Journal of Zhejiang University-SCIENCE C (Computers & Electronics)
ISSN 1869-1951 (Print); ISSN 1869-196X (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

A component-based aircraft instrument rapid modeling tool*

Fang-wen LI†, Xu-kun SHEN

(*State Key Lab of Virtual Reality Technology and Systems, School of Computer Science, Beihang University, Beijing 100191, China*)

†E-mail: lfwredlum2004@163.com

Received Sept. 14, 2010; Revision accepted Oct. 8, 2010; Crosschecked Sept. 26, 2010

Abstract: Component-based software reuse and development is considered to be the best way to improve the efficiency and quality of development. To meet the demands of aircraft instrument exhibition in an aviation virtual library, we present a modeling and simulation tool for cockpit instruments on component methods, achieving display functionality and providing a visual interaction platform in the aviation virtual library. For the rapid model construction and reuse of virtual library resources, we classify entities in the cockpit into several categories by their behavior, and design a template for each category to describe its attributes. This efficiently reduces the modeling cycle, and thus largely helps users to take advantage of the library resource.

Key words: Components, Rapid modeling, Simulation
doi:10.1631/jzus.C1001010 **Document code:** A **CLC number:** TP311

1 Introduction

Aviation technology has become the most active high-tech field in the 21st century. The field is an important symbol of a country's national power (Zhang *et al*., 2009). With the development of computer and storage technology, the aviation virtual library becomes the best option because of its capacity and convenience. The idea of building a typical immersive virtual environment in an aviation virtual library has become a reality.

An aviation virtual library includes the digitization of the physical library's displays. Users can browse all the collections and learn the knowledge structure of aviation technology. An aviation virtual library can also provide visual interaction to the public. To exhibit aircraft instruments in the aviation virtual library, first we need to digitalize them into 3D models.

Currently, the most popular way to model the exhibit is using 3D modeling software (Bruce *et al*., 1998; Alm, 2001; Robinson *et al*., 2004; Park *et al*., 2008; Wang *et al*., 2009); that is, the display of the cockpit and the control system is modeled by modeling tools. The related actions are added to the model to implement the simulation for users.

The most common instrument modeling tools are: (1) Multigen-Paradigm Creator, with a special instrument panel module (Multigen-Paradigm, Inc., 2001; 2004a; 2004b; 2004c; 2004d) that enables automatic creation of various common instrument templates, such as dial gauge, vertical gauge, cylindrical gauge, and the pitch ladder. The reuse of the instrument's unit level design solution is efficient, but does not provide system-level reuse or integration technology of the simulator, which results in a longer design cycle. (2) DiSTI GL Studio (DiSTI, 2009a; 2009b) is a platform-based rapid prototyping tool. It can make realistic virtual aircraft instruments, but the software is not flexible enough. It does not have its own graphics editor, and requires users to have a certain programming background. (3) Aviation Instrument Control developed by GMS (Global Majic Software, http://www.globalmajic.com/), which uses VB, VC, and other languages, generating 18 general aviation instruments including ADI (altitude direction indicator), altimeter, and compass.

* Project supported by the National High-Tech R & D Program (863) of China (No. 2009AA012103) and the State 211 Project of China (No. 201003005)

However, the picture is not textured or realistic, and cannot produce meters other than the 18 general aviation instruments.

Using the instrument modeling tool mentioned above we can achieve the 3D digital modeling of the cockpit, but the consideration of modeling versatility and softness is poor, resulting in a longer modeling cycle, and an inconvenience while building the virtual library. Moreover, since the internal structure of the aircraft (such as cockpit modeling and instrument behavior simulation) is more complex, a 3D display is not enough.

To solve these problems, our research deals with the method of instrument modeling, based on the special demands of an aviation virtual library, by combining domain knowledge and component technologies, classifying entities in a common cockpit design process, designing a component-based model representation for entities, and establishing appropriate templates for each of them. Users can use predefined templates to complete a simple figure modeling and interactive behavior logic modeling. In addition, we design a simulation tool that can exhibit the modeling results and provide visual interaction to the users. Compared to the same type of instrument modeling method, this system is based on components rapid modeling and the reusability, and the simulation reality is improved. This approach is much more convenient, and thus effectively reduces the modeling cycle.

2 Component-based modeling method

A component is a closed and independent reusable package that provides specific functions. It is a new software developing method following the modular, structured, and object-oriented development technology (Le *et al*., 2001; Xi and Yuan, 2009). The combination of domain knowledge and component technologies can eliminate repetitive tasks, thereby reducing development costs and improving development efficiency. At the same time, through high quality reuse of existing resources, the errors introduced by re-development is avoid, and the software quality is improved (Le *et al*., 2001; Xi and Yuan, 2009).

While building a component-based system, first we need to divide the system into several functional modules, select functions to achieve the required components, and finally put these components together to construct the target system.

Among these tasks, the classification and properties design of the entity model is the foundation of the component model development. Establishing individuals inconsequently may cause subsequent development repeat or omission, making the entity model analysis the most important part in simulation software development for virtual library construction.

2.1 Related definitions

First, we define several concepts before the library's entity attributes are introduced.

The entity model of the cockpit is the abstraction of instruments and various controllers. According to the different ways of expression, it can be divided into two classes—static entity and dynamic entity.

Definition 1 (Static entity) An entity that does not require an operation after simulation begins, e.g., dial, shell, and scale.

Definition 2 (Dynamic entity) An entity that can interact with users and requires real-time updates throughout the operation process, e.g., needle and button.

Due to the characteristics of the domain knowledge, in this study we divide the entity property into custom attributes, describing the component itself, and interactive attributes, describing the interaction between components. Custom attributes include geometric and physical attributes; interaction attributes can be further hierarchically broken down into behavioral and action attributes. Fig. 1 shows the assembly structure of the components, and the concepts are defined in the following.

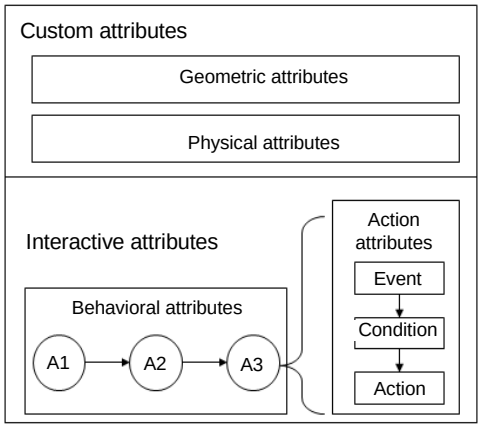

Fig. 1 Structure diagram of the components

1. Custom attributes

Custom attributes are used to customize the functions of the entity, as a single part, not affecting the globality of the software, similar to the concept of component parameters.

Definition 3 (Geometric attribute) An attribute that describes the geometrical information of the entity, such as vertices and texture.

Definition 4 (Physical attribute) An attribute that describes the basic behavior of a dynamic entity, such as the value of the button's displacement and the way the odometer changed.

2. Interactive attributes

Interactive attributes describe the interaction and communication between entities in order for achieving the coordination between entities, similar to the method of components.

Definition 5 (Behavioral attribute) An attribute that describes the entity's changing process in simulation.

Definition 6 (Action attribute) An attribute that describes the entities' changing sequence generated by their own state over time, which describes the interaction between entities.

2.2 Description of entity

To implement the component-based model, the method this research adopts is described as the following.

Classify the entities that are commonly used in the design of a cockpit, such as Table 1.

Table 1 Entity classification

Entity	Classification
Static	Line, rectangle, circle, sphere, rectangle scale, circle scale, light source, viewport
Dynamic	Needle, button, knob, switch, thumb, odometer, mutex group

The design provides an interface for each entity. Multiple entities can be implemented by the same interface. Interface design methods are shown in Fig. 2.

Interface is the interaction mechanism among the entities that provide the external interaction. Each entity interface contains one or more attributes. Due to the fact that the geometrical and interaction attributes between the entities are the same but the physical attributes are different, we design a corresponding template for each category to describe its physical attributes.

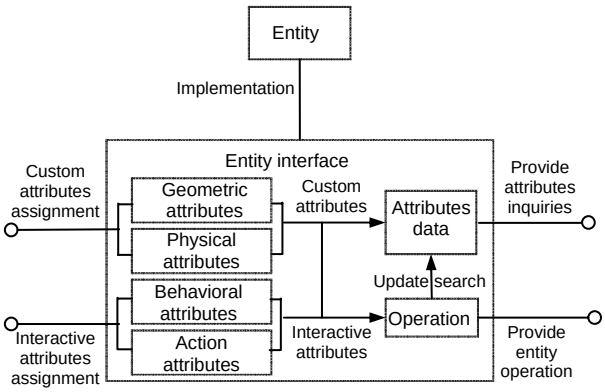

Fig. 2 Entity's interface design

2.2.1 Realization of custom attributes

1. Geometric attributes

The common figure in the modeling process can be simply summed up as a line, rectangle, sphere, etc. The system allows users to interact via a mouse to draw a common figure, and supports the texture mapping function to map images on the figures.

The system contains interfaces that are used to change the way by which the texture is mapped to the figure. The texture can be scaled up and down and rotation of the texture can be adjusted, and buttons are provided, allowing the user to set the texture to its actual size, or to reset the texture placement fields, etc. This can improve the efficiency and the authenticity of the modeling results.

The figure modeled by this system can be stored in OpenFlight format. OpenFlight is used for storing data and image information of the spatial data format, through geometrical figures, levels, and attributes to describe a 3D object; it is thought to be the industrial standard of spatial data format in the field of virtual reality. The description of geometric attributes must accord to the relevant form.

2. Physical attributes

Physical attributes describe the dynamic behavior of the basic entity. Fig. 3 shows an altimeter that contains the dynamic entity we define in our system.

Each dynamic entity has an attribute for describing the current state called ValueOut. The various dynamic entity description methods are introduced below.

(1) Push button. In the cockpit, generally buttons can be pulled or pressed; some special ones need to be rotated after pull or press. Therefore, in the template design process, the situation of

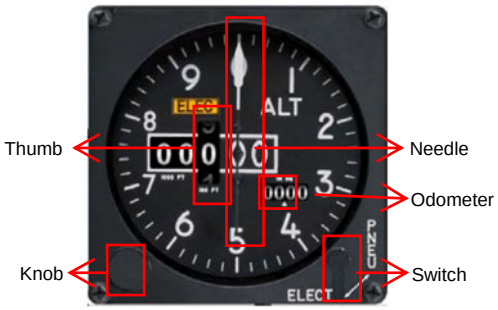

Fig. 3 Controllers of the altimeter

translation and rotation needs to be considered. Templates for buttons can be described as follows:

BTN={<Type>, <Axis>, <Value>, <Continuous>, <Angle>, <Orient>, <Detent>, <Toggle>, <ValueOut>}.

Type: whether the button needs to rotate after pull or rotate.

Axis: displacement or rotation direction along *x*, *y*, or *z* axis.

Value: displacement size of the button's pull or push status.

Toggle: when this option is checked, the button state changes only when the button is clicked; when unchecked, the button goes down when the mouse is pressed down, and returns to the up state when the button is released.

Angle, Orient: rotation angle and direction, respectively.

Detent: the number of detents or stops the button has.

Continuous: if this field is checked, the button can be rotated in either direction any number of times.

(2) Knob, Switch, Needle. Although Knob, Switch, and Needle have a significant difference in appearance, their physical attributes are basically the same, including only simple rotation operations. The description is nearly the same as that for button:

Knob/Switch/Needle={<Axis>, <Angle>, <Orient>, <Detent>, <Continuous>, <ValueOut>}.

(3) ThumbWheel. The ThumbWheel object simulates a thumb wheel found in different instrumentations. ThumbWheel objects must be used in conjunction with an odometer for proper use. At runtime, the ThumbWheel will provide mouse-dragging interaction ability that will increase or decrease the odometer according to the direction of the mouse drag. Its attributes can be described as

ThumbWheel={<Axis>, <Orient>, <Detent>, <TypeOfBehaviors>, <ValueOut>}.

The first three are the same as those for button.

TypeOfBehaviors sets the default behavior for the ThumbWheel. If set to Analog, the digits roll to the next value. If digital, each digit snaps to the next value (Fig. 4).

(a) (b) (c)

Fig. 4 Analog and digital thumbwheels
(a) Initial state; (b) Analog thumbwheel; (c) Digital thumbwheel

(4) Odometer. Odometer objects provide a method of creating analog or digital odometers or other instruments having described number thumbwheels that rotate within the instrument. The templates are described as

Odometer={<MaxValue>, <MinValue>, <InitValue>, <BaseSystem>, <TypeOfBehaviors>, <WheelList>, <OrderOfMagnitude>, <RolloverThreshold>, <ValueOut>}.

MaxValue, MinValue, InitValue: the largest and smallest values allowed to be displayed in the odometer, and the initial value displayed in the odometer object at run time, respectively.

BaseSystem defines the base system for this odometer. The default is decimal (base 10). Some other bases are also supported, e.g., base 8 which is often used in aircraft transponders.

WheelList: the list of the thumbwheels that the odometer contains.

OrderOfMagnitude: this read-only field describes what digit value will be output.

RolloverThreshold: this read-only value defines when each following digit rolls over to the next digit value.

Using these methods, we can describe every odometer rotating behavior. As per an odometer

shown in Fig. 5, units and tens digits remain unchanged; others are turntable during simulation. We can describe the attributes of the odometer as follows:

By the right thumbwheel's position it can be inferred that the InitValue is 59 375, TypeOfBehaviors is Analog, BaseSystem is decimal, OrderOfMagnitude is 5, and the RolloverThreshold is 100.

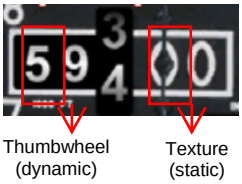

Fig. 5 Odometer on the altimeter

2.2.2 Interactive attributes

1. Behavioral attributes

The behavioral attribute, as an atomic property of the entity, is the most important to be considered in the simulation process. We can set a parameter (property) for each entity to describe the behavioral attributes, through the parameter that describes the entity resulting from actions, and directly search the component library for components.

Property={<Type>, <Name>, <InitValue>,
<ObjName>, <MethodIdx>}.

Type, Name, InitValue, ObjName: data type, name, initial value, and entity name, respectively.

MethodIdx: the corresponding method parameters. There are 24 methods designed in the system, which are commonly used in the flight simulation process, such as translation, rotation, scaling, changing the physical location, changing the physical center of rotation, and setting the dynamic physical state (ValueOut). Users can choose among these methods according to their need to operate entity and achieve the flight simulation.

Take the altimeter shown in Fig. 3 for example. First, we need to model the central needle (referred to as 'AltiNeedle'), select the needle template, and set its property. To control the AltiNeedle's rotation, we use a property named 'altitude' to indicate the degree of 'AltiNeedle', and set its InitValue to 0. ObjName is selected as 'AltiNeedle', and MethodIdx is selected as 'Set dynamic entity state'. Then we can join 'altitude' and 'AltiNeedle' together, and the needle's rotation can be controlled by 'altitude'.

2. Action attributes

Action attributes describe an entity's input and output attributes, reflecting the interaction between the different entities of the cockpit. Interaction generally requires two entities: the trigger entity and the receiving entity. The trigger entity produces active behavior and sends interaction information. Through the receiving entity, one can simulate the interaction between the trigger and receiving entity

We referenced database ECA (event-condition-action) rules to indicate the event and conditions needed for the action. We imitate the workflow engine Java Business Process Management (JBPM)'s graphical process designer, and use a workflow engine to control the behavior of the objects sequence, thereby describing the behaviors of various control devices and simulating the real cockpit environment.

3 Component library

To facilitate system management and rapid modeling, we need to establish the database of components to manage various entities effectively. The structure is as shown in Fig. 6. It consists of five components: container class, base class, template class, entity class, and composite entity class.

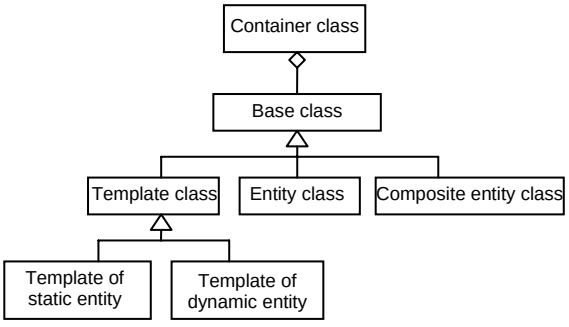

Fig. 6 Component library structure

The container class is a factory to generate specific entities which are used for storage entity management tools to provide addition, deletion, and modification operations for entities.

The base class is the prototype of the entity. Other types of entities that can improve the modeling efficiency and flexibility derive from this model.

The template class contains the description templates for buttons, knobs, and other entities mentioned previously. Template class, including static and dynamic entity templates, is stored in the component libraries, and cannot be modified or deleted by the user.

The entity class is a custom class generated according to user requirements. It can be replaced, modified, or added to other operations.

The composite entity class is the combination of several entities. In the modeling process, a combination of certain entities is often used, and encapsulating those as a composite entity can improve the development efficiency, making the whole structure more trenchant.

The component library operating flow is as shown in Fig. 7. In the modeling process, the entity container loads and edits the models from component libraries, and at the same time the corresponding template is loaded, editing and saving component's attributes. While rendering, first we need to load models from component libraries, then we can combine entities with template matching searching based on the input and output attribute values or relevant behaviors of the entities, automatically searching out related entities, and then we refine the function of them to effectively carry out a scenario simulation for the users.

The way of rendering based on component libraries will be greatly improved. An automatic extraction behavior is helpful to improve the entity design efficiency and make the development procedure more flexible.

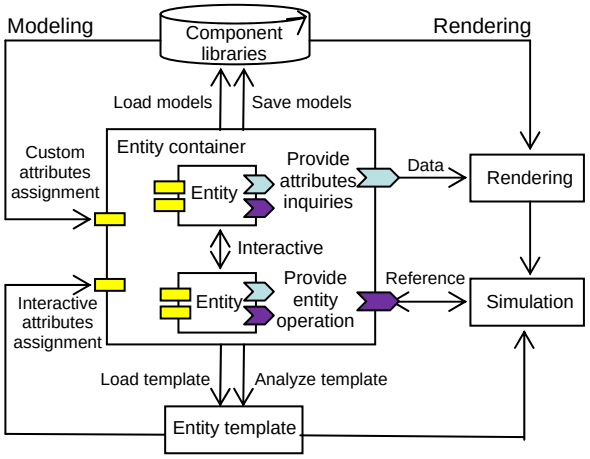

Fig. 7 The operating flow of component libraries

4 System implementation

The component is a reusable package that provides a specific function. This study deals with the design and implementation of a rapid modeling tool for aircraft instruments based on the principles referenced above. The system allows users to take a 'WYSIWYG' approach to modeling, and the model can be set to appropriate attributes or behaviors for scenario simulation.

The design flow is as shown in Fig. 8. First, the instrument designer can use the modeling tool to model the instrument and set the behaviors of the instrument. The result of the modeling can be stored in files. Using the simulation tool, users can view the modeling results.

As Fig. 9 shows, the system includes a modeling tool and a simulation tool. Here we use an example of altimeter application simulation development to specify the system development process and the use of various tools.

The final effect of altimeter development (Fig. 3) requests the following functions while rendering: the needle value and odometer's readings should be the

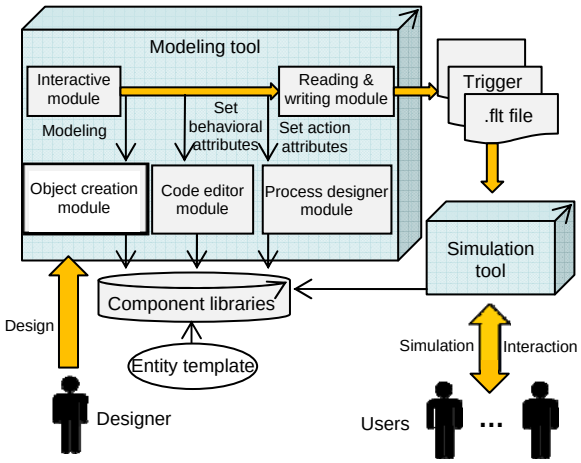

Fig. 8 Design flow based on a component library

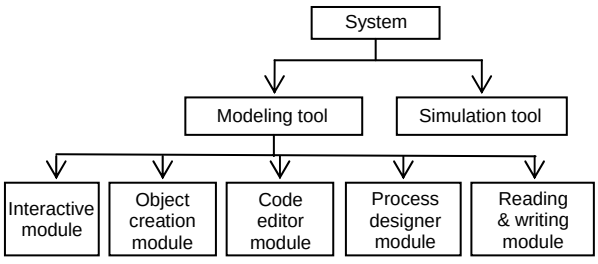

Fig. 9 System structure of our tool

same, and the barometric height adjustment knob and the electrical/PNEU switch knob should provide mouse-dragging interaction ability. First, we need to use the modeling tool to model the altimeter and set its behavior.

4.1 Modeling tool

The modeling tool can be used to model the instrument's appearance and logic behavior. The development process is as shown in Fig. 10.

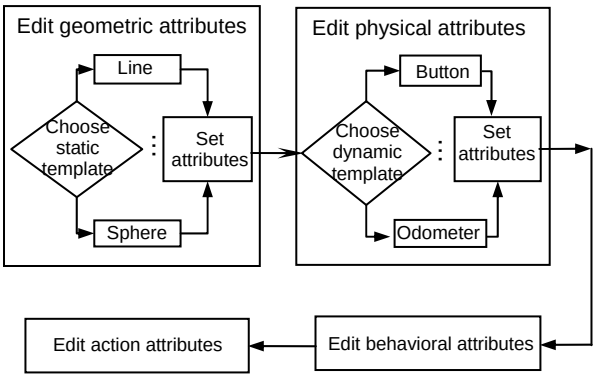

Fig. 10 Modeling tool's development process

First, a user can select the appropriate static or dynamic entity template to generate the figure, and then edit its settings to complete the instrument's appearance and basic behavior modeling. After that, through the code editor module and process designer module, the user can set corresponding entity behavioral attributes and action attributes to achieve logic simulation.

The modeling tool includes the following five modules:

1. Interactive module: a human-computer interaction interface for operation, providing object picked up, scene navigation, entity stretching, and other operations.

2. Object creation module: including two classes of objects, static and dynamic. The static entity can be created through choosing entity templates then placing and modifying geometry in the rendering window. The dynamic entity can be created by selecting corresponding static entity and modifying its attribute to complete the dynamic behavior design.

3. Code editor module: through this module we can set corresponding entity attributes and behavior, to realize logic simulation.

4. Process designer module: a graphical tool for authoring behavior of object sequence. It can express simulation processes graphically in terms of behavior, wait states for asynchronous communication, timers, automated actions, etc. To bind these behaviors together, it has an extensible control flow mechanism. This enables a smooth transition from process modeling to the practical implementation.

5. Reading and writing module: providing two functions, loading and saving files. The 'load file' function can load the Openflight format model and the scene layout file (xml format) that describes the layout of the scene. The 'save file' function can store the model to Openflight format, and the corresponding attribute of the model will be stored into a trigger file. The writing module can also save the scene layout file.

Using the modeling tool to model the altimeter, first we need to edit a static object. One selects a rectangle template in the toolbar at the left side of the window, places and modifies the rectangle with the mouse through the canvas, loads the altimeter dial's texture, and adjusts the position of the texture to adapt the size of the rectangle to complete the dial production. One repeats the steps above to produce needle, knobs, switches odometer, and other objects. The results are shown in Fig. 11.

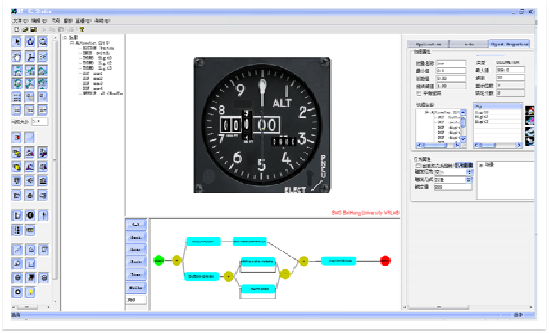

Fig. 11 Static entity modeling results

After editing the static objects, we need to edit the altimeter's dynamic behavior of its controls. One selects the needle and the needle template, sets the corresponding attribute (Fig. 12a), and then repeats the steps. The knobs, switches, thumb wheels, and odometer's attributes are set as shown in Figs. 12b–12e.

When the modeling process is complete, one clicks the 'generate code' button, and can generate a trigger file which stores the behavior set for the altimeter.

(a) (b) (c) (d) (e)

Fig. 12 Dynamic entity setting for needle (a), button (b), knob (c), thumbwheel (d), and odometer (e)

4.2 Simulation tool

After editing and setting the behavior of the model with the modeling tool, we can use the simulation tool to render the scene. This tool can load the scene file and extract models from the component libraries while rendering, can read relevant entity triggering document templates, and can simulate the dynamic behavior according to user's setting behavior.

For example, Fig. 13 is the simulation tool's rendering results of the altimeter. It can be seen that the needle and the odometer's readings are the same. This can also be achieved with knobs, switches, and other controller simulation.

(a) (b) (c)

Fig. 13 Simulation results

(a) Needle and odometer's readings are the same; (b) Button knob's initial state; (c) Button knob's rendering result

5 Conclusions

This paper introduces a component based aircraft instrument modeling tool. It uses a software reuse method in the cockpit, classifies entities in the cockpit design process, makes a component-based model representation for entities, and establishes appropriate templates for each of them. With the simulation tool, users can preview the modeling result and interact with models, thus effectively achieving rapid modeling. The tool provides an immersion display platform for an aviation virtual library. Compared to the traditional modeling methods, the reusability, reality of display, and modeling complexity are significantly improved, helping library users quickly obtain useful and realistic resources. This could also be used in cockpit design in the future.

References

Alm, T., 2001. How to Put the Real World into a 3D Aircraft Display. Int. Conf. on People in Control, p.223-227. [doi:10.1049/cp:20010466]

Bruce, S., Rice, C., Hepp, R., 1998. Design and Test of Military Cockpits. IEEE Proc. Aerospace Conf., **3**:5-14. [doi:10.1109/AERO.1998.685676]

DiSTI, 2009a. GL Studio Toolkit 4: C++ Tutorial, Version 1.0. USA.

DiSTI, 2009b. GL Studio Toolkit 4: User's Manual, Version 1.0. USA.

Le, J.J., Guo, R.Q., Zhu, S.Y., 2001. The origin and status of middleware as well as our opportunities. *Comput. Appl. Software*, **18**(11):1-4 (in Chinese).

MultiGen-Paradigm, Inc., 2001. OpenFlight API User's Guide, Version 2.5. USA.

MultiGen-Paradigm, Inc., 2004a. Creating Models for Simulation. USA.

MultiGen-Paradigm, Inc., 2004b. Creating Terrains for Simulation. USA.

MultiGen-Paradigm, Inc., 2004c. Creator Documentation. USA.

MultiGen-Paradigm, Inc., 2004d. MultiGen Creator User's Guide. USA.

Park, H.S., Choi, H.W., Park, J.W., 2008. Augmented Reality Based Cockpit Module Assembly System. Int. Conf. on Smart Manufacturing Application, p.130-135. [doi:10.1109/ICSMA.2008.4505627]

Robinson, S., Nance, R.E., Paul, R.J., Pidd, M., Taylor, S.J.E., 2004. Simulation model reuse: definitions, benefits and obstacles. *Simul. Model. Pract. Theory*, **12**(7-8):479-494. [doi:10.1016/j.simpat.2003.11.006]

Wang, L.J., Wang, Q.X., Dong, D.Y., He, X.L., 2009. Simulation Development of Aircraft Cockpit Instruments Based on GL Studio. ICIECS, p.1-4. [doi:10.1109/ICIECS.2009.5364901]

Xi, L.Q., Yuan, G.L., 2009. Analysis of research on middleware technology. *Comput. Knowl. Technol.*, **5**(4):4-82 (in Chinese).

Zhang, L., Zhuang, D.M., Deng, F., Gu, Z.W., 2009. Research on experimental table for ergonomics evaluation of aircraft cockpit. *Flight Dynam.*, **27**(1):81-84 (in Chinese).