GraphJudge: a system for assisted assessment of Computer Graphics assignments

Lindomar Rocha, Rui Rodrigues

Dept. of Informatics Engineering, Faculty of Engineering, University of Porto, Portugal

ABSTRACT

We describe a system for submission, automated validation and assisted evaluation of Computer Graphics' assignments. This type of assessments has a series of specific challenges associated to it, namely the analysis of graphical output and the variations between implementation in different development platforms and operating systems. When considering a large number of submitted programs (in the order of hundreds), preparing and evaluating the submissions quickly becomes a very time-consuming task. The goals of the system are therefore to allow submission and early validation for the students, and to ease the teachers' burden of compiling and executing, while automatically providing visual information of the submissions in an integrated and effective way.

1. Introduction

In this paper we describe the design and implementation of a system for submission, automated validation and assisted evaluation of Computer Graphics' assignments, namely (but not limited to) programs written in C++ using OpenGL [ST09], GLUT [Khr11] and GLUI [Rad10] libraries.

The creation of such system was motivated by the need to address a series of specific challenges associated to this type of assignments, namely the analysis of graphical output, the interactive input and the variations between implementations in different development platforms and operating systems.

When considering a large number of submitted programs (in the order of hundreds), dealing with compilation and execution of these submissions, and performing the evaluation itself quickly becomes a very time-consuming task. One of the main factors that contribute to this is the submission of incomplete or incompatible projects, that when compiled in a system different from the one in which they were developed lead to failed compilation due to e.g. different library paths or missing dependencies.

Although Computer-Based Assessment (CBA) systems have been gaining momentum in recent years, the issue of graphics-based assignment evaluation and the aforementioned challenges have not had specific coverage.

Therefore, we propose a system with the following goals:

- to provide to the students a submission, compilation and early validation system, guaranteeing a working and executable submission.
- to provide an assessment environment for the teachers, that not only eases the burden of compiling and executing (as this is guaranteed by the early validation), but also provides automatically visual information of the execution of the submissions in an effective format.
- to be as modular as possible, and to be able to cope with different development environments and more importantly, to deal with submissions based on different operating systems.

The remainder of this paper details the work developed to pursuit such goals. Section 2 presents an overview of related work, namely in terms of CBA solutions, and identifies their limitations in the presented context. Section 3 describes the desired workflow for the proposed system. The resulting architecture and its main components are described in section 4. Section 5 contains the results obtained with the prototype developed, and in Section 6 conclusions are drawn, followed by ideas for further improvement of the system.

2. Related work

In order to assess and evaluate the system functionalities we researched systems for the submission and evaluation of programming projects. There are two main categories of systems: those developed in the context of Computer-Based Assessments and Computer-Based Learning, and those developed in the context of programming competitions.

A study was carried out in order to find the systems that have been developed in those fields. The systems found are classified in order to keep the ones that give distinct aspects related to the submission management, support for graphical interfaces, the type of design, and features related to security and scalability to support other programming languages, and to the type of feedback given to the users.

Systems like BOSS [JGB05], GAME [BGNM04] and MILE [IPVB08], are aimed to support courses in the academic environment. They provide features such as submission process, feedback system to students and support for the student's learning process, among others. They are de-

signed to provide great performance in terms of access and response and to support system scalability for new components and new programming languages.

The BOSS system was developed to help on the submission and test of students solutions. It has a plagiarism detection module, but all its features are related to the assessment process, and it does not have any functionalities that helps and support the teacher in the students' learning process. It can be scalable to support other languages apart from JAVA and C.

The GAME system is a marking system for programs written in JAVA and C++. It holds student submissions and has an automated mark system. The feedback system gives the student the result of the compilation, and programming style and structure. The system does not have any features related to the students learning process in order to help teachers to follow and support them.

When talking about e-learning platforms the MILE system is the most complete one. It has a component, Mag that supports teachers to share learning materials with their students and to help them following their learning process by solving theoretical exercises. The system integrates an automated system, the Testovic, that tests the students submissions and gives detailed feedback and the results of the assessments. At last there is the Svetovic component that is an IDE that allows students to develop their solutions in a comfortable and familiar environment. The system has support to a large range of programming languages and can be scalable to support other languages and other components.

Other systems like DOMjudge [EKdW09] and PC2 [ABL09], that support programming competitions, were studied in order to know the evaluation criteria that have been created to support automated tests. They have to handle security and performance issues, in order to hold a large number of submissions at the same time.

These two systems have a range of features that include setting up the competition environment, holding teams submissions and give detailed feedback of their results. They also allow teams to ask for clarifications from judges. But looking to their design DOMjudge reveals a better conception, using a distributed design that separates the judgehost from the server that receives submissions. The judgehost is responsible to check team's submissions, and there can be more than one in order to optimize the system's performance when handling large numbers of submissions to be tested.

None of the studied systems has support for graphical output programming, and those that had their source code available have a design in which it would be very complex to incorporate the proposed goals. Due to this, we opted to develop a system that would fulfill those goals, while taking into account a series of concepts present in the various projects analysed here.

3. GraphJudge workflow

With the goals presented in the introduction in mind, the following basis workflow was defined:

- For a given assignment, the teacher creates the assignment in the system and prepares a description, one or more submission templates - e.g. one for each development environment, and one or more sets of input files for testing. These may include a public set (that students can access) and a private set (that only the teacher can access).
- A student is then able to access the assignment information, download a template and start developing his/her project based on that template.
- During development time, the student can submit the project for validation. In that case, after submission the system will:
  - internally compile the project and execute it (if compilation is successful),
  - optionally perform the public tests.
  - Store the textual output of compilation and the graphical output of execution (in the form of one or more snapshots, or a video) and make them available to the student.
- At any time (typically after the assignment deadline), the teacher can check the status of the submissions, in terms of compilation results and graphical output, and have access to the code and executable files.

The prototype system presented in this paper is based on this workflow, as described in the next sections

4. GraphJudge architecture

The presented workflow led to the core architecture of the system, consisting of three major components:

- **GraphJudge Manager**: the service that deals with uploaded submissions, distributes the workload by the workers (see below) and collects results
- **GraphJudge Workers**: the sub-systems in which compilation and execution occur; to ensure isolation and support for different operating systems, each worker is a virtual machine with a pre-defined operating system, and remotely controlled by the Manager.
- **GraphJudge Interface**: The interface through which teachers setup the assignments and consult results, and students access assignments, submit projects and consult results. This interface invokes the Manager to handle the submitted projects

The following sections describe these components and their functioning in more detail.

4.1. GraphJudge Manager

The GraphJudge Manager consists of a series of scripts, running on a server. These scripts were developed in Python [py10], due to its flexibility and cross-platform support. Given a project or a set of projects, the scripts are responsible for:

- Choosing the correct worker(s) according to the type of projects (e.g. Windows or Linux)
- Copying the project files to the workers
- Invoke compilation and execution on the workers
- Collect and store the results from the workers

Being the workers virtual machines, communications between the manager and the workers are network-based, namely via SSH [SSH11] for remote control, and SCP [SSH11] for copying files in both directions.

4.2. GraphJudge Workers

One of the goals of the system is to support different types of operating systems. It should also be possible to capture the graphical output of the program execution. Compiling and executing projects on the same environment as the GraphJudge Manager would either limit the number of operating systems to one (the same in which the Manager was running), or require the setup of cross-compiling and emulation on that system. It would also raise some security issues, as the Manager would be more exposed to direct tampering by the executed programs.

Taking all this into consideration, one of the main design decisions was to have the compilation and execution of projects performed in separate machines (at least one for each operating system supported), devoted to those tasks and with very few permissions in terms of network and file system access.

However, having multiple physical machines for this purpose would be an expensive solution, and unpractical in terms of setup and handling. Instead, it was decided to use virtual machines as an alternative. These are much more manageable and resource-effective, while still guaranteeing the requirements of isolation and output capturing specified above. Furthermore, they have as additional advantages the ability to easily revert changes made by e.g. a malfunctioning or malicious program, and the possibility to be easily copied for personal testing of a particular environment by teachers and students (provided that software licensing is properly dealt with). Nevertheless, as the communication is network-based, the system can cope with both virtual and physical machines as workers, if necessary (e.g. for Mac OS X systems that are not trivial to virtualize).

Each worker is therefore a virtual machine that must have installed one of the operating systems that are to be supported by GraphJudge (currently Windows or Linux), and have installed:

- an SSH server that supports SCP, to which the Manager will connect and use to invoke scripts and copy files
- a Python interpreter
- the GraphJudge Worker Python scripts
- the compilation tools and libraries required for the type of assignments to be supported (e.g. C++ compiler and GLUT and GLUI libraries)
- a screen capture tool

For the creation and management of virtual machines, the VirtualBox [Ora10] software was used. Its cross-platform support in terms of hosts and clients, the possibilities in configuring network and graphics, the ability to access via VNC and the command-line interface for scripting made it a very flexible tool for the purposes of this work.

The following sections detail the two main tasks carried out by the Workers: automated compilation and execution, and output capturing.

4.2.1. Automating the compilation and execution

As mentioned before, the type of projects to be processed may be based on different operating systems and development environments. The GraphJudge system was built in a modular way, so that different building tools (and even languages) could be accommodated at any time. In the prototype developed, the configurations that were tested were the following:

- **GNU Make-based build system** (for Linux OS based projects) - build information in the form of a makefile
- **Microsoft Visual Studio build system** [Mic10] (for MS Windows based projects) - build information in the form of a project file (.vcproj or .vcprojx)

Each configuration is supported by information on how to invoke the building tools and how to capture the results of compilation (e.g. console output, log files).

The management of the process is performed by GraphJudge Manager, as described in section 4.1, which copies the projects for a predefined location in the worker via SCP and invokes the building and execution script (B&E script) for the correct configuration on the Worker, via SSH.

The B&E script can compile and execute a single project or a batch of projects, managed in queues. It can be run in three different modes:

- **Compile**: a queue stores all project paths to be compiled. The compile thread gets the items from the queue and launches a shell process that executes the actual building tool with the project folder or file's path as an argument. The compilation output is collected either from console or from a generated log file and stored for later retrieval by GraphJudge Manager
- **Execute**: like in compile mode the paths of projects to be executed are stored in a queue and the execute thread gets items, searches for the project executable and launches a shell process that executes the project, optionally with a test as a command line argument. While this process is running, the script is able to capture its graphical output. Some options are available in the B&E script to control output capturing (see below).
- **Both**: this mode corresponds to compile followed by execute, i.e., on successful compilation, the project is added to the execute queue.

The separation between the compilation and execution is useful when multiple executions are required on the same compiled project, e.g. when different tests are to be applied by a student, or when the teacher wants to run specific tests on a project (which has been compiled on submission).

The B&E script provides some options to specify the projects to be processed and the behavior of output capturing:

- **single project path**: path to a single project.
- **projects file system path**: path to a directory containing a set of projects to be processed. Used to compile multiple projects.
- **execution capture mode**: when execution occurs, this defines the capture mode. Currently only the image modes are implemented, and video mode is being developed (see next section for details).
- **execution time out**: sets the time limit in seconds for the execution of a project, after which forcefully terminates it, if it has not ended before. The default is 5 seconds.

The details on the process of capturing the graphical output are detailed in the next section.

After compilation and execution of a project, the corresponding results are stored in a log file and an image or video. The GraphJudge Manager, which is controlling the process, copies at this point those results from the worker to the Manager server, for later publication.

4.2.2. Automating the capture of graphical output

The ability to capture and store graphical output from the execution is one of the differentiating goals of the proposed system, even more so considering that such graphics can include OpenGL-based 3D graphics, which may require specialized capture software.

Two issues arise when considering this type of capture: how to set it up, and what type of data will be stored.

The set-up is important, as it requires the correct identification of the window (or windows) corresponding to the application, and the ability to monitor the capture process in an automated way. These procedures vary depending on the operating system. On Linux (X Windows) one has to obtain the window identifier based on process information, and that identifier can then be used by the capture program (**ImageMagick's import** [Ima99]) to do the actual capture. On Windows the capture can be done using a third-party software that captures the contents of a window based on the window name and supports a command-line interface for scripting (in the current implementation **MiniCap** [Sof10] is being used).

The type of data captured is important, as it may influence the type of capture system. In our context, the following alternatives were considered: single timed snapshot, set of snapshots at regular intervals, or an actual video.

In single timed snapshot mode, a snapshot is captured after a small period of time has elapsed since the beginning of the execution. The idea is that some time is provided for initial loading of the program and content, and that the snapshot is taken already with the complete initial output available. This is suitable, e.g., in assignments where the goal is to create a static 3D scene or model, and an initial viewpoint provides an overview of the content. In these cases, this snapshot may be enough for the teacher to assess the fulfilling of requirements.

With a set of snapshots at regular intervals, one can capture multiple instants of the program execution. This can be seen as a "low-frame-rate movie", and is useful in settings where there is some sort of animation in the program - either object animation or camera animation - and it is important to have views of the different states of the scene, but it is not really relevant to have access to all the instants of execution. This can be easily implemented based on the timed snapshot mode.

Finally, the video is the more complete type of capture considered here, as it allows to document the execution in a more detailed way. However, capturing video raises some issues in terms of the resources needed for its creation. Simultaneous graphics output and video capture can lead to degraded performance, influencing both the executing program and the quality of the video. Furthermore, the video capture can generate a large file, which may be an issue when considering a large number of submissions. Therefore, it should only be used when there is an actual need for a fine-grained coverage of the execution. In the current version of GraphJudge, video capture is not implemented yet.

4.3. GraphJudge Interface

As stated in section 4.1, the GraphJudge Manager consists of a set of scripts that manage workers and their tasks. For the actual setup of the assignments and project submission, a user interface was needed.

One of the purposes of this project is to integrate it with existing e-learning platforms, such as Moodle [Moo09], in which case, the submission and reporting would be handled by those platforms. However, the ability to use the system in a standalone way is also interesting and provides flexibility in terms of development.

Therefore, a prototype web interface was developed to handle the more common user-related tasks and invoke the GraphJudge Manager. This prototype was developed in Django [Dja05], as it is based on Python, thus integrating easily with GraphJudgeManager, and it is a powerful tool for rapid development of web sites.

The interface contemplates the following tasks:

- **Students/groups**:
  - Get information about the projects to be developed, such as problem description and base structure for files to use in development.
  - Make submissions and view results.
- **Teachers**:
  - Add and manage classes
  - Add and manage groups
  - Add and manage group elements
  - Add and manage projects
  - List submissions or search by class, group and projects
  - View submissions results

The more important components of the interface will be the pages that provide to students and teachers the detailed results of a given submission, and the aggregate page that shows the teacher information about all the submissions for a given assignment. These are the pages that will actually contribute to, on one hand, reduce the number of incorrect or incomplete submissions, and on the other hand provide the teacher with information to help him in the assessment. Images of the prototype can be found on section 5

5. Prototype testing results

In the previous section the different components of GraphJudge were described, namely the Manager, the Workers, and the Interface. This section presents the prototype that was developed which implements the main functionalities of the components referred.

The prototype currently runs on a computer with Ubuntu, with a set of packages required for running the Manager and the Interface, namely:

- Python (Manager/Interface)
- Apache web server (Interface)
- Django (Interface)
- OpenSSH client (Manager - Connection to Workers)
- VirtualBox (Manager - to host the Workers)

- The actual GraphJudge Manager and Interface scripts (written in Python)

Below are some images depicting the functioning of the interface. Figure 1 shows the details of a given project, that currently include the deadline, a link to a file with the problem description, and a link to the template source files. Figure 2 shows the submission form, where the student can indicate the kind of environment that his submission corresponds (e.g. Linux or Windows). Figure 3 contains the submission results page, where after submission the student can access the compilation log and a preview of his program when it was executed on the server. This allows the student to check if what the teacher will see is what the student expected, and demonstrates the achievement of one of the main goals of the project.

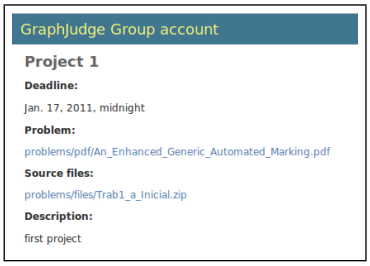

Figure 1: *Students' view of project details*

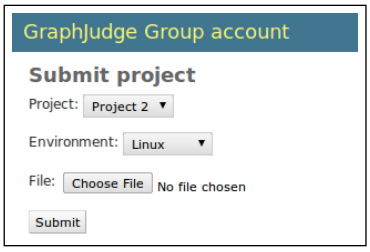

Figure 2: *Students' submission form*

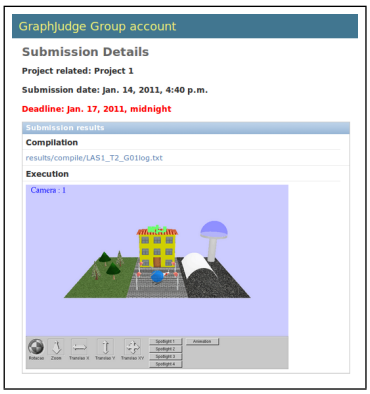

Figure 3: *Student's view of the results*

Teachers have a set of management options (see figure 4) of which the submission search form (figure 5) and the list of submissions (figure 6) are the more relevant. As it can be seen, the teacher can have an overview of the various submissions, including a preview of each submission, if available.

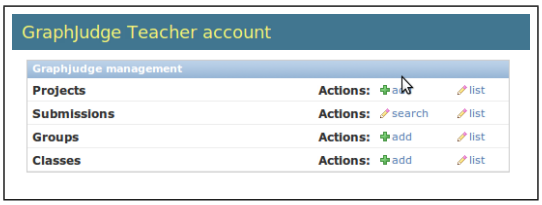

Figure 4: *Teacher's options*

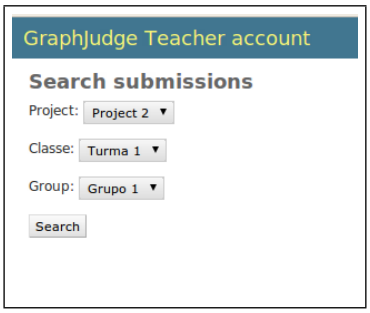

Figure 5: *Teacher's form for searching submissions*

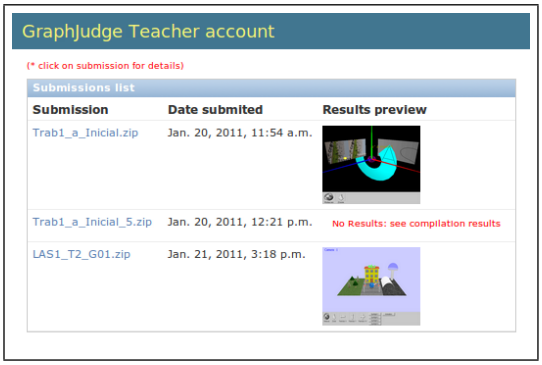

Figure 6: *Teacher's list of submissions*

Two Workers were implemented as virtual machines in VirtualBox. Ubuntu was installed on one of them, and Windows 7 on another. Some software had to be installed on the workers to support the system. Below is a list of the software (in the cases where there are differences between the workers, these are indicated):

- OpenSSH server (to receive connections from Manager)
- Python (to run the Worker scripts)
- Compilation tools: Build-essentials on Ubuntu, Microsoft Visual Studio 2008/2010 on Windows
- OpenGL, GLUT and GLUI libraries
- Image capturing software: ImageMagick on Ubuntu, MiniCap on Windows
- The actual GraphJudge Worker scripts (written in Python), that are invoked by the Manager via SSH

It is in the workers that the main activities - compilation, execution and image capture - occur. All are handled by the B&E script, that is invoked by the Manager through SSH. On a compile and execute cycle, the compilation logs are silently stored in background, but the actual execution graphical output occurs in the foreground windows, so that it can be captured. Figure 7 and figure 8 show a running program

as it is executed and captured in a Linux and Windows workers, respectively.

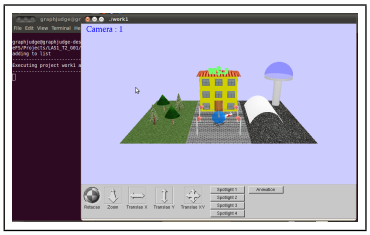

Figure 7: *Graphics capturing (Linux)*

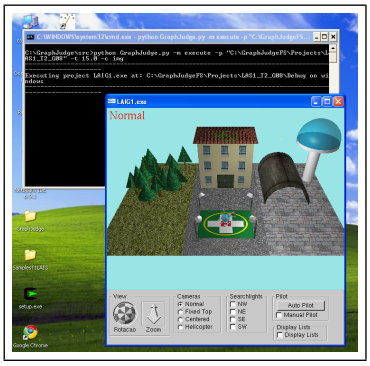

Figure 8: *Graphics capturing (Windows)*

6. Conclusions and future work

This paper presented a system for the automatic submission and validation of academic assignments with graphical output, to assist students and teachers in their assessment by capturing and storing images of the execution output. The work was developed to support Computer Graphics' courses where this type of assignments is common. This system allows students to do an early validation of their submission in terms of compilation and output, and to be sure that the teacher will see the same results. For the teachers, it eases the burden of compiling and executing and provides visual information of the execution of the submissions in an effective format. The system was developed with multiple platforms and development environments in mind, and it is therefore modular and extensible.

The current prototype includes an interface for teachers and students/groups that allows assignment creation and submission analysis by the teacher, and assignment access, project submission and results analysis to the students. It currently supports Linux and Windows environments.

The system is currently undergoing tests and improvements, and it is expected to be tested in some of our institutions' courses in a near future. Some of the improvements being considered are the introduction of video capture, a test module for image analysis to assist in the evaluation, and the possibility to define file validation rules. Regarding interaction, some ideas are being discussed for a system to batch record some actions, but already with the current implementation it would be possible to e.g. manipulate the scene by providing pre-configured camera movements in the templates and triggering them via simulated keystrokes. It has also being considered the integration of the system with existing e-learning platforms, namely with Moodle, as it is being used in our institution.

References

[ABL09] ASHOO S. E., BOUDREAU T., LANE D. A.: Programming contest control system, June 2009. 2

[BGNM04] BLUMENSTEIN M., GREEN S., NGUYEN A., MUTHUKKUMARASAMY V.: Game: A generic automated marking environment for programming assessment. *Information Technology: Coding and Computing, International Conference on 1* (2004), 212. 1

[Dja05] DJANGO SOFTWARE FOUNDATION: The django framework. http://www.djangoproject.com/, 2005. 4

[EKdW09] ELDERING J., KINKHORST T., DE WERKEN P. V.: Domjudge - programming contest jury system, 2009. 2

[Ima99] IMAGEMAGICK STUDIO LLC: Imagemagick. http://www.imagemagick.org/, 1999. 4

[IPVB08] IVANOVIC M., PRIBELA I., VESIN B., BUDIMAC Z.: Multifunctional environment for e-learning purposes. *Information Technology: Coding and Computing, International Conference on 38* (2008). 1

[JGB05] JOY M., GRIFFITHS N., BOYATT R.: The boss online submission and assessment system. *J. Educ. Resour. Comput. 5* (September 2005). 1

[Khr11] KHRONOS GROUP: Glut - the opengl utility toolkit. http://www.opengl.org/resources/libraries/glut/, 2011. 1

[Mic10] MICROSOFT CORPORATION: Microsoft Express. http://www.microsoft.com/express/, 2010. 3

[Moo09] MOODLE TRUST: Moodle. http://moodle.org/, June 2009. 4

[Ora10] ORACLE: Virtualbox. http://www.test.org/doe/, 2010. 3

[py10] Python programming language - official website. http://python.org, 2010. 2

[Rad10] RADEMACHER P.: Glui user interface library. http://glui.sourceforge.net/, 2010. 1

[Sof10] SOFT M.: Minicap is a minimal screenshot capture app, 2010. 4

[SSH11] OpenSSH. http://www.openssh.com/, Feb. 2011. 2

[ST09] SHREINER D., THE KHRONOS OPENGL ARB WORKING GROUP: *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Versions 3.0 and 3.1*, 7th ed. Addison-Wesley Professional, 2009. 1