Available online at www.sciencedirect.com
SCIENCE DIRECT®
Computers & Graphics 29 (2005) 331–339
COMPUTERS & GRAPHICS
www.elsevier.com/locate/cag

Technical section

Interpretive OpenGL for computer graphics

Bo Chen, Harry H. Cheng*

Integration Engineering Laboratory, Department of Mechanical and Aeronautical Engineering, University of California, Davis, CA 95616, USA

Abstract

OpenGL is the industry-leading, cross-platform graphics application programming interface (API), and the only major API with support for virtually all operating systems. Many languages, such as Fortran, Java, Tcl/Tk, and Python, have OpenGL bindings to take advantage of OpenGL visualization power. In this article, we present Ch OpenGL Toolkit, a truly platform-independent Ch binding to OpenGL for computer graphics. Ch is an embeddable C/C++ interpreter for cross-platform scripting, shell programming, numerical computing, and embedded scripting. Ch extends C with salient numerical and plotting features. Like some mathematical software packages, such as MATLAB, Ch has built-in support for two and three-dimensional graphical plotting, computational arrays for vector and matrix computation, and linear system analysis with advanced numerical analysis functions based on LAPACK. Ch OpenGL Toolkit allows OpenGL application developers to write applications in a cross-platform environment, and all of the OpenGL application source code can readily run on different platforms without compilation and linking processes. In addition, the syntax of Ch OpenGL Toolkit is identical to C interface to OpenGL. Ch OpenGL Toolkit saves OpenGL programmers' energies for solving problems without struggling with mastering new language syntax. Ch OpenGL Toolkit is embeddable. Embedded Ch OpenGL graphics engine enables graphical application developers or users to dynamically generate and manipulate graphics at run-time. The truly platform independent, scriptable, and embeddable features of Ch OpenGL Toolkit make it a good candidate for rapid prototyping, mobile graphics applications, Web-based applications, and classroom interactive presentation. The design issues of Ch OpenGL Toolkit and its potential applications are presented in the article. A methodology that can be used to implement a Web-based visualization system based on Ch OpenGL and Ch CGI is also introduced. The method described in the article can be easily followed to create a Web-based visualization system at low cost and with minimal effort. The software packages Ch and Ch CGI Toolkit are freely available and can be downloaded from the Internet.

Keywords: Methodology and techniques—interaction techniques; Graphics utilities—software support; Graphics systems—distributed/network graphics

1. Introduction

The field of computer graphics continues rapidly growing with an ever-increasing number of applications in diverse areas, such as entertainment, business, art, education, medicine, engineering, and industry. A number of software packages have emerged to help generate and manipulate two-dimensional (2D)/three-dimensional (3D) graphics. OpenGL [1] is a graphical application programming interface (API) for the C/C++ programming language. The primary motivation for developing OpenGL API is to create an operating

*Corresponding author. Tel.: +1 530 752 5020; fax: +1 530 752 4158.
E-mail address: hhcheng@ucdavis.edu (H.H. Cheng).

0097-8493/$ - see front matter © 2005 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cag.2005.03.002

system, window system, and hardware platform independent API for the development of 2D/3D graphics. Since OpenGL API was introduced in 1992, many applications, such as CAD, CAM, and game development, have benefited from its cross-platform accessibility. OpenGL has become a premier environment for developing portable 2D/3D graphics applications. It is also widely used for teaching and learning computer graphics. The features of device independence and portability make OpenGL a strategic interface for courses on computer graphics. Computer platforms vary from instructor to student and from school to student home. By using OpenGL, programs developed on a machine can be debugged and graded on other machines with different platforms, and the resulting graphics are the same.

Since OpenGL is one of the most popular industry standard graphical software packages, many languages, such as Fortran, Java, Tcl/Tk, and Python, have OpenGL bindings to take advantage of OpenGL visualization power. The information about these language bindings to OpenGL is introduced in Section 2. We have developed Ch OpenGL Toolkit [2]. Ch is an embeddable C/C++ interpreter. Ch OpenGL Toolkit further enhances the portability of OpenGL API. Usually, OpenGL application programs have to be compiled and linked before running these programs on different platforms. Ch OpenGL Toolkit makes OpenGL applications truly portable across different platforms. With Ch OpenGL, OpenGL application source code can readily run on different platforms without compilation and linking processes.

The design issues of Ch OpenGL Toolkit and its potential applications are presented in the article. A comparison of Ch OpenGL Toolkit to similar attempts is also given and the novel features of Ch OpenGL are highlighted. The remainder of the article is organized as follows. Section 2 reviews major language bindings to OpenGL. Section 3 introduces Ch and Ch OpenGL. Several design issues related to Ch OpenGL Toolkit are discussed. Section 4 presents some potential applications of Ch OpenGL. Section 5 summarizes different approaches of Web-based visualization systems and demonstrates how to implement a Web-based visualization system based on Ch OpenGL and Ch CGI. Section 6 discusses planned future work for portable and Web-based animation. Section 7 summarizes the presented work.

2. Related work

OpenGL is supported by major operating systems and window systems, and it is callable from many programming languages. This section introduces different language bindings to OpenGL.

The Fortran 90 Interface to OpenGL, f90gl [3], is a public domain implementation of the official Fortran 90 bindings to OpenGL. The interface is responsible for ensuring the interoperability between Fortran and C, such as matching Fortran data types to C data types, choosing subroutines and functions in Fortran for different C functions, and dealing with array arguments. Most vendor implementations of the Fortran interface to OpenGL are for a specific system with specific Fortran and C compliers. Although the Fortran 90 Interface is much more robust and portable than the Fortran 77 interface because the Fortran/C interface is contained entirely inside the Fortran 90 interface to OpenGL and hidden from the user, some potential problems still exist. For example, the user is responsible for choosing appropriate compilers that provide a sufficient inter-language calling convention, ensuring the persistence of function arguments that are assigned to C pointers internally, dealing with *unsigned int* data type in C functions, and paying special attention to the array order difference in Fortran and C [4].

The JOGL Project is a reference implementation of the Java bindings for OpenGL API, and is designed to provide hardware-supported 3D graphics to applications written in Java [5]. Jogl provides access to the latest OpenGL routines (OpenGL 1.4 with vendor extensions) as well as platform-independent access to hardware-accelerated off-screen rendering (“pbuffers”). JOGL was designed for the most recent version of the Java platform, J2SE 1.4 and later. It only supports true color (15 bits per pixel and higher) rendering, and it does not support color-indexed modes. Since JODE is an ongoing project, the coverage of accessing OpenGL functionality is limited, and there are some issues remain on different platforms, which can be found in JOGL user’s guide on the Web page [5].

Scripting languages have been increasingly used for rapid prototyping, dynamic manipulating components, and Web-based applications. The reason for the increasing popularity of scripting languages in these application areas is that scripting languages provide rapid turnaround during development by eliminating compile times and allow users to dynamically program the applications at run-time [6]. Most scripting languages embed different binary libraries inside the language allowing users access to the library’s functionality in scripts. For example, many scripting languages have bindings to OpenGL providing interpretive access to the OpenGL libraries so that developers can perform interactive 3D graphics in scripts. As examples, Tcl/Tk and Python bindings to OpenGL are described below.

There are several Tcl/Tk bindings to OpenGL. The most popular two implementations are TKOGL and Togl. TKOGL [7], a Tk OpenGL widget, enables the creation and display of 3D graphics using the OpenGL API. The advantage of this implementation is that the

3D graphics widget behaves like a Tk 2D widget, enabling both the experienced and novice users to generate and display 3D models in a concise manner. The drawback of the system is that it ties OpenGL commands to the widget and destroys one of the main advantages of OpenGL, window system independence. Togl [8] is also a Tk widget for OpenGL rendering. Although Togl provides the means to open a window for displaying OpenGL graphics, it does not include Tcl bindings for any of the OpenGL rendering functions. A typical Togl program has Tcl code for managing the user interface and a C program for computations and OpenGL rendering. To use Togl effectively, one should be familiar with Tcl, Tk, OpenGL, and C programming.

PyOpenGL [9] is the Python binding to OpenGL and related APIs. It is an influential scriptable package for 3D graphics and has received some practical applications. However, like other script language bindings to OpenGL, the syntax of functions in PyOpenGL has been modified from standard OpenGL API. Furthermore, the calling method and functionality of some functions appeared in PyOpenGL 2 are different from that of the C counterparts because of the differences between C and Python. For example, each PyOpenGL function call, which has an argument of an array pointer or array element, has different format from a C function call due to the difference between C and Python in the way that they access arrays.

3. Ch and Ch OpenGL

Ch, originally developed by Cheng [10,11], is an embeddable C/C++ interpreter for cross-platform scripting, shell programming, numerical computing, and embedded scripting. It supports all features of the C language standard ratified in 1990. Many new features such as complex numbers, variable length arrays, IEEE floating-point arithmetic and type-generic mathematical functions first implemented in Ch were adopted in C99, a new C standard ratified in 1999. In addition, Ch supports classes in C++ for object-based programming. Like other mathematical software packages, such as MATLAB, Ch has built-in support for two and three-dimensional graphical plotting, computational arrays for vector and matrix computation, and linear system analysis with advanced numerical analysis functions based on LAPACK. With the power of computational arrays, we can specify vector and matrix operations directly in expressions in the same way as scalars in both interactive execution and programs. Fig. 1 shows interactive execution of programming statements in a Ch shell. The type declarators *array* and *int* declare variables A and B as computational arrays of *int* type. The header file array.h needs to be included in a program to use computational arrays.

Ch OpenGL is a Ch binding to OpenGL. It provides access to the full functionality of OpenGL, GLU, GLUT, and GLAUX. Ch OpenGL has many novel features. First, Ch is a cross platform C/C++ interpreter. As a part of Ch distribution, Ch OpenGL allows OpenGL application developers to write applications in a cross-platform environment. All of the OpenGL application source code can readily run on different platforms without compilation and linking processes as shown in Fig. 2. Second, the syntax of Ch OpenGL is exactly the same as C interface to OpenGL. There is no need of learning new syntax. Third, Ch OpenGL is embeddable. Embedding Ch into graphics applications allows developers or users to dynamically generate and manipulate graphics at run-time. Finally, computational array in Ch makes vector and matrix operations carried out in 3D graphics more concise. We believe that the computational efficiency of graphics applications can be

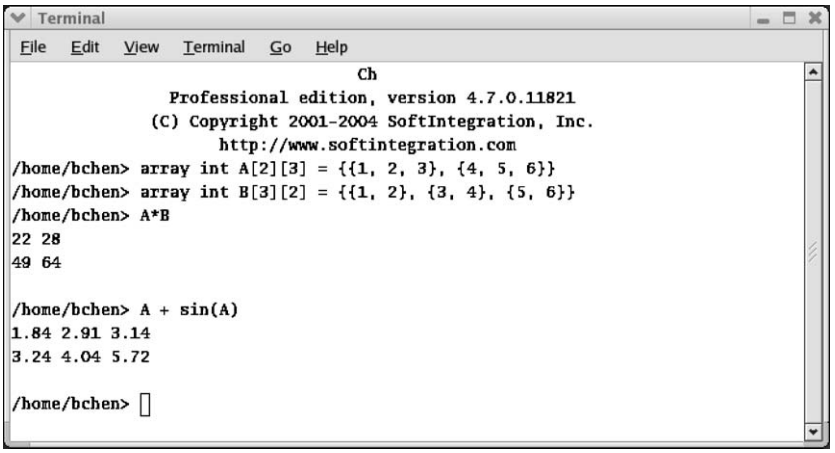

Fig. 1. Vector and matrix operations in Ch.

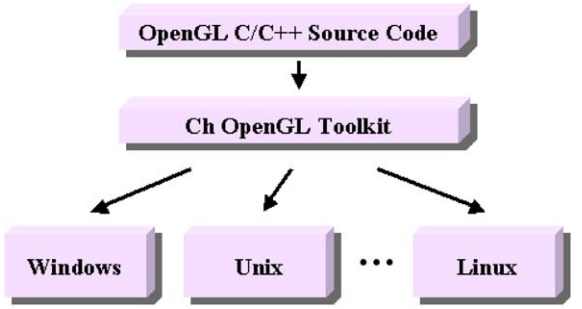

Fig. 2. Run C/C++ OpenGL source code on different platforms.

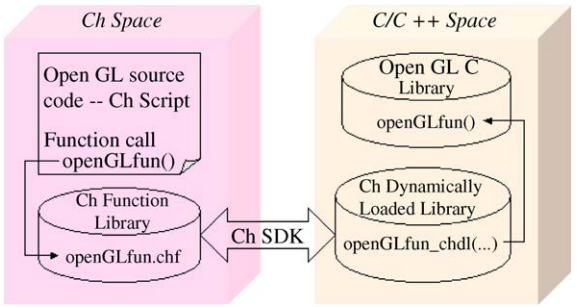

Fig. 3. Ch SDK allows Ch scripts interfacing with C/C++ binary libraries.

significantly improved by taking advantage of Ch numerical capabilities.

Ch interface to OpenGL is much simpler than any other language interface to OpenGL because both Ch and OpenGL share same syntax and data types. Ch scripts can access functions in static or dynamical C/C++ binary libraries through Ch SDK. Fig. 3 shows how a Ch OpenGL script interfaces with the OpenGL C library. When a Ch OpenGL script (C/C++ OpenGL source code) runs, Ch looks for function files that correspond to functions in the script in a Ch function library. These function files pass function parameters in the Ch space to the corresponding functions in the Ch dynamically loaded library (CDLL) in the C space through Ch SDK. Functions in the CDLL invoke OpenGL C functions and return results back to Ch function files. Ch OpenGL Toolkit consists of a Ch function library and a Ch dynamically loaded library for OpenGL. The implementation of Ch OpenGL Toolkit is straightforward for simple OpenGL C functions. For some functions, which have arguments of pointers to callback functions, need to be specially handled because these functions in the C space have callbacks in the Ch space. The strategy to solve this problem in Ch OpenGL Toolkit is to create a Ch callback counterpart in the C space. As an example, the program for the system shown

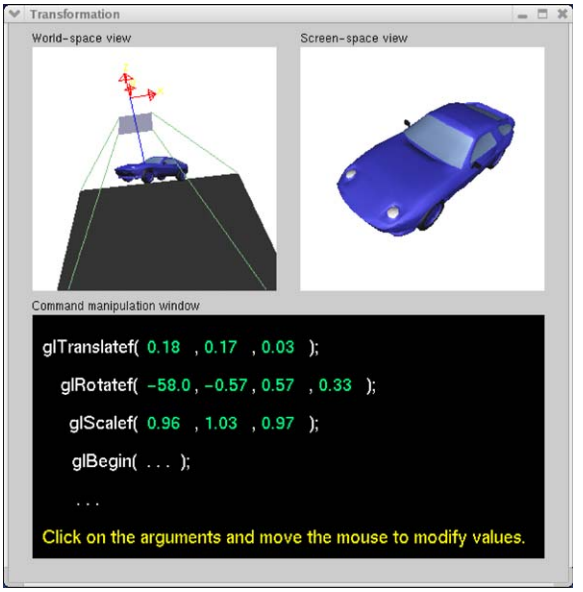

Fig. 4. Running an OpenGL program interpretively and interactively.

in Fig. 4 [12] uses function *glutDisplayFunc* to set the display callback for each window in GLUT. When GLUT determines that the screen-space window needs to be redisplayed, the display callback for the screen-space window in the C space is called, and this C callback in turn calls the Ch callback through a dynamically loaded library as shown in Fig. 5.

4. Potential applications of Ch OpenGL

4.1. Run OpenGL programs interpretively

Ch OpenGL Toolkit supports core OpenGL, GLUT, and GLAUX. A C/C++ program using OpenGL functions can be readily treated as a Ch script. The program can directly run in different command shells, an integrated development environment (IDE), or Windows explorer in Windows without compilation and linking. Fig. 6 illustrates how a C program *transformation.c*, one of the popular Nate Robins' OpenGL tutorial demo programs [12], is executed interpretively in a Ch shell. Fig. 4 displays the output from the execution of the program *transformation.c* in a Ch command shell. The source code of all Nate Robins' OpenGL demos is available at [12,13]. They are readily to run in Ch without compilation [13]. Interpretive OpenGL is well suited for graphical rapid prototyping, interactive classroom presentation, and student learning. The effects of different parameters on the output can be appreciated instantly by modifying their values and

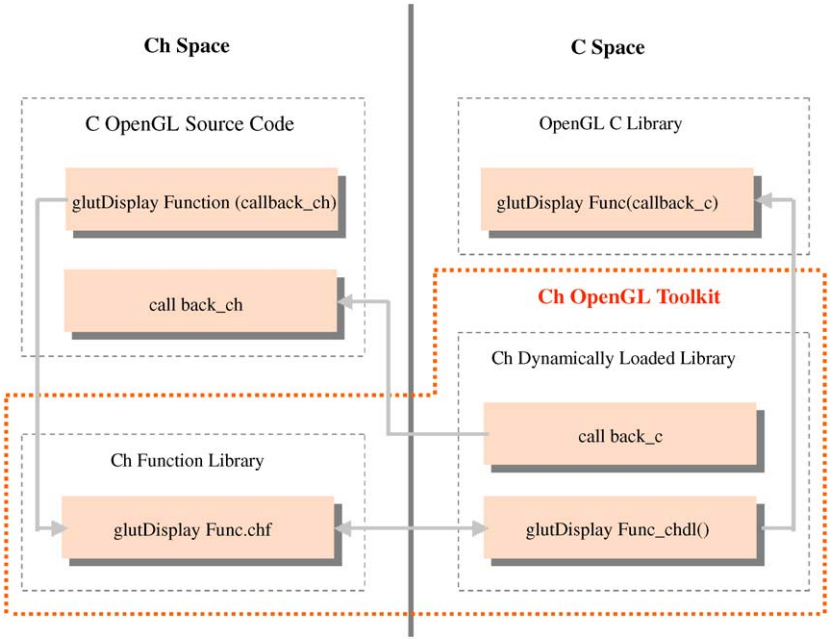

Fig. 5. Handle functions with callback arguments.

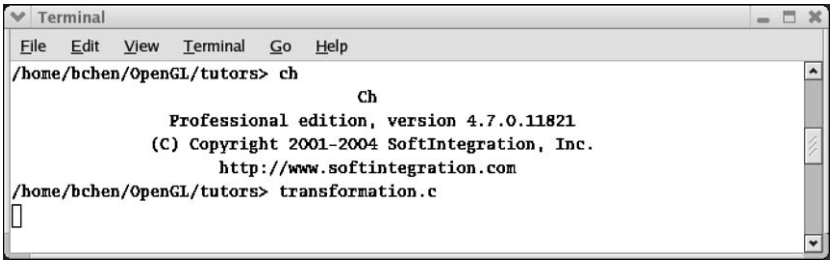

Fig. 6. Interpretive execution of the C program *transformation.c* in a Ch shell.

running source code immediately without recompilation and linking.

4.2. Mobile graphics

In internet-based distributed cooperative graphics systems, there is a need to send graphics to different hosts in the system. Instead of sending graphics data, we may send code to the target systems to generate and manipulate graphics locally. This approach reduces the network traffic dramatically in heavy graphics transmission systems. Graphical mobile programs can be dynamically generated by application programs online in one machine and run on other machines with different platforms. In order to support the execution of the mobile graphical code in a heterogeneous network, a scriptable graphical engine, such as Ch [11], should be embedded into each host to which mobile graphical code may migrate. Ch OpenGL is a good candidate for this kind of applications.

4.3. Web and network applications

Many existing visualization applications were written in C/C++ using OpenGL. However, difficulty in interfacing with Web servers impeded their applications in Web-based visualization systems. As a superset of C, Ch is suitable for Web-based applications because of its interpretive nature. Many Web-based applications, such as a Web-based system for control system design and analysis [14], have been developed in Ch. With Ch OpenGL and Ch CGI [15], C/C++ OpenGL programs can be used in Web-based visualization systems.

5. Ch OpenGL Web applications

This section introduces different approaches of Web-based visualization systems and presents how to implement a Web-based visualization system based on Ch, Ch OpenGL, and Ch CGI.

5.1. Different approaches of Web-based visualization systems

Web-based visualization systems are usually based on client/server architectures as shown in Fig. 7. For server-based approaches, visualization is executed on the server side, and resulting graphics files are returned to a client machine for viewing in a browser. An end-user sets application parameters and activates a visualization process through a form-based interface in a browser. The form is processed by CGI, ASP, or JSP, or other scripts, which invoke an application program on the server machine and send the resulting graphics data or file to the client machine.

Fig. 7 (a) shows a server-based visualization system, which uses virtual reality modeling language (VRML) [16]. VRML is an ISO standard file format for describing interactive 3D scenes and worlds. In this approach, VRML scenes are generated on the server side by a VRML engine and rendered on the client side by a VRML browser plug-in. This approach requires the client machine to have a Web browser with a VRML plug-in. Furthermore, the client machine has to supply certain graphics power to support the rendering of VRML objects.

Instead of rendering VRML scenes on the client side, a different approach, shown in Fig. 7(b), does complete visualization and rendering on the server side and simply transfers resulting image files to the client. This approach has several advantages. First, the client machine has the lowest software and hardware requirements. Second, the system can create high quality images by taking advantage of high quality rendering software provided on the server machine. Third, the close coupling of visualization and rendering improves interaction with underlying data.

Unlike server-based approaches, visualization is completely done on the client side in client-based approaches. For the system shown in Fig. 7(c), the server machine only acts as a database to provide visualization data.

The approach showed in Fig. 7(d) transfers both application programs and visualization data over the Web. The application programs are usually Java applets. Java applets downloaded from the server execute visualization on the client machine. Java 3D, a low level 3D scene-graph based graphics programming API, has to be installed on the client machine so that Java applet can use Java 3D to render 3D graphics.

Ch OpenGL based Web visualization system uses the server-based architecture shown in Fig. 7(b). OpenGL application programs are installed on the visualization server. These programs retrieve users' data from HTML documents and deliver resulting graphics image files to

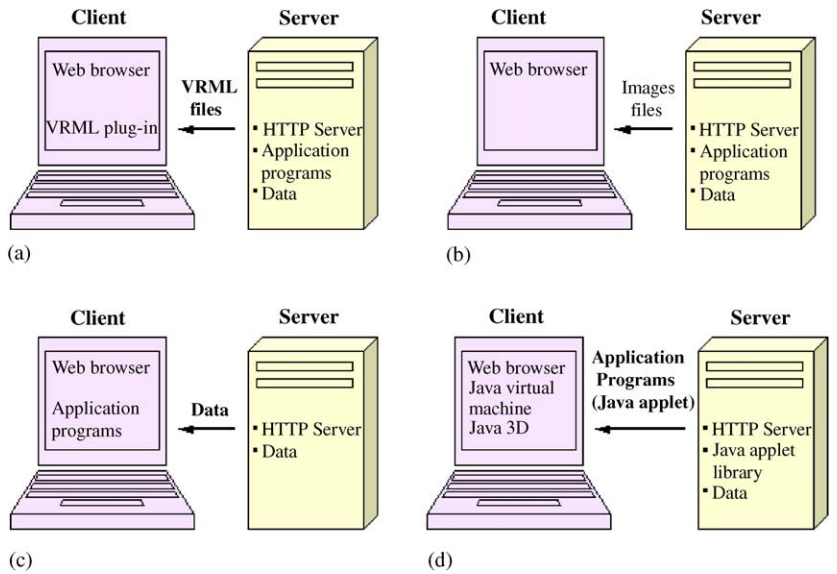

Fig. 7. Commonly used approaches for Web-based visualization systems: (a) Server-based visualization—transfer VRML files, (b) server-based visualization—transfer image files, (c) client-based visualization—transfer data, (d) client-based visualization—transfer software and data.

users through Ch CGI. The Ch OpenGL based Web visualization system offers the following features:

1. C/C++ visualization programs can directly be used in Web-based visualization systems.
2. Ch OpenGL visualization server is application transparent. It runs existing OpenGL application programs without any modifications. Moreover, server programs are platform-independent.
3. From the user's point of view, server visualization power can be accessed with the lowest requirements—only a Web browser is needed.
4. From the system developer's point of view, Ch OpenGL visualization server programs are easy to develop and maintain. Furthermore, Ch OpenGL based Web visualization system avoids costly new hardware and software investments. Existing computer facilities and network infrastructure can be used to deliver server visualization power to remote users. The software packages Ch, Ch OpenGL, and Ch CGI toolkit are freely available and can be downloaded on the Web [11].

5.2. Off-screen rendering interface and image file creation

The graphics generated by OpenGL is normally rendered into a window. For Internet-based applications, however, the generated images are required to render into an image buffer instead of a window. This is called off-screen rendering. Off-screen rendering is generally provided by OpenGL extensions to a native window system, such as GLX in Unix or WGL in Windows. The portability and performance trade-offs of the off-screen rendering facilities for WGL, GLX and Mesa were described in [17].

Mesa [18] is a 3D graphics library with APIs identical to those of OpenGL. Mesa's off-screen rendering interface called OSMesa is an operating system and window system independent facility for off-screen rendering. The OSMesa is quite simple. It provides 3 functions for making off-screen renderings. Function OSMesaCreateContext() creates an RGBA-mode context. Function OSMesaMakeCurrent() binds an image buffer to the context and makes it current. Function OSMesaDestroyContext() destroys the context. The default maximum image size is 1280×1024 and it is specified by macros MAX_WIDTH and MAX_HEIGHT in the header file *src/config.h*. If users want to generate images larger than the default size, they have to edit the *src/config.h* file to change MAX_WIDTH and MAX_HEIGHT to the desired values and recompile Mesa.

Since Mesa does not provide facilities for writing image files from an off-screen image buffer, we created a function called ChCreateImage() to achieve this purpose. The prototype of function ChCreateImage() is as follows:

int ChCreateImage(*unsigned char *curpix, int pixelsize, unsigned int width,*
unsigned int height, int outputtype, …
/ int imagetype, char *imagename */*).

This function takes the pointer to the off-screen image buffer, the number of bytes for each pixel, the image width, and the image height as the first four arguments. The function can create two output image formats, and the output format is decided by the fifth argument. When the value of the fifth argument is CH_IMAGE_STREAM, the function simply outputs the resulting image to the standard output stream. If the value of the fifth argument is CH_IMAGE_FILE, the function saves the image to a file. The image file format and name are specified by the arguments *imagetype* and *imagename*. In the current implementation, image files can be saved to either PPM or PNG format. Two macros, CH_IMAGE_PPM and CH_IMAGE_PNG, represent PPM format and PNG format, respectively.

5.3. Implementation of Ch OpenGL based visualization server

The implementation of Ch OpenGL based visualization server is based on an HTTP Web server as shown in Fig. 8. By using a standard Web server, there is no need for developing new server programs. Users' requests are sent to the server as HTML documents. Ch CGI programs extract parameters that are encoded in HTML documents and invoke corresponding OpenGL

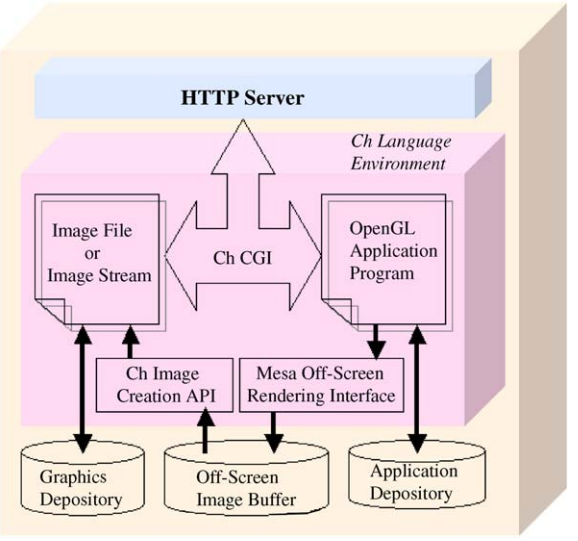

Fig. 8. Modular structure of the visualization server.

programs. Since we want to display resulting graphics images in the user's browser, the off-screen rendering method described in the previous subsection has been used in the visualization server implementation. Instead of rendering 3D graphics generated by an OpenGL program to a window, we use Mesa off-screen rendering interface to save it to an off-screen image buffer. After an image is created in the image buffer, a Ch image creation API is used to convert image data to the PNG format, and the resulting image is displayed on the user's Web page through Ch CGI.

CGI enables a Web server to receive clients' requests, execute application programs on the server, and send execution results back to clients. HTML documents that a Web server retrieves are static. With CGI, Web servers are able to output dynamic information on Web pages according to clients' requests. Although a CGI program can be written in any language that allows it to be executed on the host, many people prefer to write CGI programs using scripting languages since they are easier to debug, modify, and maintain than a typical compiled program. With Ch, C/C++ programs become scripts. In addition, Ch CGI toolkit [15] provides four easy-to-use classes, namely, CResponse, CRequest, CServer, and CCookie, for CGI programming in Ch. These four classes provide member functions similar to API in active server page (ASP) and Java server page (JSP). The member functions of these classes can directly be integrated into OpenGL C application programs. The user's inputs, through a fill-out form on the Web page, are extracted by Ch CGI programs and passed to OpenGL application programs directly. Similarly, output images created by OpenGL application programs can be used to generate dynamic Web pages inside the application programs.

5.4. Web-based interactive 2D/3D graphics

Fig. 9 shows an example of using Ch OpenGL based Web visualization system to interactively configure 3D graphics. The source code for this example is included in Ch OpenGL distribution. End-users can set the dimensions of the output image, the colors of torus, cone, sphere, and the background through a fill-out form. These parameters are encoded by the client browser and decoded by the Ch CGI member function CRequest::-getFormNameValue(). Since Ch CGI toolkit is a set of C++ classes, we can integrate CGI programs with OpenGL application programs so that the system parameters submitted by end-users can be passed to application programs without any additional interface. Once a user activates an application program, the application program creates a new off-screen image according to new parameters and sends the output

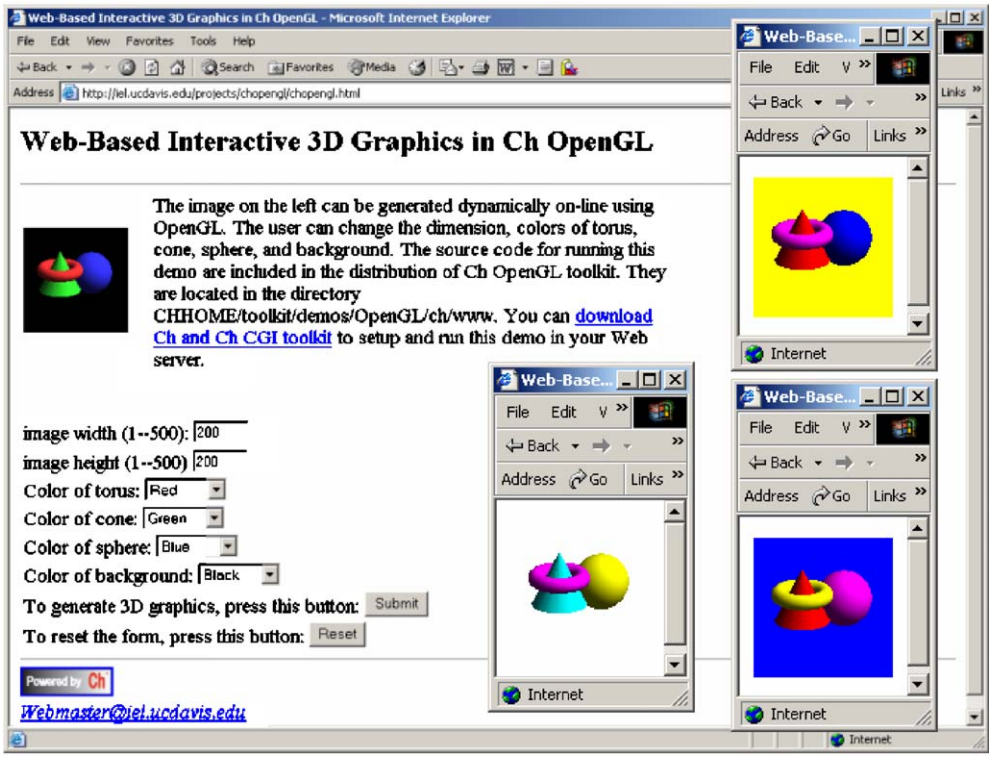

Fig. 9. Web-based interactive configurations of a 3D graphics.

resulting image as a standard output stream to a CGI program to create a new Web page for display.

6. Future work

Ch supports X11/Motif and Win32 graphical APIs. Ch OpenGL graphical animation in a standalone computer works both in Unix and Windows platforms. However, X11/Motif and Win32 graphical APIs are platform dependent. Although GLUT is portable to different platforms, its graphical user interface is very limited. GTK+ is a powerful multi-platform toolkit for creating graphical user interface. The current version of Ch only contains GTK+ version 1.2.8, which does not support GTKGLArea, a widget that allows OpenGL commands within a GTK drawing area. When the Ch binding to the latest GTK+ is available, graphical user interface, such as menus, tool bars, entries, and dialogs, can be integrated into OpenGL applications and allows the user to dynamically manipulate graphics in heterogeneous platforms.

The Ch OpenGL Web application example given in the article is based on Ch CGI. CGI is a very simple interface for writing dynamic Web pages. The interface is not meant to be high performance and complicated. Applications with animation cannot be performed by simple CGI interface. We are investigating appropriate approaches to solve this problem. One possible solution is to develop a browser plug-in with an embedded Ch to perform visualization on the client machine as shown in Fig. 7(d). Security is the major concern for this approach. Safe Ch [11] and encryption might be necessary for both code transmission and execution.

7. Conclusions

OpenGL is the most popular industry standard graphical API for developing portable 2D/3D graphics applications. Ch OpenGL further enhances the portability of OpenGL. Ch OpenGL allows OpenGL application developers to rapidly develop and deploy applications across different platforms. It is the only implementation that enables C/C++ OpenGL programs to run on different platforms without compilation and linking processes. In addition, the syntax of Ch OpenGL Toolkit is identical to C interface to OpenGL. Ch OpenGL saves OpenGL programmers' energies in problem solving without struggling with mastering new language syntax. Ch OpenGL Toolkit is embeddable. Embedded Ch OpenGL graphics engine allows graphical application developers or users to dynamically generate and manipulate graphics at run-time. Because Ch OpenGL is truly platform independent, scriptable, and embeddable, it is a good candidate for rapid prototyping, mobile graphics applications, Web-based applications, and classroom interactive presentation.

A methodology that can be used to implement a Web-based visualization system based on Ch OpenGL and Ch CGI was introduced in the article. The method described in the article can be easily followed to create a Web-based visualization system at low cost and with minimal effort. The software packages Ch and Ch CGI toolkit are freely available and can be downloaded from the Web [11]. With Ch and Ch CGI, C/C++ visualization programs can directly be used in Web-based visualization systems. Ch OpenGL based Web visualization system offers the possibility for new forms of customer service, such as interactive 3D product configuration, ordering, and customer training. It is also an ideal environment for teaching and learning computer graphics.

References

[1] Woo M, Neider J, Davis T, Shreiner D. OpenGL programming guide: the official guide to learning OpenGL, version 1.2, 3 ed. Reading: Addison-Wesley; 1999.
[2] Ch OpenGL toolkit, http://www.softintegration.com/products/toolkit/opengl/, Softintegration, Inc.
[3] f90gl Fortran interface for OpenGL and GLUT, http://math.nist.gov/f90gl/.
[4] A Fortran 90 Interface for OpenGL: revised January 1998. NISTIR 6134, February 1998.
[5] The SUN/SGI Java/OpenGL bindings, https://jogl.dev.java.net/.
[6] Ousterhout JK. Scripting—higher level programming for the 21st century. Computer 1998;31(3):23–30.
[7] Esperanca C. A Tk OpenGL widget. Proceedings of USENIX Fifth Annual Tcl/TK Workshop. 1997.
[8] Paul B, Bederson B. Togl—a Tk OpenGL widget, http://togl.sourceforge.net/.
[9] PyOpenGL 2.0–the PyOpenGL binding, http://pyopengl.sourceforge.net/.
[10] Cheng HH. Scientific computing in the Ch programming language. Scientific Programming 1993;2(3):49–75.
[11] Ch—an embeddable C/C++ interpreter, http://www.softintegration.com/, Softintegration, Inc.
[12] Robins N. OpenGL tutorial, http://www.xmission.com/~nate/tutors.html.
[13] Ch OpenGL toolkit demos, http://iel.ucdavis.edu/projects/chopengl/.
[14] Yu Q, Chen B, Cheng HH. Web-based control system design and analysis. IEEE Control Systems Magazine 2004;24(3):45–57.
[15] Ch CGI toolkit, http://www.softintegration.com/products/toolkit/cgi, Softintegration, Inc.
[16] Ames AL, Nadeau DR, Moreland JL. The VRML Sourcebook. New York: Wiley; 1996.
[17] Paul B. SIGGRAPH'97 Course 24: OpenGL and window system integration—OpenGL/Mesa off-screen rendering, 1997, http://www.mesa3d.org/brianp/sig97/offscrn.htm.
[18] The Mesa 3D graphics library, http://www.mesa3d.org.