Research on the application of virtual instrument technology in simulation training

Qiang Xiao*, *Long* Wang, *Juan* Fang, and *Guiping* Xiao

Jiuquan Satellite Launch Centre, Jiuquan, Gansu, China

Keywords: Simulation, Pointer, Window messaging, Thread.

Abstract. This paper analyzes the feasibility of the application of GL Studio3.2 and Vega Prime2.2 software in combination, explores the application mode in the simulation system, and gives the call method of GL Studio3.2 control. Proposed by passing a pointer address way to deliver complex data types, with the method of window messaging to avoid the waste of time slices, with the method of thread messaging without window messaging method of thread.

1 Introduction

For the simulation training system, whether it is a desktop-based virtual simulation training system, or a semi-physical simulation and physical simulation training system, it is inseparable from the simulation of the instrument and the control panel [5]. Instrument simulation is a key problem in the simulation of various weapons and equipment. The development of virtual instruments mainly involves two aspects: model construction and system implementation. The model usually consists of a geometric model and a behavioral model. The geometric model completes the expression of the virtual instrument geometric information, and the behavior model is used to describe the interaction of the virtual instrument. On the basis of model construction, the system is implemented by computer language.

Virtual instrumentation is essentially an interactive visual simulation that can be developed from the ground up using the 3D graphics driver interface OpenGL or DirectX, or it can be developed using common visual simulation tools, such as the traditional modeling tool 3DSMax Solidwork. Modeling and integration with the visual driver, but these methods are inefficient and difficult. The most effective method is to use professional instrumentation simulation development tools such as GL Studio, VAPS, GMS, CST, Iocomp, etc. for development. Using these tools can reduce the difficulty of modeling, reduce the workload, increase work efficiency, and the effect is very realistic.

GL Studio is a virtual reality rapid prototyping tool [3] that specializes in creating virtual instruments. It is independent of the operating system platform and is used to create real-time 2D or 3D photo-level interactive graphical interfaces. It can be connected to HLA/DIS simulation applications; the C++ and OpenGL source code generated by it can be run separately or embedded in other applications; it runs on Windows, IRIX and Linux

* Corresponding author: 627574615@qq.com

operating systems, and can also run in real-time operations such as VxWorks. The system has a wide range of applications in desktop-based virtual simulation training systems and instrumentation for various physical and semi-physical simulations [1][2]. GL Studio is one of the most advanced human-machine interface development tools in the world, and it has a very large share in the field of training and simulation. Because of its good architecture coupling and excellent reuse mode, it has been adopted by many softwares. As their instruments, panels and interfaces are used in the production module, MPI's Vega Prime uses it to make its own instrument module into the 3D virtual scene. A perfect simulation of the virtual instrument is achieved.

2 System implementation method

2.1 Making virtual models

Virtual models can use GL Studio to build geometric models, or they can be built in other specialized 3D model development tools such as Creator, which can be inserted directly into GL Studio. Generate the gls file by selecting the appropriate project type application or component in the build environment.

Design graphical interface [4]. Take a picture of the physical instrument to obtain the texture, and process the texture into the form required by GL Studio development. Use the graphics processing tool to make the texture of the virtual model, including the background texture of the entire device and the texture of each functional original, and visualize it for GL Studio. Textures are added to each model in the environment to name each functional component. If the texture has been added in the Creator environment, the mapping process can be omitted in GL Studio.

According to the needs of real-time simulation, use the graphical user interface to add attributes, methods and variables to the instrument model, add appropriate callback functions for the interface time that needs to respond, and add code to achieve dynamic control of the model. The GL Studio code generator generates C++ OpenGL source code for various objects created by the GL Studio designer, and then compiles the source code into the corresponding executable file or library function according to the project type.

Apply the created model to the software for debugging. After the above steps, the 3D solid model and the panel are completed and submitted in the form of a dynamic link library.

2.2 Build the virtual environment

The virtual model and the models such as panels and meters are combined and integrated into the visual simulation driver to realize the display and interaction of the three-dimensional virtual model. The created model dynamic link library file is imported into the scene through the GL Studio plugin of the Vega Prime environment, and is configured along with the model created by Creator to complete the construction of the virtual environment.

Vega Prime-based visual simulation mainly includes three parts: preprocessing of visual model, LynX Prime graphical interface and visual simulation programming. The preprocessing of the visual model is to statically model the terrain features in the scene in the early stage, and also to model the complex entities; the lynX Prime graphical interface design mainly includes the content settings and large scene application settings; visual simulation program Design is the key stage of visual simulation. After completing the whole simulation process, the acf file configured in LynX is called to receive the simulation

data sent by the background database in real time. The API function provided by Vega Prime is used to simulate the motion of the model according to the scene. The special effects are processed and controlled to provide users with real-time and smooth visual simulation results.

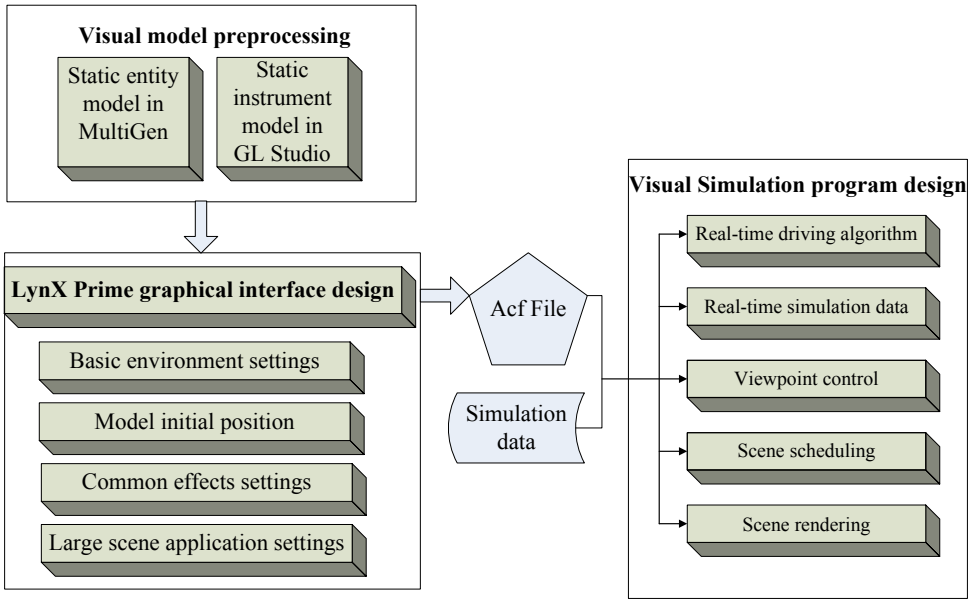

Fig. 1. Visual simulation framework based on vega prime.

Drivers for Vega Prime include initialization, definition, configuration, simulation loops, and exit shutdown systems. Initialization is used to initialize the Vega Prima system and create shared memory and semaphores; the definition is to create a 3D model through the acf application definition file or to create a 3D model through explicit function calls; the configuration is the parameters set by Vega Prime through the definition phase Calling the configuration function to complete the configuration, mainly includes parsing the acf file and establishing a connection with the class; the simulation loop uses the function control functions to complete the rendering and driving of the scene according to the transmitted simulation data, and in each loop body, the application generates a new one. One frame; finally, the vp:shutdown function is called to clean up and free the memory occupied by the kernel, ending the thread.

2.3 Feasibility of combining the two technologies

Quickly complete the production of the instrument through the GL Studio visualization platform, packaged into a class library. The vpGLStudio module is called in the Vega Prime, and the instrument object in the 3D scene is created according to the created instrument class library, and the splicing setting is completed for the instrument object, and can also be dynamically set through the package interface after the Vision loading.

GL Studio provides two ways to call plugins, one is based on source-level calls, and the other is a call to a dynamic link library. The source-level based call is to insert the source file with the suffix gls into the main project. The dynamic link library call inserts a file with the suffix dll into the main project. The plugin called by the source code selects the Standalone AppWizard type project when creating the project in VC++; the plugin called

by the dynamic link library selects the LiveComponent Wizard type project when creating the project in VC++.

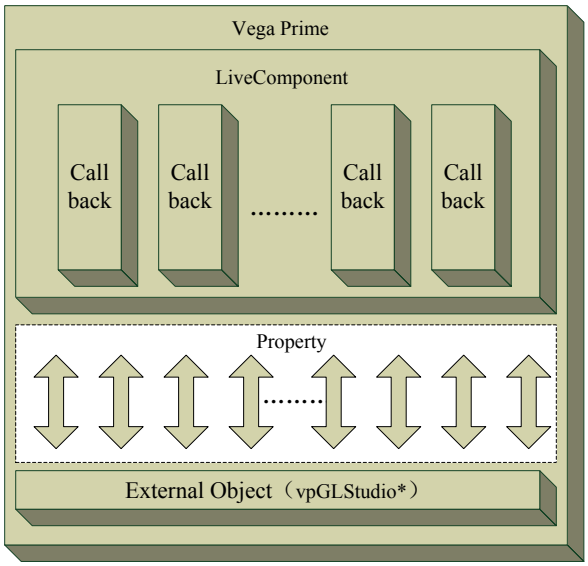

Fig. 2. Schematic diagram of the combination of the two technologies.

The two types of plugins are designed the same in GL Studio, including the graphical interface and writing code. The difference is that the Standalone type plugin can be compiled, linked and run in VC++, and generate a separate exe executable file, run exe the file can be seen and manipulated to design the virtual instrument interface.

After the LiveComponent type plugin completes the graphical interface design and generates code in GLStudio, only the link can be compiled in VC++, but it cannot be run. The LiveComponent type plugin is compiled by generating a file with a suffix of dll and can only be run with the inserted file when it is inserted into another executable file.

The instrument can be divided into two categories according to its functional characteristics: one is the input completion function that receives external data, such as a scale band, a pointer, a text box, and the like. The other type needs to send data such as buttons, switches, buttons, and so on. Therefore, the GL Studio simulation software provides a property interface for external data input to the instrument object, and a callback function is used for logical interaction inside the instrument. For the packaging of the visual simulation platform, it is necessary to provide the corresponding interface to complete the data input to the instrument properties, which is done by calling the vpGLStudioComponent class of the vpGLStudio module. Figure 2 is a schematic diagram of the combination of the two technologies.

The meter display interface glsSetAttribute is packaged as follows:

```
vpGLStudioComponent* m_gls；
/ / Find the instrument name of the input data
m_gls=vpGLStudioComponent::find(glsName)；
/ / Complete the assignment of the instrument properties
m_gls->setAttrib(attribName，attriValue)
```

The function setAttrib provides two parameters, attribName represents the data input property interface provided during the instrument making process, and attriValue represents the data value of the external input. The interactive display of the instrument can be completed by calling the glsSetAttribute interface during the visual simulation process.

3 Type plugin application

3.1 Standalone type plugin application

Fig. 3. Standalone type plugin is inserted into the main panel.

In the left part of Figure 3, the Altimeter is a Standalone type plug-in. To insert the plug-in in the main panel on the right side, you need to select the design file with the suffix gls, and introduce the header file and source file generated by Altimeter into the design project of the main panel.. Click on the plugin in the object list of the main panel design file to enter the design window of the plugin. When writing code in the main panel design file requires connection control for the plugin, the plugin can be controlled by the plugin name->property name. Class properties can be created for each design project. Each class property will generate three class members: member variables, input methods, and output methods. The name and content of each class member can be obtained by automatically generating user customizations. Once the class is created, the reference project can access the class properties through external information. The Type content in the class attribute represents the type of the class attribute, which is the type of the member variable. Name indicates the name of the attribute. Initial Value sets the initial value of the member. The Set Method area defines the input method. For input methods, the return type is null (Void), and the method name is automatically generated from the class attribute name or custom name. The parameter list form is (const type & value), where type is the type of the attribute. The method declaration and content area behind the parameter list are first set to {member variable name=number}, where the member variable name is automatically generated from the class attribute and the value is already set in the parameter list. If all the code is written to the text area below the input method, the initial method declaration and content area can be cumbersome, so just enter the custom code. The definition method and input method of the Get Method output method are the same.

3.2 LiveComponent type plugin application

The multi-function display plug-in is a LiveComponent type plug-in. When inserting the plug-in in the main panel, you must select the file suffixed by the LiveComponent plug-in to generate the dll file, and the plug-in after the main panel editing window is the dynamic running effect. The LiveComponent type plugin implements control of the plugin through a SetResource() function, and accesses the plugin property through the GetResource() function.

mfdDevice->SetResource("altitude","1000.24");

The above statement sets the value of the property altitude of the LiveComponent plugin mfdDevice to 1000.24.

mfdDevice->GetResource("altitude");

The above statement obtains the current value of the property of the LiveComponent plugin mfdDevice, and the return value of the GetResource function is the result of the altitude property, which is returned in string format.

Fig. 4. The LiveComponent type plugin is inserted into the Vega Prime environment.

Standalone type plugins can only be applied in the same kind of files, that is, support inserts into GL Studio's design files; LiveComponent type plugins can be applied to different environments because they are inserted into dll files, not just GL Studio. A dll file is equivalent to encapsulating a program into a component, retaining the interface, and other programs can call the dll file by accessing the interface. Figure 4 shows a panel in which the LiveComponent type plugin is inserted into the Vega Prime environment and can be operated in 3D space.

4. Research on application of key technologies

4.1 Application of pointer in interface data transmission

The external environment Vega Prime implements access to the plug-in properties of the LiveComponent type through the functions SetResource() and GetResource(). However, during the access process, only basic types of data can be accessed, such as int, bool, double, WPARAM, etc., and cannot be accessed for complex type data such as structures. In order to achieve access to complex types of data, it is envisaged to pass the function to pass the pointer address of the data, because the data of the pointer address is 4 bytes, which is consistent with the number of int type data bytes. The data content in the memory unit is accessed by the pointer address, and a program is used to verify the rationality and feasibility of the assumption.

In order to realize the data update of the above three meters as shown in Fig. 3, the attribute WPARAM Datas is first defined for the instrument panel.

The structure of the above three meter data is defined in Vega Prime:

Struct flydatas{type1 data1;type2 data2;type3 data3;};

At some point, the flight data flydatas datas_t needs to be imported into the panel plugin. In Vega Prime, the address of the flight data is passed into the plugin using the mfdDevice->SetResource("Datas", &flydatas) function. The second parameter of the function &flydatas is the flight parameter. address.

In the interface function void Datas (const WPARAM & value), the value passed into the plugin is the pointer address of the flydatas parameter. In order to get the data value from this address, you need to define the data type Struct flydatas{type1 data1;type2 data2;type3 data3;} in the plugin header file. Define the data flydatas datas_t in the body of

the function void Datas (const WPARAM& value) function, then datas_t =*((flydatas*)value).

The interface function WPARAM Datas (){return _Datas;} returns the value of the attribute variable _Datas corresponding to the current control interface. In Vega Prime, the datasadd =mfdDevice->GetResource("Datas") function returns the pointer address of _Datas, and Vega Prime obtains the complex data from the panel control through datas_t =*((flydatas*)datasadd).

4.2 Window-based messaging

The data interaction between the LiveComponent type plugin and the external environment such as Vega Prime is realized by the SetResource() function and the GetResource() function, but there is a real-time problem with the method, such as clicking one of the switches in Figure 4, then the LiveComponent An attribute is changed. If the external environment wants to know the change time of the attribute, it must constantly check the value of the attribute through the GetResource() function. In order to achieve this value without interruption, a separate thread must be opened for Vega Prime to continuously detect the value of this attribute. This method greatly wastes the resources of the system.

In order to solve the above problem, communication between LiveComponent and Vega Prime can be realized by sending and receiving messages. When the LiveComponent process sends a message, the system copies the message from the user buffer to the message buffer in the kernel and then hangs the message buffer into the message queue. Messages sent by the LiveComponent process remain in the message queue until they are received by another process, Vega Prime. When the process receives the message, the system unpacks the message buffer from the message queue and copies the message from the kernel's message buffer to the user buffer.

Sends a message to the Windows main program window when any component on the LiveComponent panel is active. First define two variables: the message variable msg and the window variable vpw, complete the search for the main window in the callback function of the switch, and send the data to be sent to the main window through the PostMessage function. The PostMessage function prototype is:

BOOL PostMessage (HWND hwnd, UNIT Msg, WPARAM wParam, LPARAM lParam);

Where hwnd is the window handle that needs to receive the message, and the handle can be passed to the control inside the control through the setAttrib call property interface when the control is loaded. Msg is a message that needs to be sent. The message can be predefined in the header file. wParam and lParam are the parameters that the message needs to be passed. The return type of the function is Boolean, and the message delivery returns successfully to true, otherwise it is false. The main program receives the message sent by the LiveComponent and logically processes the data. The message is received and processed in the window procedure function DefWindowProc function.

The above method not only avoids the main program from constantly detecting the attribute state to the possession of the time slice, and can transmit more types of data through the message, because the two parameters wParam and lParam of the message can be received by the pointer address, specifically 3.3. The window-based messaging process is shown in Figure 5.

4.3 Thread-based messaging

In some specific cases, an application opens up multiple threads to implement different functions. For example, a program may include a simulation push thread, a rendering thread,

an interface thread, a receiving message thread, and some threads include a window, and then through PostMessage. A message can be sent to the thread by sending a message to the window, but some threads do not contain a window, but also require a message in the LiveComponent control. In order to solve this problem, you can use the PostThreadMessage function to implement message passing to the thread.

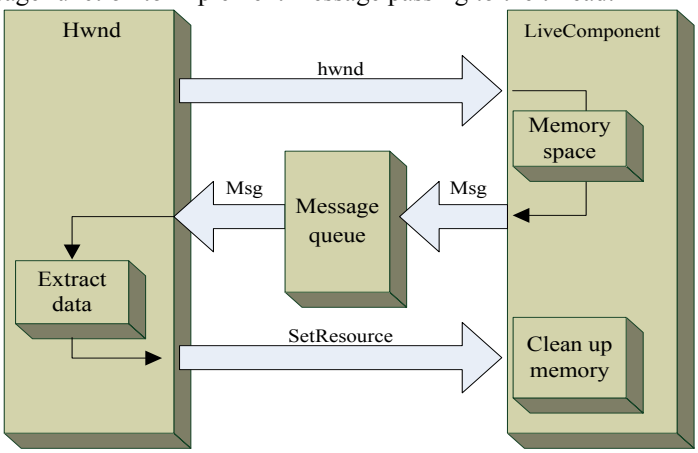

Fig. 5. Window-based messaging process.

The function prototype of PostThreadMessage is:

BOOL PostThreadMessage(DWORD idThread, UNIT Msg, WPARAM wParam, LPARAM lParam);

Where idThread is the thread number that needs to receive the message. This number can be passed to the inside of the control through the property interface of the function setAttrib when the control is loaded. Msg is a message that needs to be sent. The message can be predefined in the header file. wParam and lParam are the parameters that the message needs to be passed. The return type of the function is Boolean, and the message delivery returns successfully to true, otherwise it is false. PostThreadMessage is only responsible for delivering messages. The parameters passed after being passed are destroyed. In order to prevent the thread from receiving data after receiving the message, the transmitted data needs to open up space in advance, and then the memory space is cleaned through the property interface. The thread-based messaging process is shown in Figure 6.

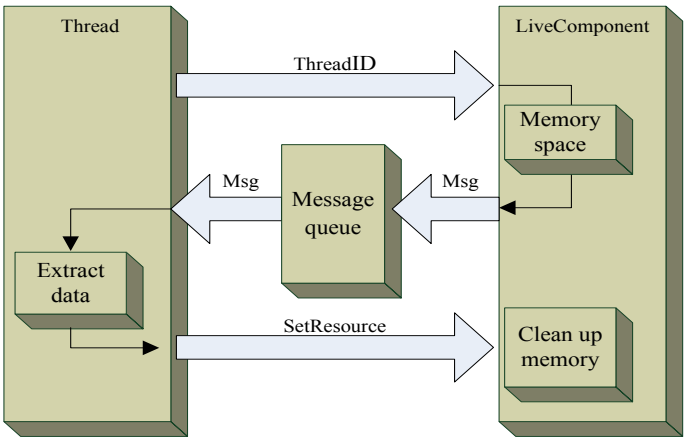

Fig. 6. Thread-based messaging process.

5 Conclusion

With Vega Prime as the main control integrated development environment and GL Studio dll as the instrument-driven simulation mode, it is a new attempt for complex large-system simulation and a new application mode of virtual instrument technology. This development and application mode, Can solve some problems in virtual operation training and testing. In the interface simulation of actual equipment, a large number of mathematical model calculations, data processing and analysis, the advantages of virtual instrument technology can be utilized. Therefore, this development process, as a new development model, has certain promotion and application value. However, this development model as a new idea, there are still many problems that have not been solved, and further application research is necessary.

References

1. Distributed Simulation Technology TNC.GL Studio User's Guide[Z].2007.
2. GL Studio Version 3.2 API Documentation Inc. U.S.A: Distributed Simulation Technology Inc[Z].2007.
3. YuHui et al. Application of GL Studio virtual instrument technology in system development. National Defense Industry Press.2010
4. YuanMei,BaiGang.Design of virtual multi-function display system. Journal of System Simulation,2006
5. ChenXin et al. Development of virtual instrument technology. Dual-use Technology And Products,2009