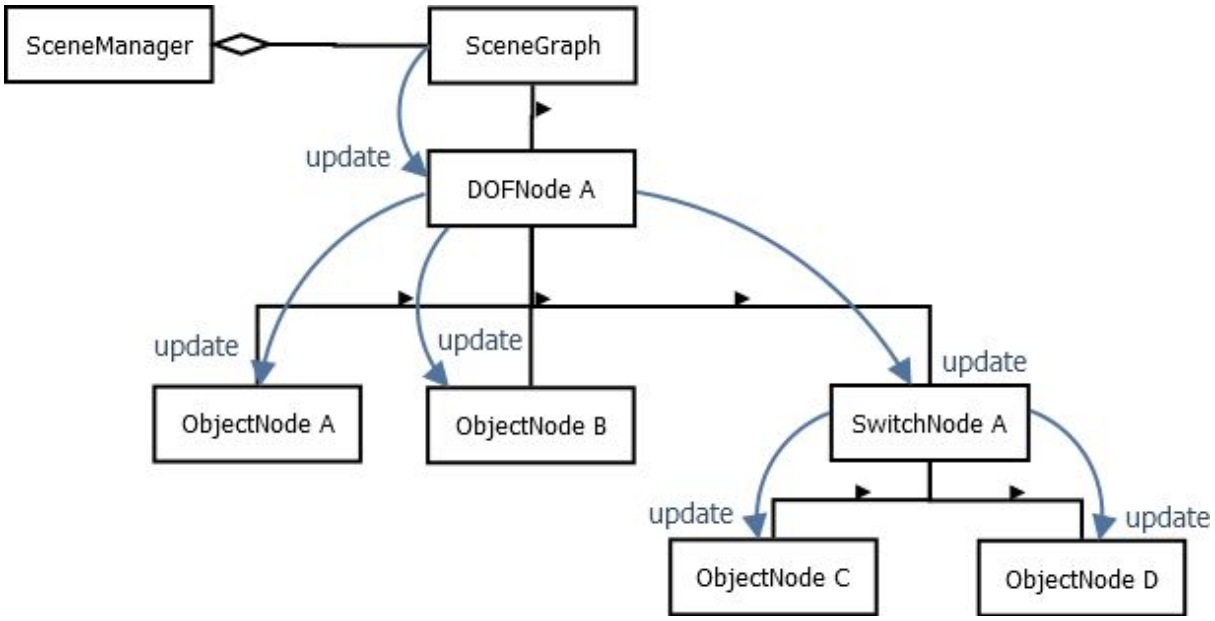


Update

Durante la fase de "update" los nodos se actualiza de manera recursiva. Primero realizan el "update" de ellos mismos y luego llaman al update de los nodos hijos:

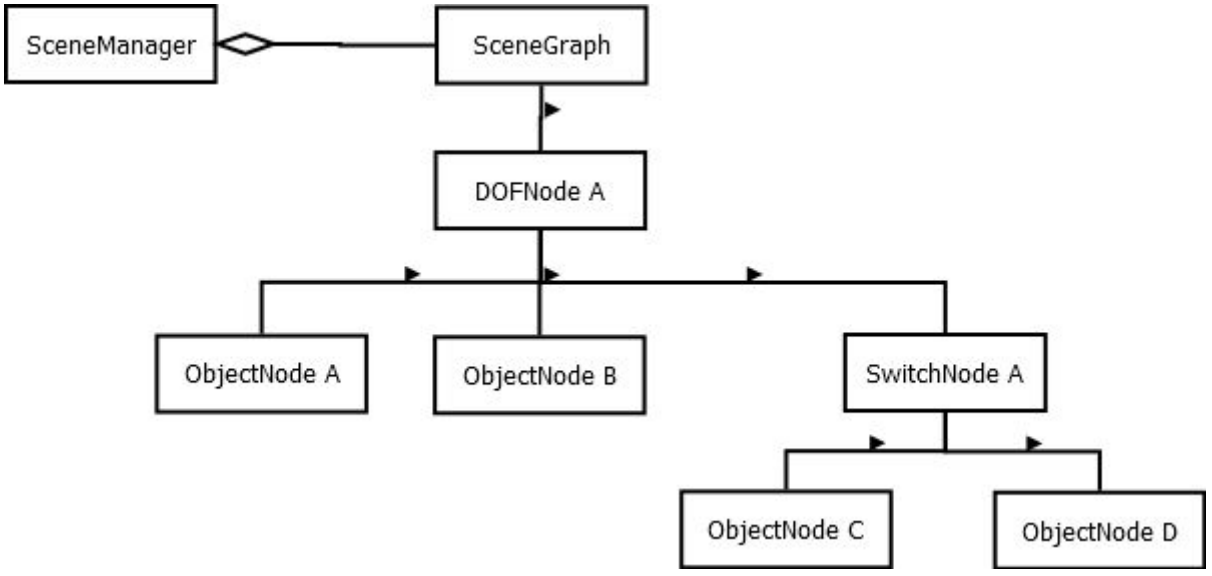
Update Method
-> Node Update

-> Call Children Update



Drawing

Durante la fase de "Drawing" Scene Graph regresa una buffer de GameObject en el orden que deben ser dibujados.



Ya que el árbol recorre sus hijos de izquierda a derecha, el resultado del gráfico anterior es el siguiente:

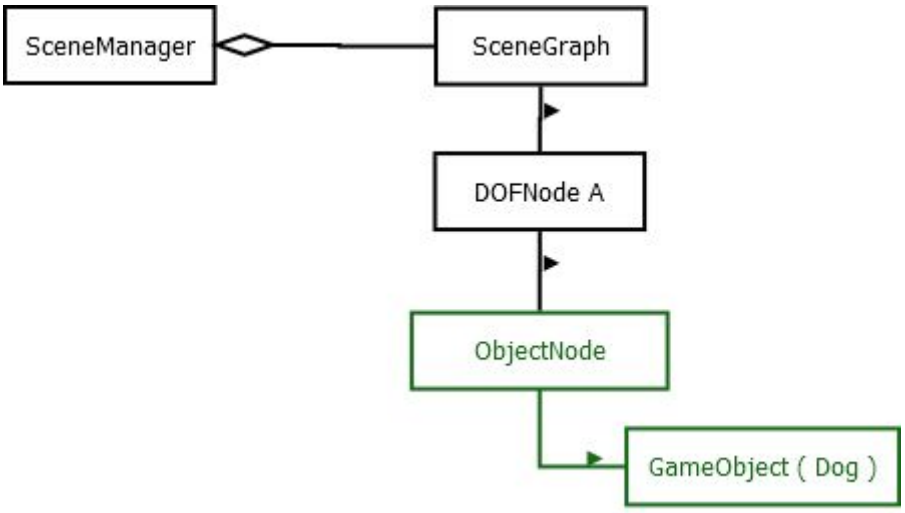
FILO
ObjectNode D
ObjectNode B
ObjectNode A -> First In - Last Out

El primer "draw" que se llama es del ObjectNode D hasta llegar al ObjectNode A.

ObjectNode

Para crear un ObjectNode, basta con crear un GameObject desde el Scene Manager:

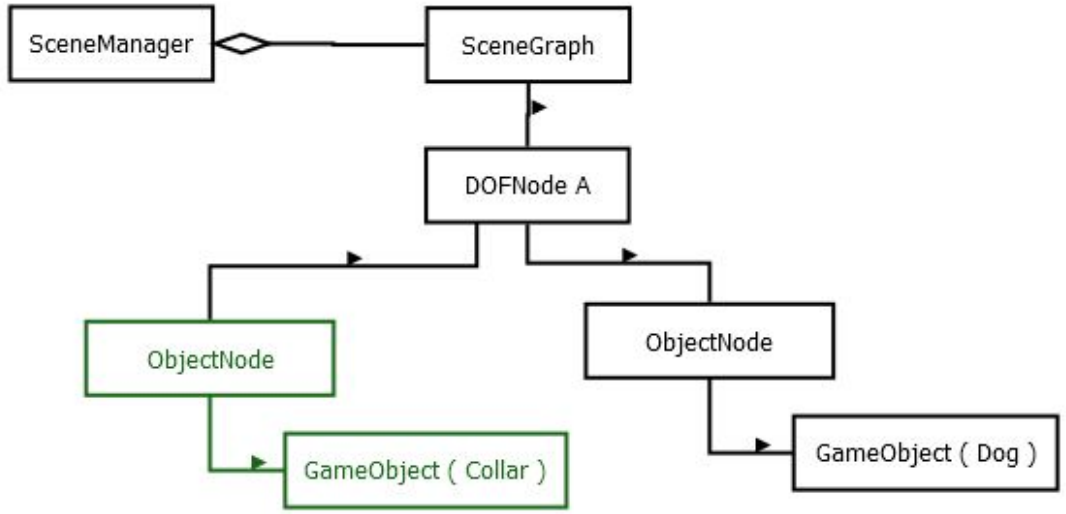
```
// SceneManager pointer
auto p_scene_mng = SceneManager::getSingletonPtr();

// Create Dog
WeakPtr<hcEngineSDK::GameObject> dog;
dog = p_scene_mng->createGameObject("Dog");
```

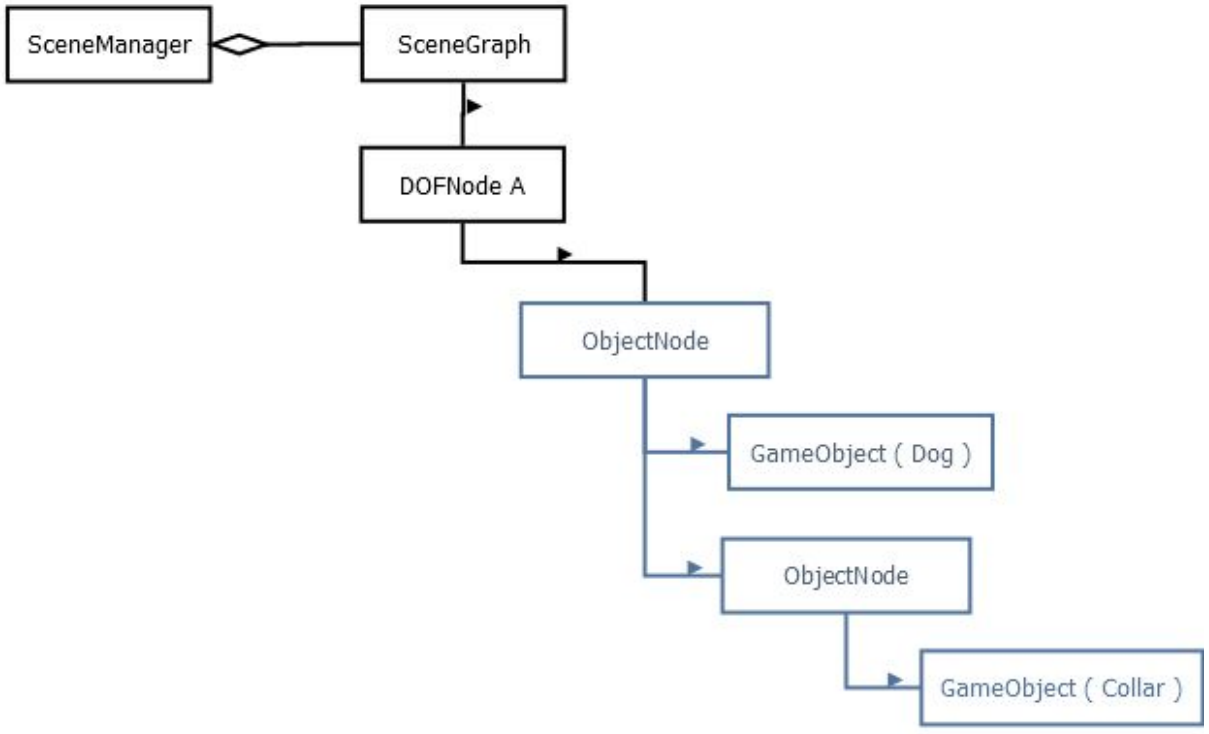
Para agregar un hijo al nodo de “Dog”, primero debe crearse el objeto que desea agregar como hijo. Posteriormente, se debe asignar dicho objeto, como hijo del nodo “Dog”.

```
// Step A: Create collar
WeakPtr<hcEngineSDK::GameObject> collar;
collar = p_scene_mng->createGameObject("Collar");

// Step B: add Collar to Dog node
dog->addChild(collar);
```

Step A



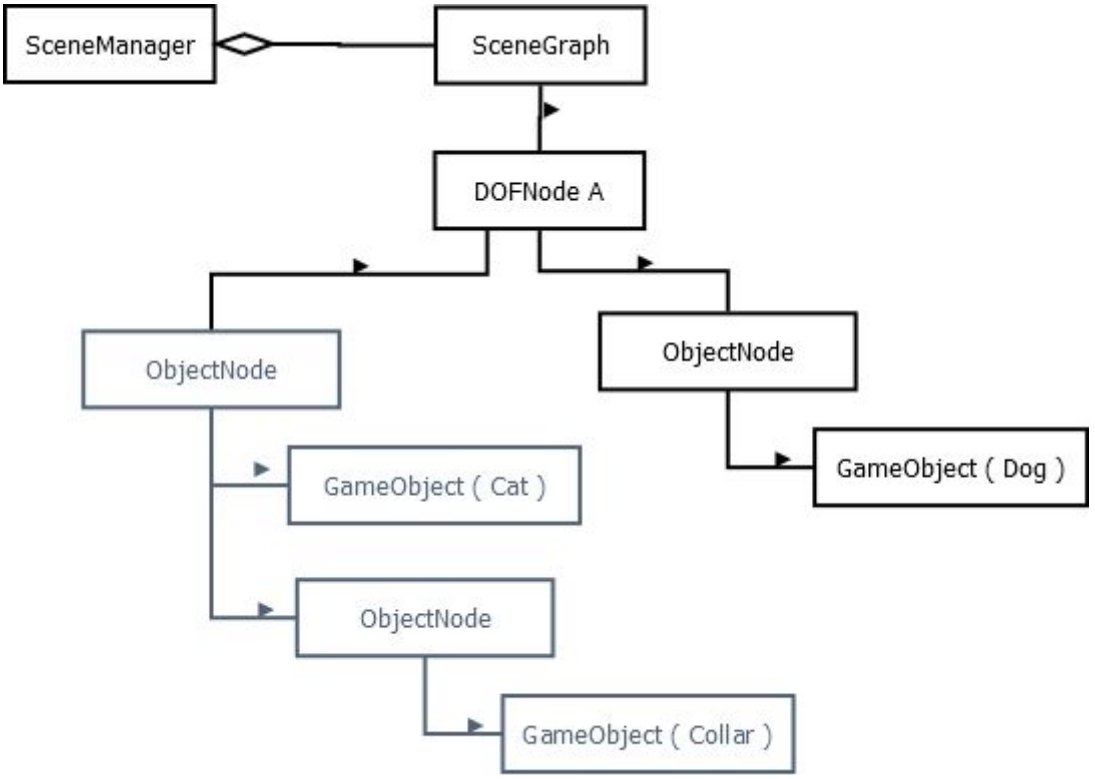
Step B



Así mismo, asignar un "GameObject B" como hijo de un "GameObject C", implica que el "GameObject B" dejaré de ser hijo de su padre actual (si tiene uno), para ser hijo del "GameObject C".

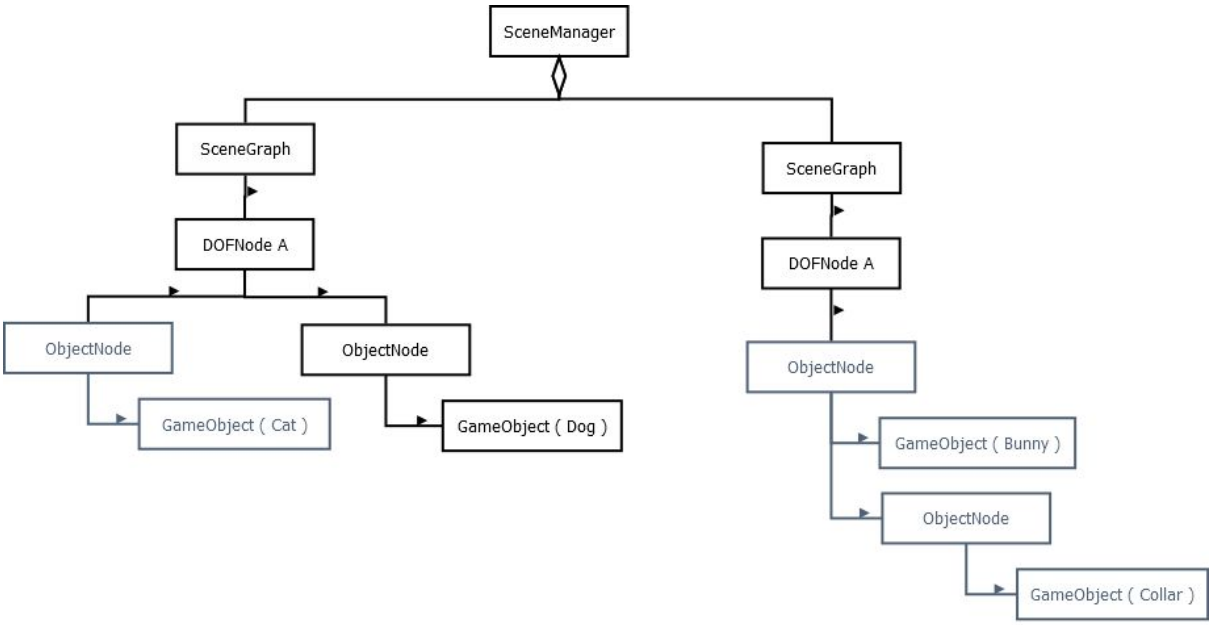
```
// create Cat
```


```
WeakPtr<hcEngineSDK::GameObject> cat;
cat = p_scene_mng->createGameObject("Cat");

// add Collar to Cat node
cat->addChild(collar);
```



Los nodos también pueden cambiar de Scene Graph, por medio la misma sintaxis. En el siguiente ejemplo, “Collar” pasa a ser hijo de “Bunny”, que pertenece a otro Scene Graph

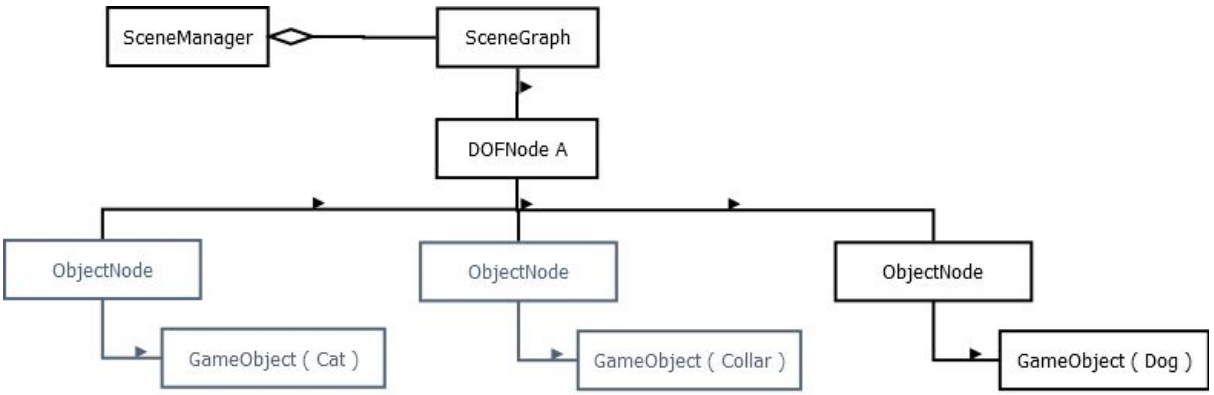
```
// add Collar to Bunny node
bunny->addChild(collar);
```

Cuando un hijo “A” es removido de su padre “B”. Este pasa a ser hijo del padre del padre “C”. En el siguiente ejemplo, “Collar” es removido de “Cat”, por lo tanto “Collar” ahora es hijo de “DOFNode A”.

El objeto removido no puede desligarse del árbol por completo, solamente subir de nivel o ser asignado a una nueva posición.

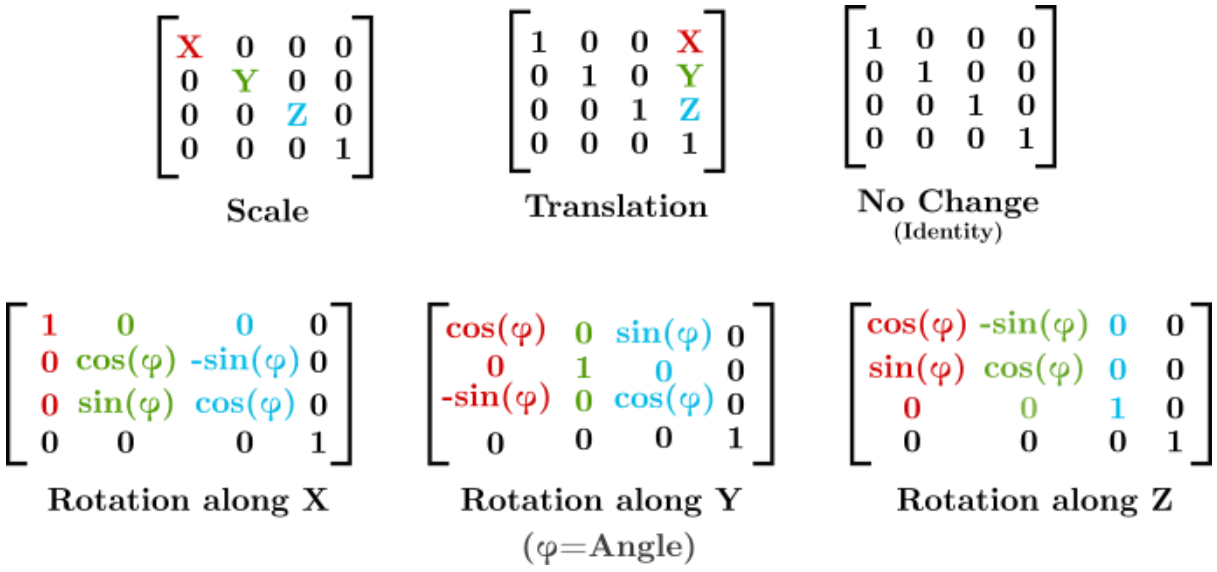
```
// remove Collar from Cat, and get the Collar GameObject
WeakPtr<hcEngineSDK::GameObject> collar_ref;
collar_ref = cat->removeChild("Collar");
```



La única manera de desligar un objeto del Scene Graph es por medio de eliminación. Eliminar un GameObject de la escena, implica que todos los nodos derivados de dicho GameObject, también serán eliminados del grafo.

Transformaciones

Básicamente, nos referimos a la aplicación de matrices de traslación, escala y rotación a los vértices de un modelo. La representación por Matriz de cada una de estas transformación son las siguientes [7]:



Como se puede observar, existen varias matrices de transformación. Las rotaciones, incluso, poseen tres matrices, cada una representa la transformación de los vértices sobre uno de los tres ejes espaciales (x,y,z).

Composición de Matrices

Las matrices no son conmutativas, esto quiere decir que el orden de los factores SÍ altera el producto. Una malla de vértices puede sufrir cambios muy radicales, incluso cambiar de forma si las matrices no son

multiplicadas de forma correcta. Para ello se sigue el siguiente orden de multiplicación [4]:

$$p' = (T * (R * (S * p)))$$

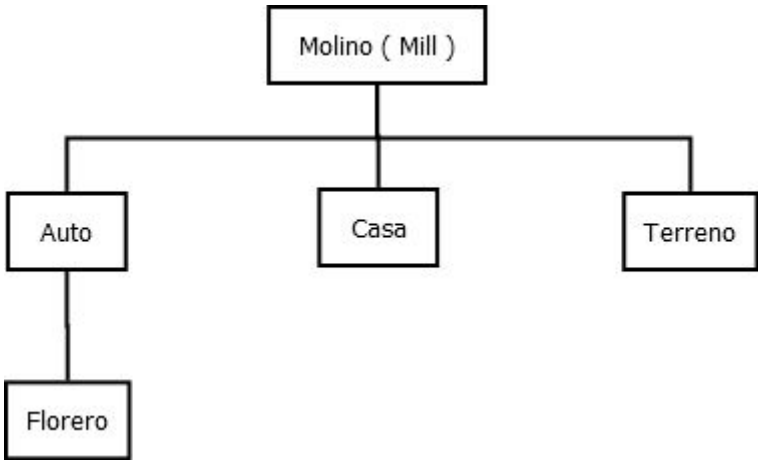
donde " *p´ ″* es el vértice transformado, *"T"* es la Matriz de traslación, *"R"* la matriz de rotación, *"S"* la matriz de escala y *"p"* el vértice que se desea transformar [9]. Así mismo, obtendremos los mismo resultados si es aplicado de la siguiente manera:

$$p' = (T * R * S) * p$$

En un motor de videojuegos, previa a la fase de dibujo, las matrices de "Modelo" son calculadas. Las Matrices de Modelos es el resultado de la multiplicaciones de *T * R * S* respectivamente. Esta Matriz es quien dicta cómo se deben transformar los vértices del modelo.

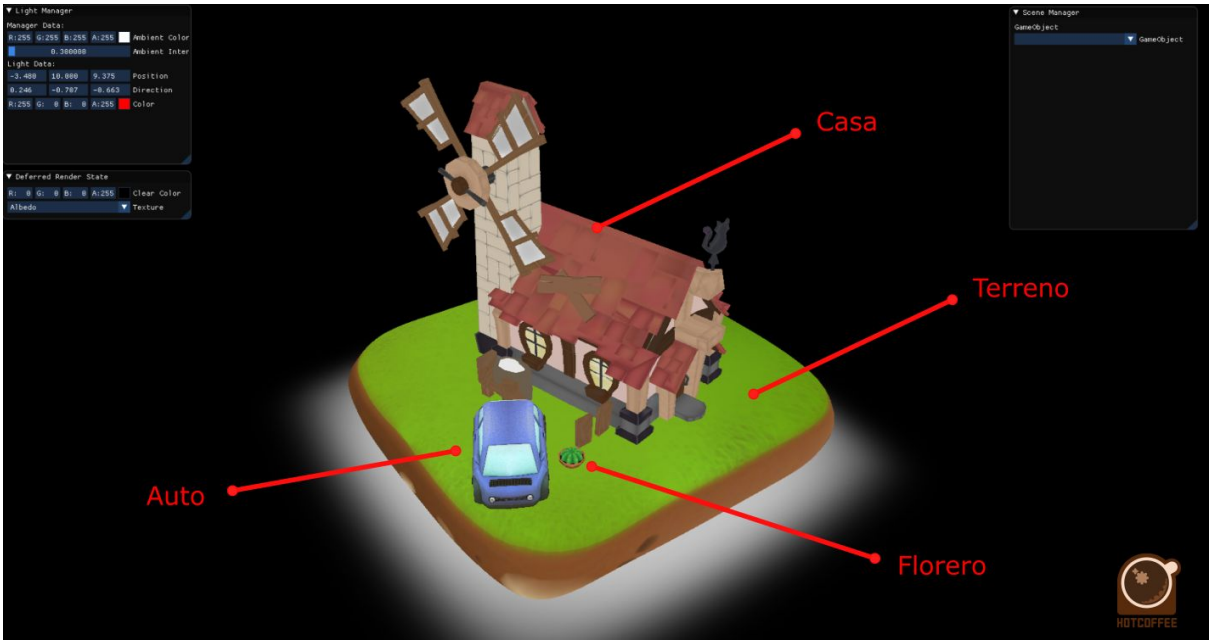
Scene Graph y Transformaciones

El scene graph estable el orden en el que las transformaciones son aplicadas a los objetos, por medio de descendencia. Sea la siguiente relación entre objetos:



HotCoffee Design: Escena del Molino. *Grafo que ilustra la jerarquía entre los objetos de la escena del molino.*

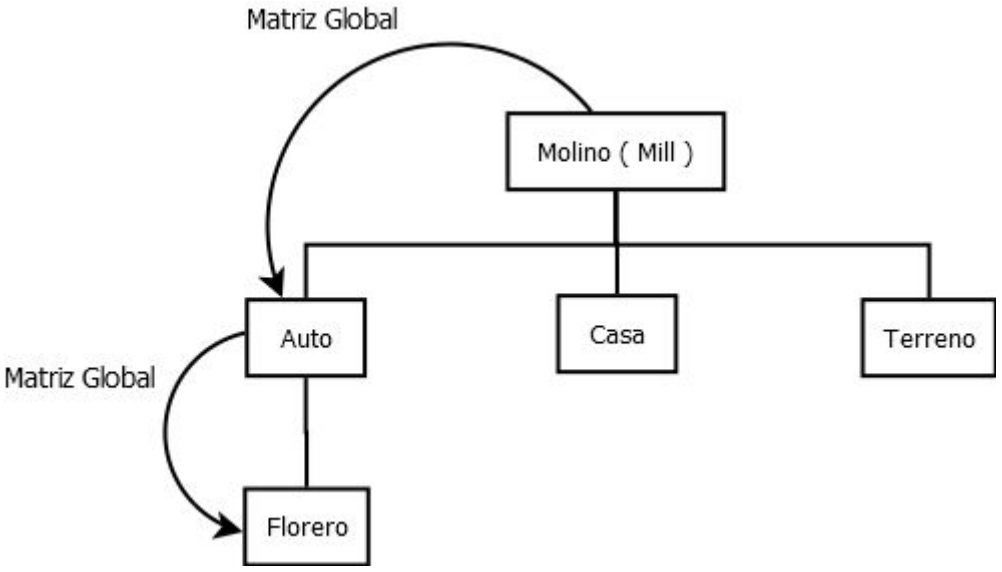
El objeto Molino (Mill) es quien contiene a los demás objetos del mundo. Cualquier transformación aplicada al objeto Molino afectará a los objetos que descienden de él.



HotCoffee Engine: Escena del Molino*. En la escena del molino existen los siguientes objetos en el mundo: Casa, Auto, Florero y Terreno.*

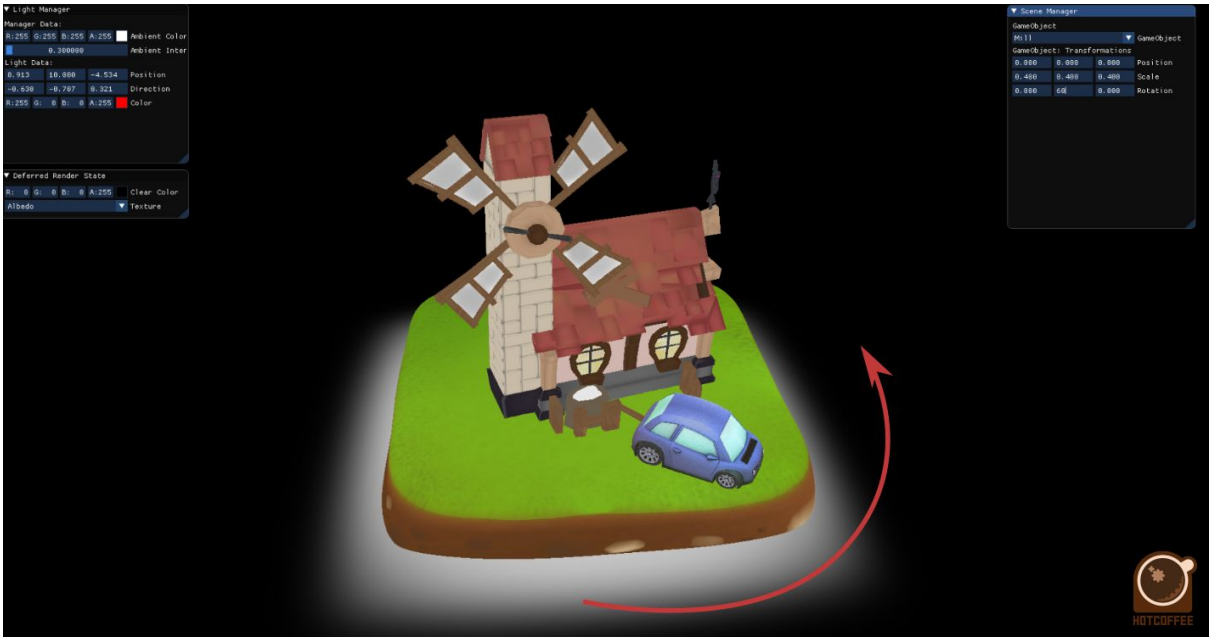
Al objeto "Molino" se le aplica una matriz de rotación en el eje Y de 60°. Por consecuencia, todos los objetos contenidos en "Molino" recibirán dicha transformación a partir del punto de ancla del objeto padre. Para entender por completo este efecto, debemos tomar en cuenta la existencia de las matrices locales y las matrices globales.

Las matrices locales definen las transformaciones de un objeto las coordenadas locales. Las matrices globales definen las transformaciones de un objeto a partir de las transformaciones obtenidas por los nodos padres.



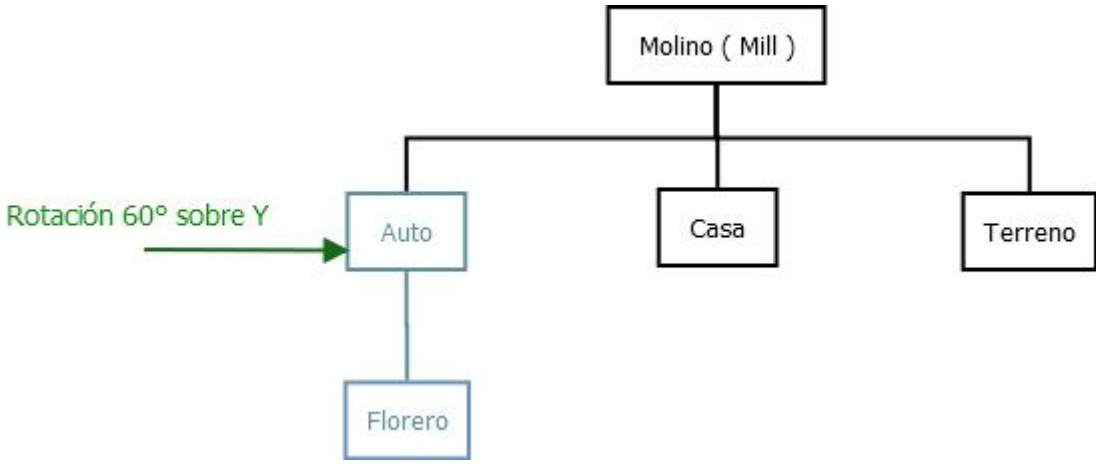
HotCoffee Design: Escena del Molino. *El orden y herencia de matrices son establecidas por la relación "padre-hijo" entre los nodos del grafo.*

La matriz global deriva del cúmulo de transformaciones pasadas. El nod "Auto" se verá afectado por la matriz global del nodo "Molino". El nodo "Florero", se verá afectado por la multiplicación de las matrices globales de "Auto" y "Molino". De esta manera es como los objetos se ven afectados por las transformaciones de sus antecesores.



HotCoffee Engine: Escena del Molino*. El nodo "Molino" ha recibido una rotación de 60° sobre el eje Y, por lo tanto afecta todos los objetos descendientes.*

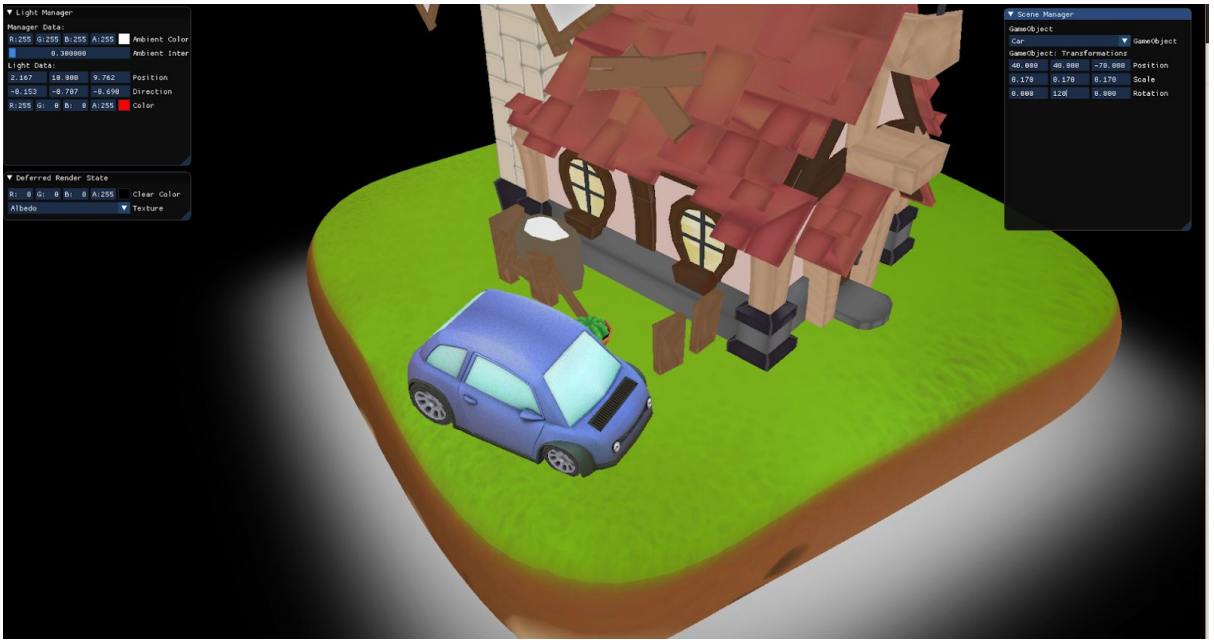
Si la misma matriz de rotación es aplicada al objeto “Auto”, éste tendrá efecto en el “Florero”, pero no con los demás objetos del mundo. Esto se debe a que las transformaciones aplicadas a un objeto sólo afectan a los objetos descendientes, más no con los objetos precedentes.



HotCoffee Design: Escena del Molino. *Una matriz de rotación de 60° sobre el eje Y es aplicada al nodo “Auto”, por consiguiente, el nodo “Florero” será afectado.*

HotCoffee Engine: Escena del Molino. *Dentro de la jerarquía de la escena, el “Florero” es hijo de “Automóvil”.*

Auto después de la transformación. Note que el florero ha sido afectado por la transformación, sin embargo los demás objetos permanecen intactos:



HotCoffee Engine: Escena del Molino. *El Automóvil ha recibido una rotación de 60° sobre el eje Y. La transformación afecta a sus objetos descendientes, en este ejemplo es el objeto "Florero".*

El Scene Graph es un elemento que no sólo relaciona los objetos que existen en el mundo, sino también es utilizado para las animaciones de objetos e incluso optimización de render.

Los grafos pueden considerarse elementos sencillos, pero que son básicos en sistemas complejos como los motores de videojuegos.

Game Resources Management

Recurso

Recurso es todo material ajeno al Motor pero necesario para el desarrollo del juego. Algunos recursos son Texturas, Modelos, Animaciones, Clips de Audio [2].

La ***Ilustración: Modelo Tridimensional*** e ***Ilustración: Textura para Modelo*** son ejemplos de recursos para un videojuego.

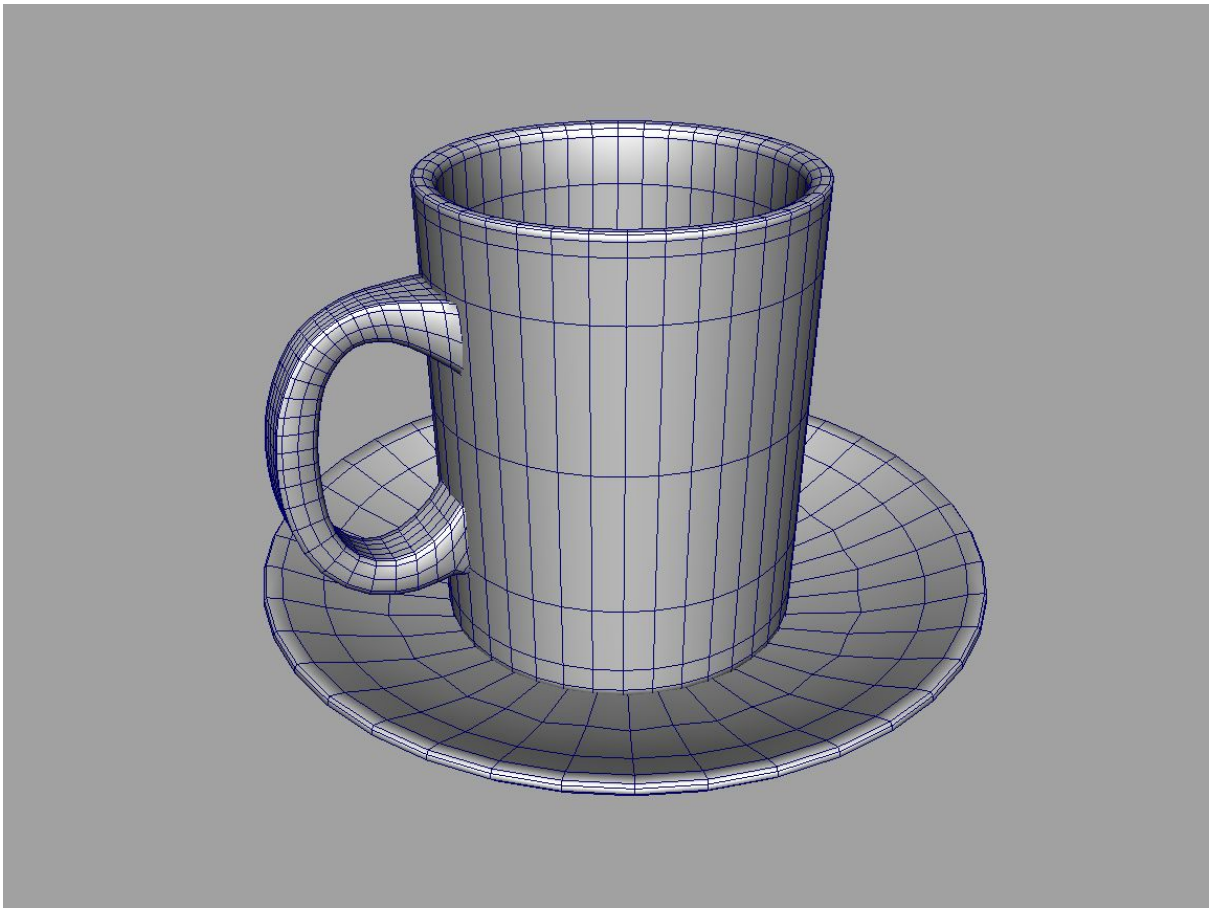


Ilustración: Modelo Tridimensional. *Enlace: [https://3dmodelsworld.com/downloads/coffee-cup-mug-3d-model/] Modelo tridimensional con forma de taza de café.*

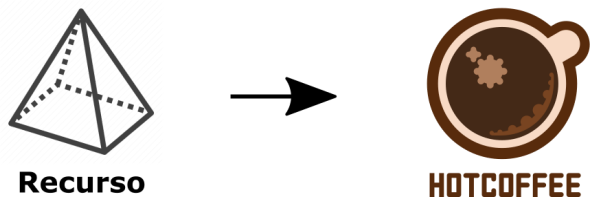
Ilustración: Textura para Modelo. *Enlace:*
[https://www.gamedev.net/forums/topic/634893-the-best-way-to-texture-a-3d-model/]
Textura utilizada para pintar el modelo tridimesional de un soldado.

Administración de Recursos

Resource Loader

Resource Loader tiene la funcionalidad de importar recursos del juego desde el disco. Se diseñó para ser un plug-in, de esta forma existe la

facilidad de cambiar las herramientas de importación de recursos en el futuro [3].



Resource Manager

Resource Manager administra y almacena los recursos que han sido importados desde el disco. Provee punteros inteligentes que apuntan a los recursos del juegos *[11]*.

Resource Allocator

Template abstracto que almacena recursos de un tipo. Utiliza un mapa desordenado para almacenar los recursos:

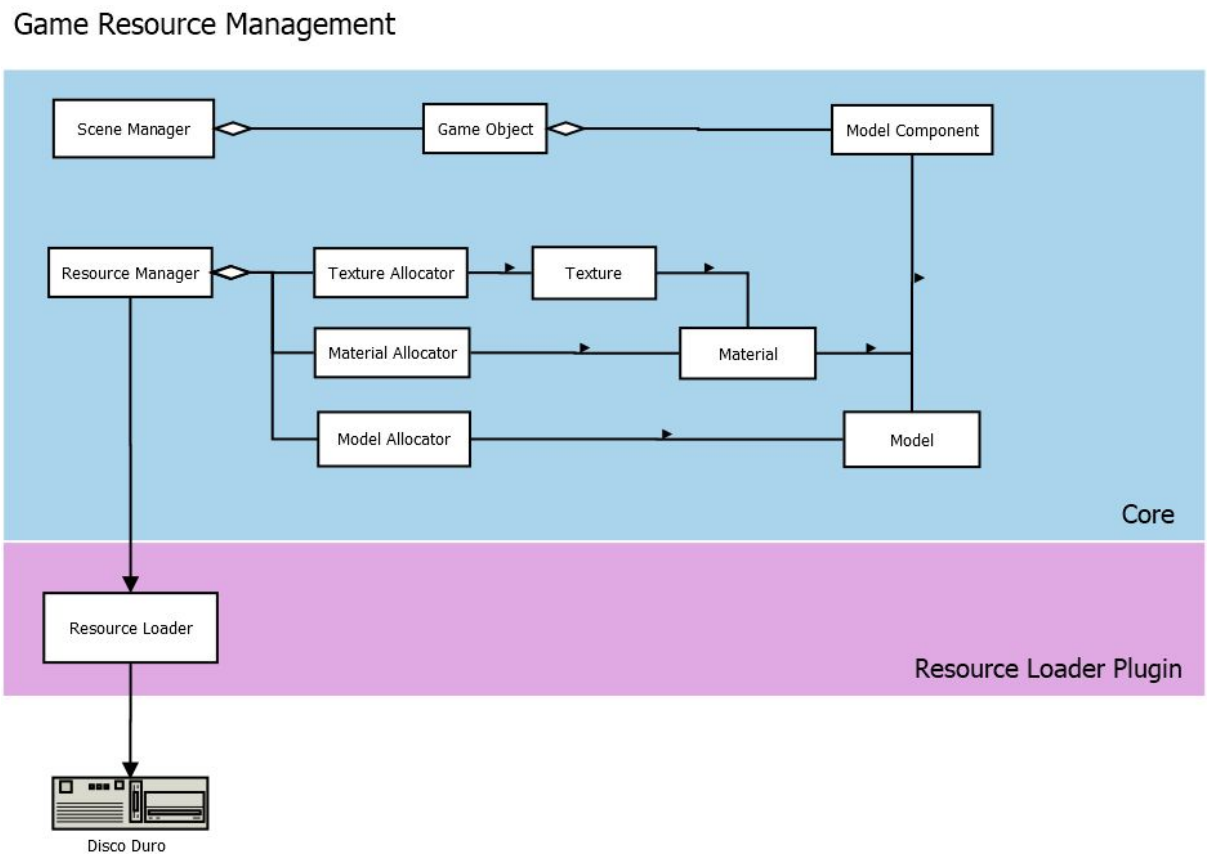
key: Resource URL
value: Resource Smart Pointer

En esta versión, existen tres tipos de alojadores de recursos: Texture, Material y Model [10].

Model Component

Componente que provee una interface de control con uno de los modelos del juego. Durante la fase de render, es el medio por el cual se dibujan los modelos. El Model Component es una clase que pertenece a GameObject [1].

Mapa de la administración de recursos



HotCoffee Design: Relación entre Resource Manager y Scene Manager. *Resource Manager aloja los recursos de una manera eficiente rápida, mientras que Scene Manager hace uso de dichos recursos.*

Carga de Modelos

Para cargar modelos es por medio de la interface del Resource Manager, que puede obtenerse por medio del HotCoffee API.

Para cargar el modelo se debe proveer la ruta del archivo, relativa a la carpeta de recursos que fué definida para el proyecto. Del mismo modo, para obtener un modelo previamente, se debe proveer la ruta del archivo.

Ejemplo:

```
// Obtener el Resource Manager
hcEngineSDK::WeakPtr<hcEngineSDK::ResourceManager>
p_resource_manager
      = HotCoffee::GetResourceManager();
```


```
// result handle
hcEngineSDK::OPRESULT::E result;

// Cargar el Modelo - Claire Model
result =
p_resource_manager->loadModel("Models/Claire_Redfield/claireredfie
ldout.obj");

// revisar resultado
if (result != hcEngineSDK::OPRESULT::kOk)
{
  // Error: Fallo al cargar el recurso
}

// Obtener el Modelo - Claire Model
hcEngineSDK::SharedPtr<Model> p_claire =
p_resource_manager->getModel("Models/Claire_Redfield/claireredfiel
dout.obj");

// Revisar puntero
if (hcEngineSDK::SharedPtr<Model>::IsNull(p_claire))
{
  // Error: Modelo no encontrado.
}
```

Carga de Texturas

Al igual que los modelos, y cualquier otro recurso, las texturas se cargan y acceden por medio del ResourceManager.

Ejemplo:

```
// result handle
hcEngineSDK::OPRESULT::E result;

// cargar textura desde archivo
result =
p_resource_manager->loadTexture("Models/Claire_Redfield/claireredf
```


```
ieldhead_d.tga");

// manage result
if (result != hcEngineSDK::OPRESULT::kOk)
{
  // Error: Fallo al cargar la textura
}
```

Creación de Materiales

En esta versión, los materiales aún no pueden cargarse desde archivos. Sin embargo pueden crearse desde el API. Los materiales son creados por medio del Resource Manager. Las propiedades, como las texturas y el Sampler State pueden ser editados.

Ejemplo:

```
hcEngineSDK::SharedPtr<Material> p_material =
     p_resource_manager->createMaterial("Claire_Hair");

p_material->addTexture
(
  hcEngineSDK::hcGraphicsSystem::TEXTURE_TYPE::kDiffuse,
  p_resource_manager->getTexture
  (
    "Models/Claire_Redfield/claireredfieldhair_d.tga"
  )
);
```

Asignación de Materiales al Modelo

Los modelos poseen varios meshes, por lo tanto, se debe indicar el índice del mesh al que se quiere agregar el material. El método para asignar materiales a los meshes regresan un OPRESULT que reporta el resultado de la operación.

Ejemplo:

```
// result handle
hcEngineSDK::OPRESULT::E result;

result =
p_claire->setMeshMaterial(p_material, 0);

// manage result
if (result != hcEngineSDK::OPRESULT::kOk)
{
  // Error
}
```

Creación de GameObjects y Componentes

Todos los GameObjects que pertenecen al juego son creados desde la clase SceneManager. Scene Manager guarda los GameObjects en un mapa desordenado:

key: Name

value: SharePtr<GameObject>

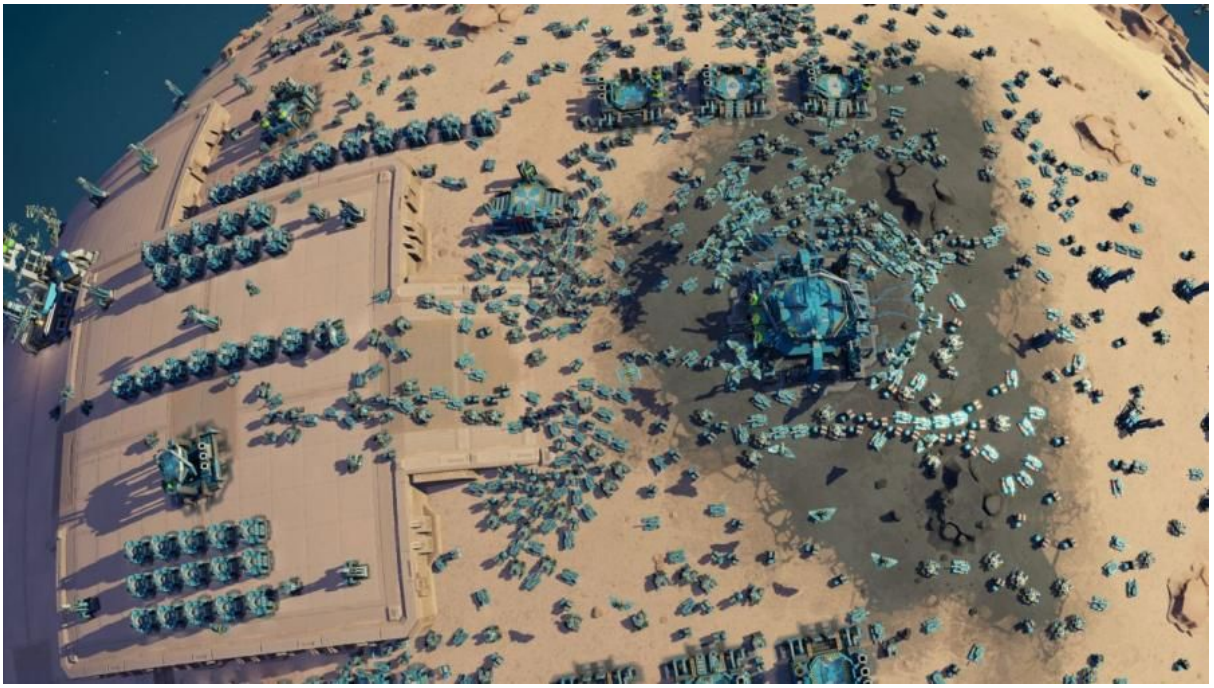
Los Componentes son creados desde el mismo GameObject, como se ilustra en el siguiente bloque de código:

```
// p_claire : SharePtr<Model> modelo previamente cargado.

WeakPtr<hcEngineSDK::GameObject> m_claire_go;
WeakPtr<hcEngineSDK::ModelComponent> p_model_cmpnt;

m_claire_go =
SceneManager::getSingletonPtr()->createGameObject("Claire_Agent");

p_model_cmpnt =
m_claire_go->createComponent<hcEngineSDK::ModelComponent>();
p_model_cmpnt->setModel(p_claire);
```


Game Scene Manager

Los GameObjects son toda entidad que ocupa un punto en el espacio tridimensional del juego. El mundo virtual está compuesto por GameObjects con características y comportamientos diferentes. Los GameObjects interactúan unos con otros y no es de extrañar que suelen buscar referencia a otro GameObject entre los múltiples objetos que existen [3].

La cantidad de GameObjects dentro de un entorno digital pueden llegar a ser bastante grande, por tal razón los algoritmos de búsqueda de objetos deben ser óptimos para encontrar entidades específicas entre todo el mundo virtual.

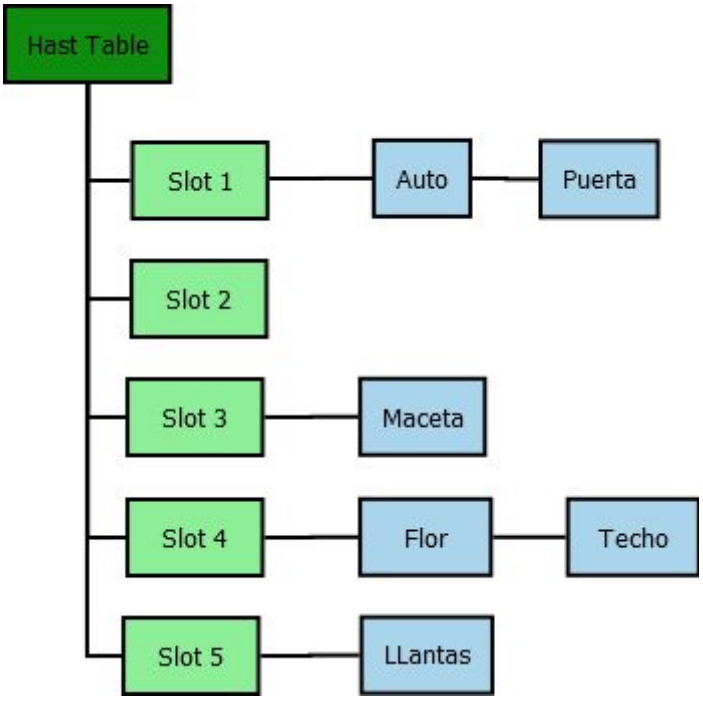


***Planetary Annihilation Titans: Planetary Annihilation Inc**. Enlace: [https://www.mmoga.es/Steam-Juegos/Planetary-Annihilation-TITANS.html] En esta escena se puede apreciar una abrumadora cantidad de objetos dinámicos en tiempo real (todos los pequeños bloques azules son vehículos en movimiento).*

Scene Manager

Cuando se trata de optimización y búsqueda de objetos, una simple lista ligada se queda muy corta. El HotCoffee : Scene Manager sigue la famosa táctica napoleónica *"Divide and Conquer"* .

Los objetos son separados en distintos grupos de para facilitar su búsqueda dentro del complejo entorno virtual. Scene Graph, toma el UUID (identificador único) de cada GameObject para realizar una operación de dispersión. Este algoritmo de dispersión es quien establece el grupo al que será asignado dicho objeto [13].

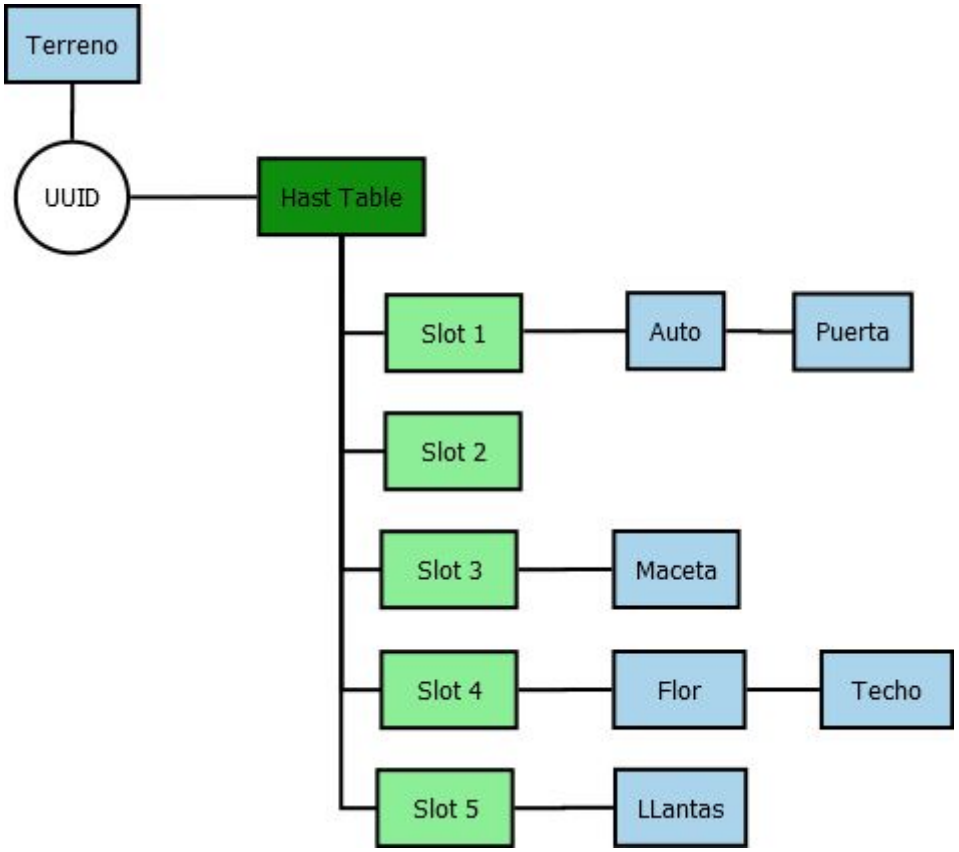


HotCoffee Design: Tabla Hash con elementos de la Escena del Molino. *Los GameObjects han sido distribuidos a lo largo de los diferentes grupos (Slots) que existen en la tabla hash.*

Tabla Hash

Scene Manager posee una hash table exclusivamente para la búsqueda de objetos. El objetivo de esta tabla hash es agilizar la búsqueda de los objetos en tiempo real. Las tablas hash sons estructuras de datos bastante eficientes y realizan una buena labor en búsqueda de información [13].

La tabla hash contiene diferentes slots donde almacena los objetos que ingresan a la escena. HotCoffee utiliza la UUID de los objetos como "llave" durante el proceso de dispersión .



UUID

Un UUID es un identificador único. Está compuesto por una serie de segmentos obtenidos de diferentes fuentes del ordenador [14]. Los segmentos son definidos en la tabla ***Composición UUID***.

Segmento	Dígitos Hexadecimales
time_low	8
time_mid	4
time_high_and_version	4
clock_seq_and_reserved_And_clock_seq_low	4
MAC Address	12

Tabla: Composición de UUID

Se tomará de ejemplo la UUID de la ilustración ***Ejemplo de UUID,*** con el fin de desglosar e identificar los diferentes elementos que componen al identificador.

c2f1a568-fd6e-46c9-95d7-e6aafd6d56bd

Ilustración: Ejemplo de UUID. *UUID.*

En la tabla ***Disección de UUID*** se desagrupan los distintos elementos de la UUID. Se puede visualizar a qué segmento pertenece cada parte de la UUID.

Segmento	Dígitos Hexadecimales
time_low	*c2f1a568*
time_mid	*fd6e*
time_high_and_version	*46c9*
clock_seq_and_reserved_And_clock_seq_low	*95d7*

MAC Address	*e6aafd6d56bd*

Tabla: Disección de UUID. *Identificación de los distintos segmentos de la siguiente UUID: "c2f1a568-fd6e-46c9-95d7-e6aafd6d56bd".*

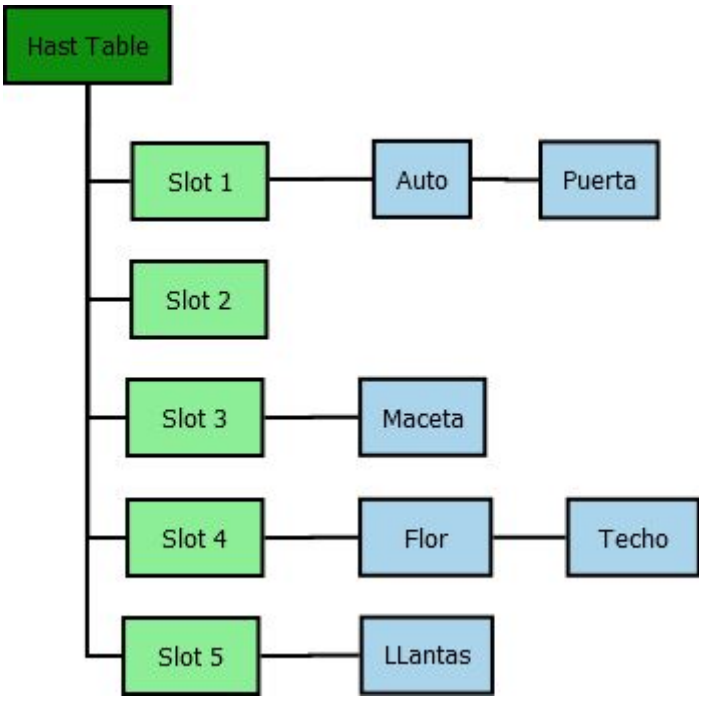
HotCoffee posee su propio generador de UUIDs llamado *UUID Generator*. Durante la creación de un nuevo GameObject, UUID Generator asigna un nuevo UUID al objeto.

Para finalizar, se debe tener presente que todo GameObject de HotCoffee Engine posee un UUID, el cual posteriormente es utilizado para las funciones de búsqueda e inserción en los contendores del motor.

Función de Dispersión : División

La función de dispersión consiste en un algoritmo que obtiene una cadena de datos y retorna una índice finito, correspondiente al identificador del slot de una tabla hash. Esto quiere decir que si una tabla posee *n* cantidad de slots, el índice no puede sobrepasar dicho tamaño pues indicaría un slot inexistente.

Tomaremos de ejemplo la tabla hash de la ilustración: ***HotCoffee Design: Tabla Hash con elementos de la Escena del Molino.*** La tabla hash posee un tamaño de 5 slots que pueden albergar objetos.



HotCoffee Design: Tabla Hash con elementos de la Escena del Molino. *La tabla hash posee un tamaño de 5 slots.*

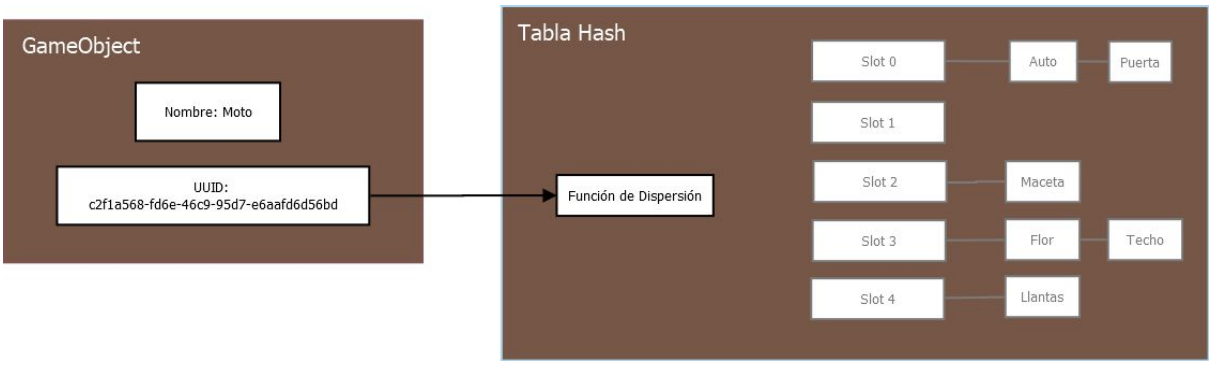
Sin embargo en programación, usualmente la enumeración de elementos no inicia con el número "1", sino con el número "0". Por lo tanto se puede decir que la tabla hash posee 5 slots numerados del "0" al "4". El "0" cuenta como una posición o índice de un slot. Véases ***Tabla: Numeración de Slots***.

Índice de Slot	
0	*Primer Slot*
1	*Segundo Slot*
2	*Tercer Slot*
3	*Cuarto Slot*
4	*Quinto Slot*

Tabla: Numeración de Slots. *Tamaño de 5 slots. Los slots son numerados del 0 al 4.*

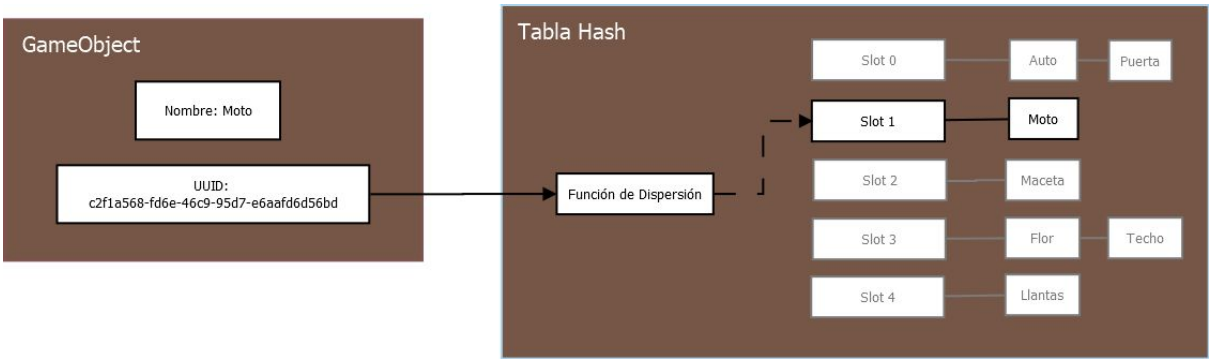
La función de dispersión obtiene una cadena de datos y a partir de ella genera una "Dirección" ó "Índice" de slot que corresponde a uno de los slots contenidos en la tabla hash. Retomando el ejemplo anterior, la función de la tabla hash sólo puede generar índices entre 0 - 4 respectivamente. A modo de ilustración, ejemplificamos el proceso de "dispersión" de un GameObject en la siguiente secuencia de ilustraciones.

Suponga que desea guardar el objeto "Moto" en la tabla hash. La Tabla Hash de HotCoffee Engine utiliza el UUID de los objetos para generar la "Dirección" correspondiente en la tabla hash. ***Véase Función de Dispersión: Fase 1***.



Función de Dispersión: Fase 1. *Se revisa el UUID del GameObject para obtener la dirección correspondiente.*

La función de dispersión obtiene la dirección del slot donde será almacenado dicho GameObject. ***Véase Función de Dispersión: Fase 2.***



Función de Dispersión: Fase 2. *El GameObject es almacenado en la dirección obtenida.*

Se espera que la función de dispersión reparta los objetos a lo largo de todos los slots lo más uniforme posible. Existen diversos algoritmos para dicho cometido. HotCoffee utiliza el algoritmo de "división".

El algoritmo de división es sencillo, pero práctico. La función de dispersión toma el UUID para generar un equivalente numérico. Este equivalente numérico puede ser un número muy pequeño o muy grande.

Por ejemplo, asumamos que el equivalente numérico de la UUID "*c2f1a568-fd6e-46c9-95d7-e6aafd6d56bd"* es el siguiente: "*15334862"*. Notará que el número obtenido es bastante grande. Si tomamos de ejemplo nuestra querida tabla hash anterior ***HotCoffee Design: Tabla Hash con elementos de la Escena del Molino,*** no podremos usar dicho número como "dirección", pues dicha tabla sólo posee 5 slots.

Para obtener un número entre 0 - 4 (recuerde la numeración en programación), existe el pequeño truco del "módulo". El módulo es un operador matemático que obtiene el residuo de una división.

Dividendo % Divisor = Residuo

Continuando con el ejemplo anterior, obtenemos el residuo de la división entre el equivalente numérico de la UUID y el número máximo de slots. El residuo de una división oscila entre 0 y (Divisor - 1).

15334862 % 5 = 2

El resultado es **2**, por lo tanto 2 es la dirección del slot donde será guardado el objeto "Moto".

Complejidad en Notación Big O

La eficiencia de una tabla hash depende de varios factores, entre ellos: el tamaña de slots de la tabla y la función de dispersión. Idealmente, la complejidad de inserción y búsqueda de una tabla hash es constante. ***Véase Notación: Mejor caso de complejidad***

$$O(1)$$

Notación: Mejor caso de complejidad. *La complejidad es constante. [13]*

Sin embargo, una mala implementación de la función de dispersión puede generar un número alto de colisiones en un mismo slot. A continuación tomaremos la ***Ilustración: Tabla Hash Ineficiente*** para ejemplificar un caso de ineficiencia.

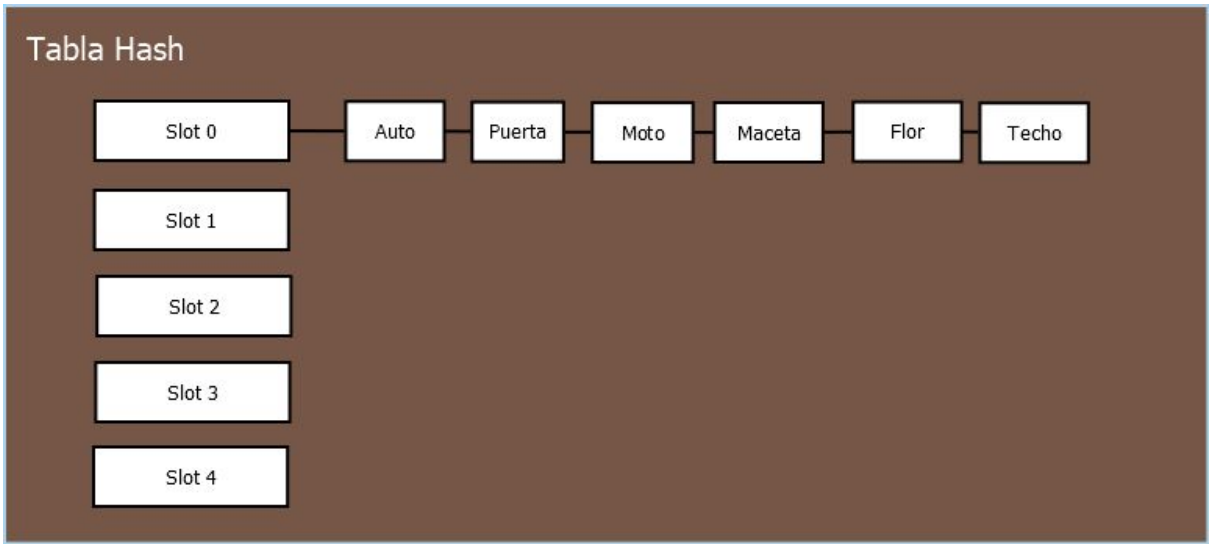


Ilustración: Tabla Hash Ineficiente. *Todos los elementos de la tabla hash están almacenados en el slot 0.*

La tabla almacena todos los elementos en el slot 0, por lo se han generado 6 colisiones durante la inserción. Las 6 inserciones se debe a los 6 elementos que fueron almacenados en un mismo slot. La función de dispersión no ha realizado una dispersión a lo largo de todos los slots, en

consecuencia tenemos una tabla hash que actúa como una simple lista ligada. Por lo tanto la complejidad de búsqueda ha cambiado por "Big O de n" Siendo "n" el número total de elementos en la tabla. ***Véase Notación: Peor caso de complejidad.***

$$O(n)$$

Notación: Peor caso de complejidad. *La complejidad es constante. [13]*

Conclusiones

Búsqueda de Objetos

Scene Manager almacena los objetos de manera eficiente por medio de una tabla hash. Esta permite una búsqueda eficiente de gameobjects en tiempo real.

Administración de Recursos

Por medio de nuestro Resource Manager, ahora es posible almacenar por categoría los recursos compartidos del juego. HotCoffee también es capaz de compartir recursos entre las entidades del juego Para ello se realizaron algunas herramientas como punteros inteligentes que direccionan a los recursos compartidos.

El motor puede administrar y compartir recursos entre los objetos del juego, de esta manera se evitan los recursos duplicados y por ende el mal uso de memoria. Ejemplificamos el uso de recursos de compartidos por medio de la ***Ilustración: Textura de Árbol Compartido,*** que simula la escena de un bosque.

La ilustración presenta cuatro GameObjects que requieren de la Textura "Árbol". En lugar de cargar en memoria cuatro veces la textura, se carga una sola vez y luego se comparte entre los objetos que la necesitan.

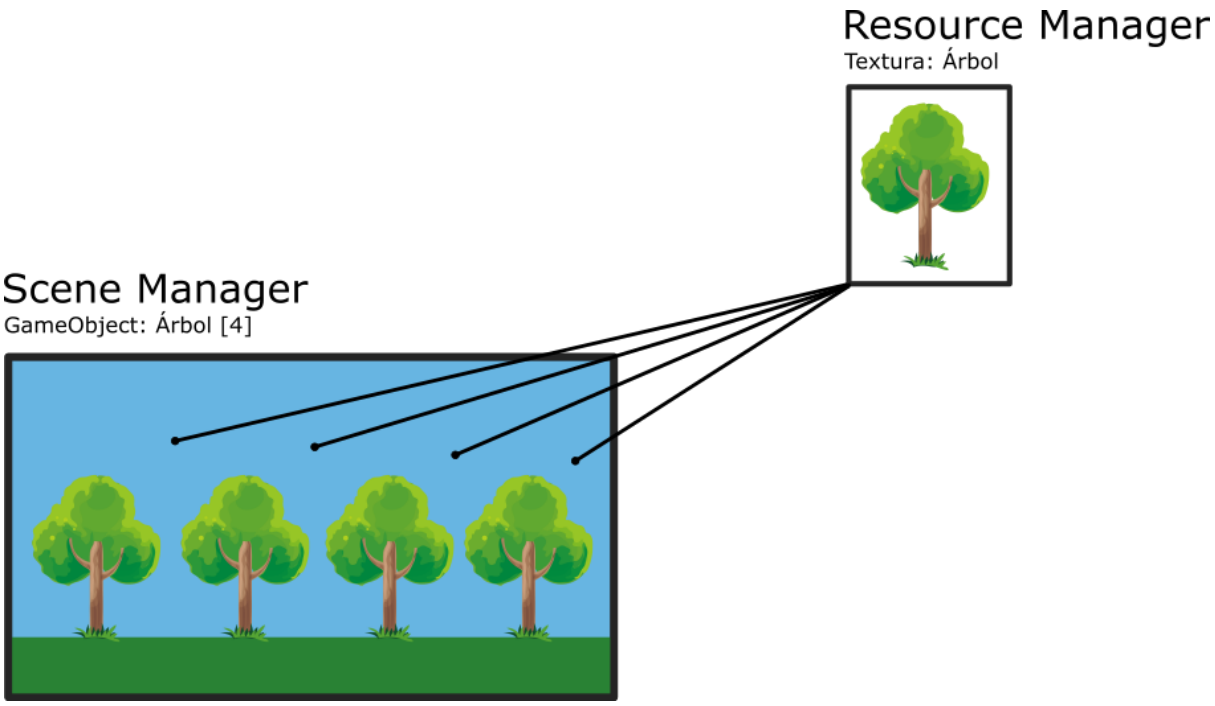


Ilustración: Textura de Árbol Compartida. *La textura se carga una sola vez, y luego se comparte entre todos las cuatro entidades que la necesitan.*

El desarrollador no tendrá que preocuparse por liberar los recursos cuando no los necesite, estos se autodestruirán gracias al Resource Manager y los Punteros Inteligentes.

Scene Graph

La relación "Padre - Hijo" entre objetos de la escena permite al motor realizar los cálculos necesarios para una correcta transformación de matrices. En el siguiente ejemplo ilustramos una combinación de recurso compartido con Scene Graph para generar un bosque. ***Ilustración: Escena del Bosque con Transformaciones***

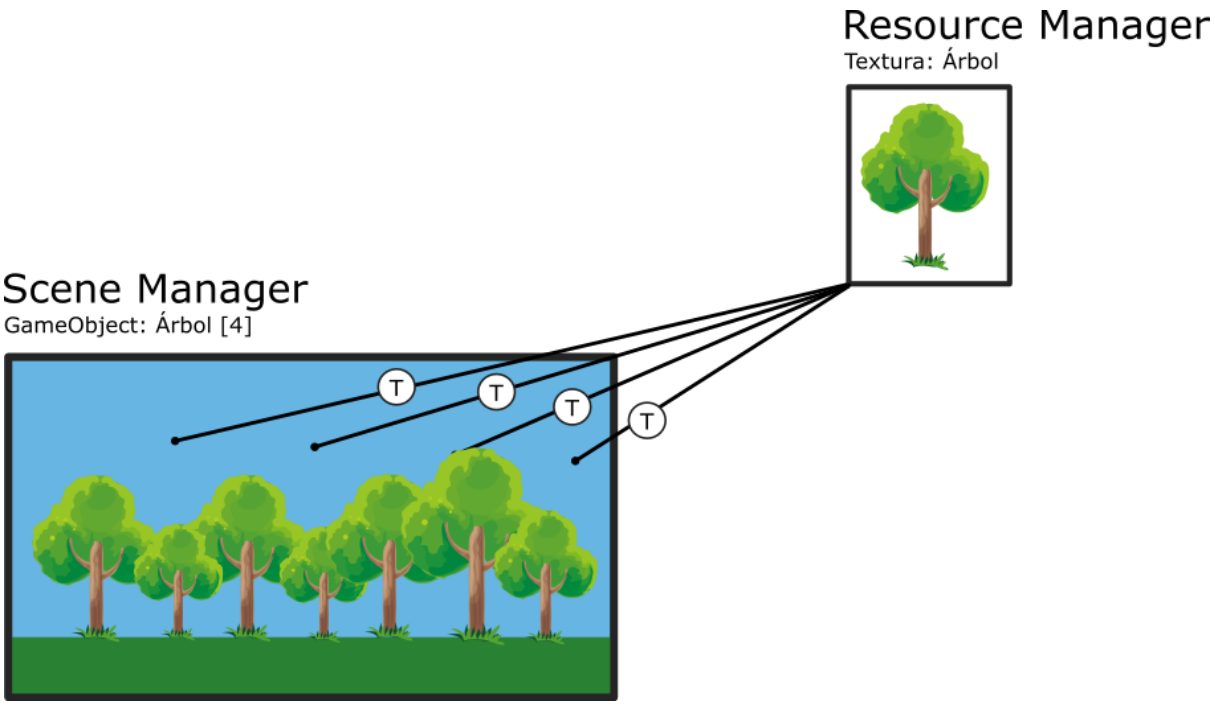


Ilustración: Escena del Bosque con Transformaciones. *Los objetos pueden compartir el mismo recurso, y al mismo tiempo aplicar diferentes transformaciones para aparentar árboles de diferentes tamaños.*

Bibliografía

[1] Andrei Alexandrescu. (2001). Modern C++ Design: Generic Programming and Design Patterns Applied. One Lake Street: Addison Wesley.

[2] Frank D. Luna. (2012). Introduction to 3D Game Programming with DirectX® 11. Boston, Massachusetts: MERCURY LEARNING AND INFORMATION.

[3] Tomas Akenine-Moller. (2008). Real-Time Rendering, Third Edition. Broken Sound Parkway: Taylor and Francis Group.

[4] Eric Lengyel. (2016). Foundations of Game Engine Development, Volume 1: Mathematics. Estados Unidos: Terathon Software LLC.

[5] Eric Lengyel. (2011). Matemáticas para Videojuegos en 3D. Estados Unidos: Cengage learning.

[6] What-when-how.com. (2019). *Scene Graphs (Advanced Methods in Computer Graphics) Part 2*. [online] Available at: http://what-when-how.com/advanced-methods-in-computer-graphics/scene-graphs-advanced-methods-in-computer-graphics-part-2/ [Accessed 12 Aug. 2019].

[7] Cs.princeton.edu. (2019). [online] Available at: http://www.cs.princeton.edu/courses/archive/spr11/cos426/notes/cos426_s11_lecture10_transform.pdf [Accessed 12 Aug. 2019].

[8] Angel, E. (2016). *Hierarchical Modeling and Scene Graphs*. [online] Cs.utexas.edu. Available at: https://www.cs.utexas.edu/users/fussell/courses/cs354/lectures/lecture13.pdf [Accessed 12 Aug. 2019].

[9] Torres, J. (2012). *Matrix Transformations*. [online] Openscenegraph.org. Available at: http://www.openscenegraph.org/index.php/documentation/knowledge-base/41-matrix-transformations [Accessed 12 Aug. 2019].

[10] Cleopesce, D. (2014). *A Resource Manager for Game Assets*. [online] GameDev.net. Available at: https://www.gamedev.net/articles/programming/general-and-gameplay-programming/a-resource-manager-for-game-assets-r3807/ [Accessed 12 Aug. 2019].

[11] Wells, R. (2018). *C++ Game Dev 9: Resource Management - that games guy*. [online] that games guy. Available at: http://thatgamesguy.co.uk/cpp-game-dev-9/ [Accessed 12 Aug. 2019].

[12] Anon, (2013). *Grafos*. [online] Available at: https://users.dcc.uchile.cl/~bebustos/apuntes/cc3001/Grafos/ [Accessed 14 Aug. 2019].

[13] Granada, U. (2013). *TABLAS HASH*. [online] Decsai.ugr.es. Available at: http://decsai.ugr.es/~jfv/ed1/tedi/cdrom/docs/tablash.html [Accessed 14 Aug. 2019].

[14] Rouse, M. (2005). *What is UUID (Universal Unique Identifier)? - Definition from WhatIs.com*. [online] SearchMicroservices. Available at: https://searchmicroservices.techtarget.com/definition/UUID-Universal-Unique-Identifier [Accessed 14 Aug. 2019].