Journal of Zhejiang University SCIENCE A
ISSN 1009-3095 (Print); ISSN 1862-1775 (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

High-extensible scene graph framework based on component techniques*

LI Qi-cheng‡, WANG Guo-ping, ZHOU Feng
(*Department of Computer Science & Technology, Peking University, Beijing 100871, China*)
E-mail: {lqc; wgp; zf}@graphics.pku.edu.cn
Received Apr. 24, 2006; revision accepted May 16, 2006

Abstract: In this paper, a novel component-based scene graph is proposed, in which all objects in the scene are classified to different entities, and a scene can be represented as a hierarchical graph composed of the instances of entities. Each entity contains basic data and its operations which are encapsulated into the entity component. The entity possesses certain behaviours which are responses to rules and interaction defined by the high-level application. Such behaviours can be described by script or behaviours model. The component-based scene graph in the paper is more abstractive and high-level than traditional scene graphs. The contents of a scene could be extended flexibly by adding new entities and new entity components, and behaviour modification can be obtained by modifying the model components or behaviour scripts. Its robustness and efficiency are verified by many examples implemented in the Virtual Scenario developed by Peking University.

Key words: Scene graph, Entity, Entity component, Behaviour script, Behaviour model
doi:10.1631/jzus.2006.A1247 **Document code:** A **CLC number:** O343.2

INTRODUCTION

With the tremendous growth of the digital entertainment industry in the past decades, the scale of game scene is becoming more gigantic. To effectively manage and organize the massive data, game engine usually uses hierarchical scene graph. Many popular 3D modelling tools, such as Maya, 3D Studio Max, use scene graph to manage the scene content (Sowizral, 2000). 3D modelling software—Houdini directly edits scene graph to create and modify the virtual environment. Most virtual reality systems, especially those distributed ones, adopt the architecture based on scene graph, for example DIVE (Frécon and Stenius, 1998), Repo-3D (MacIntyre and Feiner, 1998), Avocado (Tramberend, 1999). With the support of a lot of 3D Graphics Specification (VRML (Lukka, 1999), X3D (Brutzman, 2003)) and common 3D programme development library (SGI Open Inventor (Hartley, 1998), Java3D (Orr, 2001)), the scene graph has become one of the most popular ways in 3D applications.

In the virtual environment or the game, scene graph manages the scene contents or the scene objects according to the affiliation relationship, and is a directed acyclic graph (DAG). Its multifarious group-node affects or restricts its child graph. Each leaf-node is a container of the geometry object and the attribute object, with the geometry object encapsulating the geometrical data of the scene object, and the attribute object inflecting the rendering effect on the render environment of the scene object.

Our component-based scene graph takes each scene object as an entity where the content and the behavior are separated. The behavior is defined by two means: script or behavior model. Differing from the traditional scene graph in which various group-

‡ Corresponding author
* Project supported by the National Basic Research Program (973) of China (No. 2004CB719403), and the National Natural Science Foundation of China (Nos. 60573151 and 60473100)

nodes are utilized to perform different function element, with the function element of our scene graph being separated and encapsulated from the scene graph. By this data separation, the entity, the behavior model and the function element can be composable. In this way we can extend the system flexibly by adding new dynamic link libraries (DLL) without touching the system source code which bring numerous advantages in the construction of the scene or the virtual environment

RELATED WORK

The scene graph was first presented by Strauss and Carey (1992). One of the most famous scene graphs in commercial software is Open Inventor (Hartley, 1998) which is the 3D graphics program development library of SGI. Recently, with the development of Object-Oriented Design and the reusability technology in the software engineering field, the design of the scene graph tends to the "Kernel+ Plug-in" model so that the extensibility of the scene graph becomes one very important feature. The Kernel usually includes the hierarchical structure of the scene graph, mathematical library and basic graphics units. Plug-in usually provides the reading-writing, operation, and rendering of the complex object. OSG (Burns and Osfield, 2004) is a typical programming toolkit which uses the "Kernel+Plug-in" model. The kernel of OSG realizes a classical hierarchical structure of scene graph, and provides multifarious group-nodes to realize the space transform, fog, light, etc. The plug-in of OSG provides the conversion of file format, the rendering of complex object and special traversal of scene graph, and so on.

Another trend of scene graph design is to separate data from algorithms, because scene data is independent while the algorithms depend on a different development platform. BOOGA—a Component-Oriented Framework (Amann *et al*., 1997) for computer graphics is such a representative work. BOOGA is made of three layers: the Library Layer, the Framework Layer, and the Component Layer. The Framework Layer encapsulates the geometric object of 3D graphics, and the Component Layer is the combination of different components, each of which performs an operation on the data structures from the framework layer. The Component Layer encapsulates the operation algorithm of each object. BOOGA consists of ready-to-use components to support rapid application development. New components can be seamlessly integrated into the existing system. VRS (Döllner and Hinrichs, 2000; 2002) is another similar render system. VRS consists of graphics object, scene graph and behavior graphs. Graphics Objects represent shapes, graphics attributes, and non-graphics attributes. Behavior graphs describe event-dependent and time-dependent behavior of scene objects.

Behavior is also an important function of the scene graph. Java3D uses behavior node to realize animation, keyboard and mouse event response, movement, and selection. Open Inventor implements behavior by the call-back function of scene graph node. Both these two methods have their own disadvantages. While realizing a new behavior, Jave3D must increase a behavior node and Open Inventor needs to write a new call-back function.

ARCHITECTURE

Our scene graph framework consists of five parts: the scene graph kernel, the entity library, component library, behavior library and scene graph engine (Fig.1). The scene graph kernel can create leaf-node and group-node, realize basic function element and define the interfaces for all components. The entity library is the dataset of all entities. The initial entity library is null. The component library is the aggregate of entity components, behavior model components and function element components. The script library is the aggregate of the behavior script to which the entity can be loaded. Scene graph engine is made of

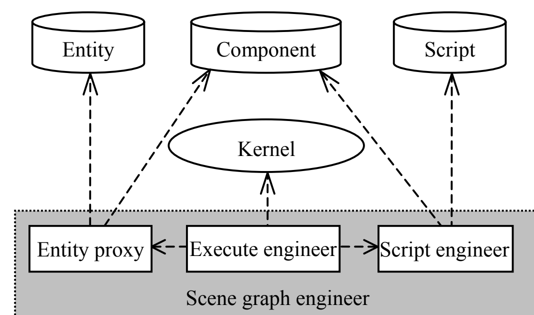

Fig.1 The architecture of our scene graph

three child engines: scene graph execution engine, entity proxy and script engine.

An instance of the scene graph is a hierarchical graph in which the kernel uses structural node to link the entity instance. Running a scene graph instance is such a procedure: the execution engine visits the entity proxy to get the entity data and all kind of components, and the script engine gets behavior script and translates it for execution engine according to the behavior script template engine from component library. The entity proxy maintains all the entities and components of current scene graph. When the scene graph needs a new entity and component, the entity proxy loads it from the entity library and the component library. If an entity and component is not active for a long time, entity proxy will release it.

Kernel

The element of the scene hierarchical graph is called the scene element, which includes structural node, function element and entity instance. The structural node responsibility is to organize the scene element. The function element offers operation for a structural node, and entity instance is the scene content. Scene hierarchical graph is a directed acyclic graph (DAG), and has only one root. Every scene element, except the root node, can be shared (Fig.2).

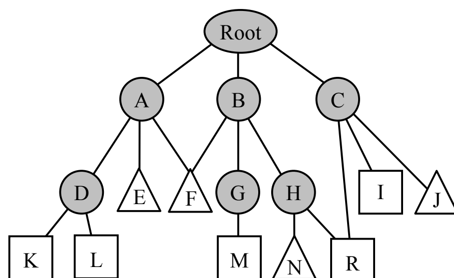

Fig.2 Scene hierarchical structural graph
A, B, C, D, G, H are structural nodes; E, F, N, J are function elements; K, L, M, R, I are entity instances. F is shared by A and B; R is shared by C and H

1. Structural node

Structural node is the container of scene elements. It has two kinds of nodes: group node and leaf node. The group node contains leaf node, and the leaf node contains entity instance. Both the group node and leaf node do not have any function, and just organize the scene according to affiliation. Structural node has several attributes, with its child tree inheriting these attributes.

2. Function element

The traditional structural node has various functions, and usually provides some operations, such as space transform node, light node, fog node, billboard node, switch node and so on. We separate these nodes from structural node, and encapsulate them as function elements. Structural nodes only organize the scene, and their function is realized by function elements. A structural node can link some new function elements, so that we can increase a new function element. All function elements inherit the same interface, with the interface having three functions: the before-traversal-function, the after-traversal-function, and the interaction handler function which responds to the keyboard and mouse event.

The advantage of separating function element from structural node is releasing the restriction of scene objects' functional affiliation. Consequently our scene graph is purely organized according to content affiliation. Since a group of scene elements share one function, conventional scene graphs take the functional node as the parent of these group nodes. For example, if a group of unassociated scene elements have the same lighting parameters, conventional scene graphs will take these group scene elements as the child of the light node. While such group scene elements in our scene graph will share the light node. Obviously, our scene graph architecture reduces the depth of the hierarchical structure, since we do not need to increase a parent node for every new function. In this way, it is more convenient for our framework to add new functions, while our scene graph can maintain a small kernel.

The behavior script and the behavior model

Our scene graph system realizes the entity behavior by two means: script and behavior model. The behavior script is executed when handling an event and the behavior model is a traversal method of scene graph. Behavior script should be translated by script engine; the behavior mode is a compiled DLL and can be executed directly. When the behavior script and the behavior model are executed, the input comes from the property list of the entity and the output edits the entity property.

1. Behavior script

A behavior script is a sequence of script commands, each of which corresponds to a function. The system defines all the script commands in the script template and the command function is implemented in the component. For the in-depth discussion of the definition of script criterion, please refer to (Adams, 2002).

2. Behavior model

The behavior model, for example, the physical simulation model, is to realize behaviors for every entity. The behavior model does not use script, but uses a compiled DLL-Behavior model component to realize behaviors. In the executing mode of behavior model, the required attributes can be dynamically added to the property list of the entity instance. These attributes which can be dynamically appended are called behavior model property aggregate. The behavior model only visits the property aggregate of the behavior model. For example, for a simple mechanical simulation behavior model, its property aggregate is mechanical parameters one, and the behavior model will compute the mechanical state for every entity instance according to their property aggregate of the behavior model.

Scene graph engine

Scene graph engine is made of three parts: entity proxy, script engine and execution engine, in which the execution engine is core, and takes charge of executing the scene graph instance, and the entity proxy and the script engine provide service for executing the engine.

1. Entity proxy

The entity proxy maintains the required entities and the entity components of the current scene graph instance. The operation of the entity proxy is to load and unload the entity components by use of the Reference Count Semantics to manage the life of entity components, and buffering the entity components by LRU algorithm to avoid dithering. The advantage of using the entity proxy to manage entity components is that entity components are transparent for scene graph execution engine.

2. Script engine

The function of the script engine is translating behavior script to executable code. The script engine installs the required script template components from components library. A script template defines a set of script commands which are encapsulated in a component. The script engine chooses a script template component according to various applications. During the execution of an entity instance, if it refers to behavior script, the script engine will be run to translate the referred script to executable code.

3. Execution engine

The task of scene graph execution engine is scene graph traversal implementation. The basic traversals in the scene graph are: update traversal, clip traversal and render traversal. The traversal in the scene graph is from update to clip and render orderly. Users can add a new traversal at an arbitrary position in the scene graph. The execution engine traverses the hierarchical graph from top to bottom and from left to right in the scene graph, with child nodes inheriting the attributes of parent, but brothers do not affecting each other.

The core of scene graph traversal is the rendering traversal. The render traversal manages the render environment and draws the scene with the entity's render function. The process of render traversal is that when a structural node is visited, the before-traversal-function elements with which this structural node link is executed, and then to traverse all children of the structural node; finally the after-traversal-functions are executed in reverse; when a entity instance is visited, the entity's render function is called upon to draw the entity. Fig.3 is an issue of render traversal.

EXAMPLES

Example 1: A simple 3D scene

The example shows how to create a 3D scene by using our scene graph framework. The virtual scene consists of rabbit, windmill, carriage and terrain, and we use structural node to link the various entities and form a hierarchical graph. We write a behavior script for the rabbit and windmill. The behavior script of rabbit is executed in the operation of handling event, and the behavior script of windmill is run in the update operation; the rabbit's behavior is jumping when it is clicked, and the windmill's behavior is to turn around along the *z*-axis. Fig.4 is the issue of forming the scene, Fig.5 is the behavior script of the scene and Fig.6 shows the snapshot of the virtual scene.

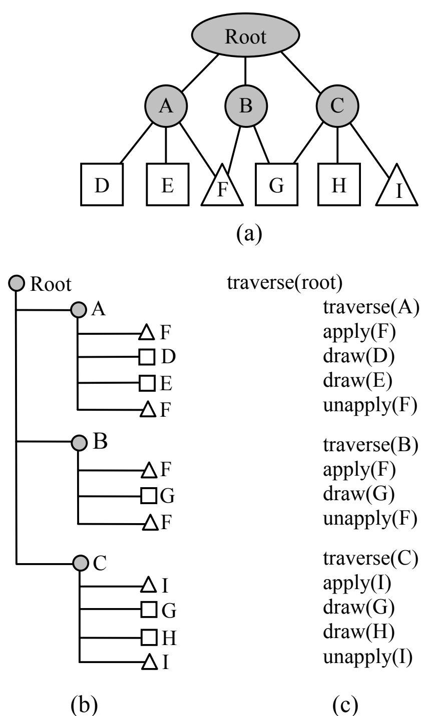

Fig.3 Issue of traversal scene graph. (a) A scene graph instance. A, B, C are structural nodes, F, I are function elements, and D, E, G, H are entity instances; (b) The expanding graph of render traversal; (c) The puppet code for rendering procedure. The functions of "apply" and "unapply" are applying function element and canceling function element, and correspond to the executing function before traversal child nodes and the executing function after traversal child nodes

```
// the code of forming scene
GroupNode root=new GroupNode;
LeafNode[A, B]=new LeafNode;
EntityInstance [D, E, F, G]=new EntityInstance;
FuncElement H=new LightElement;
root.append([A, B]);
root.append(H);
A.append([D, E, G]);
B.append(F);
D.attach("Terrain");
E.attach("Windmill");
F.attach("Rabbit");
G.attach("Carriage");
```

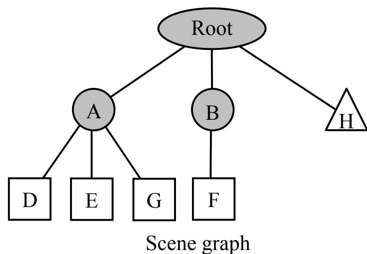

Fig.4 The issue of forming the scene

```
(1) The behaviour script of Windmill (E1)
/* The rotation angle add one every frame */
E1.rotation.axis=vector3f(0, 0, 1);
E1.rotation.angle=E1.rotation.angle+float(1);
(2) The behaviour script of Rabbit (E2)
/* if a clicked event happens, Set the velocity and acceleration property of rabbit */
If event.type==Mouse_Down Then
    E2.velocity=vector3f(1, 1, 0);
    E2.acceleration=vector3f(−0.01, −0.01, 0);
End if
```

Fig.5 Behavior scripts of the scene

Fig.6 The virtual scene

Fig.7 The race game

Example 2: A race game

Our scene graph framework can be taken as a mini game object-oriented game engine. We implement a race game to test the effect of our game engine. Based on the hierarchical tree-structured scene graph, our game engine supports the hierarchical view frustum culling by AABB-boundingbox for each node, and a triangle-based quadtree for polygon culling and terrain rendering (polygons are internally sorted by texture), and a hierarchical Z-buffer designed for hidden face removal. Furthermore, the functions such as collision, dynamic lighting and shadows are implemented.

The race game is run on a PC whose CPU is AMD Sempron 2500+, memory is 512 MB, and the graphics card is ATI9550. The scene has 22634 vertexes and 14316 faces, and the resolution of rendering window is 640×480, then the race game can run at the speed of 270 fps. Fig.7 (see page 1251) shows a snapshot of the race game.

CONCLUSION

A novel scene graph framework which can create a virtual scene conveniently is presented in this paper. We provide a flexible mechanism for the scene graph to extend content and behavior. As long as a new scene object is encapsulated and added into the entity library and the component library, it can join the corresponding system. Furthermore, we do not need to modify the source code for the new scene object. We also can dynamically increase the entity's behavior. If all the entities need a common behavior, we can add a behavior model to implement it. If an individual entity needs a new behavior or changes its behavior, we can implement this by writing a script or editing the script. The examples verify that our scene graph framework is extensible and efficient.

References

Adams, J., 2002. Programming Role-playing Games with DirectX 8.0. Premier Press, p.623-654.

Amann, S., Streit, C., Bieri, H., 1997. BOOGA—A Component-Oriented Framework for Computer Graphics. Graphicon'97, p.193-200.

Brutzman, D., 2003. X3D-edit Authoring for Extensible 3D (X3D) Graphics. Educators Program from the 30th Annual Conference on Computer Graphics and Interactive Techniques, ACM Press, p.1. [doi:10.1145/965106.965137]

Burns, D., Osfield, R., 2004. Open Scene Graph A: Introduction, B: Examples and Applications. IEEE Virtual Reality, p.265-265. [doi:10.1109/VR.2004.1310100]

Döllner, J., Hinrichs, K., 2000. A Generalized Scene Graph API. Vision, Modelling, Visualization 2000 (VMV 2000), p.247-254.

Döllner, J., Hinrichs, K., 2002. A generic rendering system. *IEEE Trans. on Visualization and Computer Graphics*, **8**(2):99-118. [doi:10.1109/2945.998664]

Frécon, E., Stenius, M., 1998. DIVE: A scaleable network architecture for distributed virtual environments. *Distributed Systems Engineing Journal (Special Issue on Distributed Virtual Environments)*, **5**(3):91-100.

Hartley, R., 1998. Open Inventor. Linux Journal, Content #53es.

Lukka, T., 1999. An Introduction to VRML. Linux Journal, Contents #57.

MacIntyre, B., Feiner, S., 1998. A Distributed 3D Graphics Library. Proceedings of SIGGRAPH'98, p.361-370.

Orr, G., 2001. An introduce to Java3D. *Journal of Computing Sciences in Colleges*.

Sowizral, H., 2000. Scene Graphs in the New Millennium. *IEEE Computer Graphics and Applications*, **20**(1):56-57. [doi:10.1109/38.814563]

Strauss, P.S., Carey, R., 1992. An object-oriented 3D graphics toolkit. *Computer Graphics (SIGGRAPH'92)*, **26**(2): 341-349. [doi:10.1145/142920.134089]

Tramberend, H., 1999. Avocado: A Distributed Virtual Reality Framework. Proceedings of IEEE Virtual Reality'99, p.14-21.