A Programmable Tutor for OpenGL Transformations

Carlos Andújar and Pere-Pau Vázquez

Departament de LSI, Universitat Politècnica de Catalunya, Barcelona, Spain

Abstract

Computer Graphics is a growing field that is becoming more and more present in Computer Science studies at University level. This has lead to the creation of a large number of support materials, such as tutorials, white papers, and teaching books. Geometric transforms is commonly one of the topics that all Computer Graphics courses must deal with, and, traditionally, is one of the issues that students find difficult to master. Some of the problems come from the difficulty of using a blackboard to represent 3D objects, and it is therefore very convenient to have tutorial applications that are able to help through the learning process. In light of this, we have built a geometric transformation tutor that is intended to improve student understanding of the operations that are carried out on the OpenGL transformation matrices, and the results in terms of visualization. The application achieves a nice balance between the flexibility provided by a script language and the easiness of use. A key benefit of the application is the possibility of writing interactive tutorials demonstrating topics such as camera and object animation.

Categories and Subject Descriptors (according to ACM CCS): I.3.4 [Computer Graphics]: Graphics Utilities K.3.1 [Computers and Education]: Computer Uses in Education

1. Introduction

Although Computer Graphics is a relatively recent field, as compared to other classical Computer Science areas such as programming, it is now a very active research and development field. Computer Graphics teaching is commonly faced with two kinds of teaching paradigms, the face to face teaching using a blackboard, and lab sessions. The contents of a classical Computer Graphics course include the explanation of 3D transformations and their implementation using the OpenGL matrix operations. For the Computer Graphics practitioner, it is extremely important to master transforms. Transforms are used to position, reshape, and animate objects, lights and cameras. Transforms are used also to ensure that computations are carried out in a particular coordinate system, and project objects onto a plane in different ways.

Commonly, students find difficult to understand how the transformation operations are affecting the camera and why a certain view looks as it does. Usually, students consolidate the theoretic aspects in lab classes, but with a high amount of effort and time devoted to this sole feature. Most of the time, they are not able to infer where the camera is placed and where it is looking at.

In this paper we present a flexible tutorial application intended to make the learning process of the geometric transformations in OpenGL faster. Our context is an introductory Computer Graphics course of one semester taught to undergraduate students in Computer Science. Moreover, it is a compulsory course, so the students may or may not be motivated by the subject. Laboratory classes help to consolidate the fundamental concepts that are taught at theory classes, and they are tightly coupled, that is, the concepts seen in lectures are practiced in laboratory classes within a couple of weeks. In our laboratory classes, we use OpenGL as 3D API.

Our system allows to visualize both the projected view that a certain camera configuration will yield, together with a representation of the actual position of the camera, the viewing frustum and the object, so that the user can really see where the camera is placed with respect to the scene he or she wants to render. The main features of our application are:

- Continuous knowledge of the operations that are applied to define the OpenGL matrices.
- The combination of two views, one for the rendering result and another that shows the camera configuration in

relation to the object, avoids the user to *lose* the object or the camera.
- Two navigation modes: free camera and user camera. The first one allows the user to interact with the object and see the transformations that are applied to it. The latter permits the inverse process, the user issues OpenGL commands and sees how they affect the visualization and camera configuration.
- Flexible modification of the commands applied. The user can add, remove, and modify the parameters of the OpenGL transformations easily and interactively.
- A script language for computing the parameters of OpenGL commands.

The rest of the paper is organized as follows. We review previous work in Section 2. Section 3 introduces the teaching goals addressed by our application. In Section 4 we detail the features of our tool. Section 5 covers the experience we have had with the tutor. Finally, Section 6 presents some concluding remarks and some lines of future work.

2. Previous Work

Computer graphics is quite a difficult subject to teach, particularly with traditional face-to-face teaching. Consequently, several tools have been built in order to improve students' learning process.

UisGL is a portable set of C++ classes that provide support to the management of data structures and transformations [1]. Its main objective is to provide the students with a core set of classes that can be used to develop algorithms in a top-down manner.

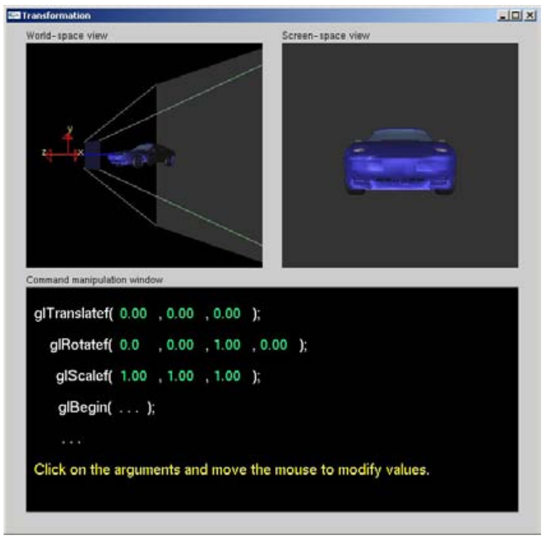
Naiman's teaching modules [2] were one of the first attempts to develop a set of working examples as supplemental material. They cover fundamental Computer Graphics topics presented in introductory courses and consist not only of executable code, but also descriptions of algorithms, demonstrations of the basic concepts, and exercises.

Wernert [3] presents an environment for developing and analyzing graphics algorithms. The system is based upon Iris Explorer. Its main objective is to help the students to bring into the practice the knowledge acquired during lectures. Although the environment is highly flexible, the use of networks of modules to facilitate coding sometimes results in complex designs with a high amount of modules.

Fortunately, the technology we have now can be used to overcome most of the problems those early projects could suffer from, such as rendering speed, or the fact that some software was proprietary.

As stated in Klein *et al.* [4], teaching Computer Graphics with traditional technologies and tools makes difficult to the student to understand *real life* examples. As shown, the traditional focus is to develop sample programs that illustrate concrete concepts. Unfortunately, in former times, these programs were implemented using expensive hardware and students did not had either the software nor the hardware to experiment with them at home. Nowadays, things have changed. First, most commodity PCs come with a graphics card that is powerful enough to execute complex Computer Graphics programs. And second, there is a relatively large amount of freely available software that is present in a wide range of hardware platforms and operative systems, such as Qt system [5] or OpenGL. With this in mind, Klein and Hanish [4] propose a Computer Graphics course that provides an integrated view of the theoretic concepts together with a set of Java applets that illustrate the concepts. Similarly to other applications, this online graphics course does not provide a great flexibility for the user to modify the code to be executed. A similar approach is taken by Pan *et al.* [6], and by Ullrich and Fellner [7]. In this last case, emphasis is put, not only on the features provided by the tools, but also on their design, in order to obtain a set of components that are easily adapted or combined with other applications. Other systems also address similar objectives [8, 9, 10].

It is worth to mention the Nate Robins tutors [11] (see Figure 1), which are similar to ours in spirit (they also focus on OpenGL operations and provide samples of modifiable code). These are a set of OpenGL applications that address simple problems such as texturing, light placement, and shapes.



```
glTranslatef( 0.00 , 0.00 , 0.00 );
   glRotatef( 0.0  , 0.00 , 1.00 , 0.00 );
     glScalef( 1.00 , 1.00 , 1.00 );
       glBegin( . . . );
       . . .
```

Click on the arguments and move the mouse to modify values.

Figure 1: *Snapshot of the Nate Robins' transformation tutor*

As stated by Cunningham [12], nowadays technology allows us to teach Computer Graphics in two different ways: a traditional approach with higher emphasis in fundamental algorithms and techniques, and a high-level, API-oriented approach that can probably attract a greater audience. Our approach is a hybrid method, such as Angel's system [13], probably more focused on fundamental concepts, but with a tight coordination with laboratory classes, that are held using OpenGL. The fundamental concepts explained in class are experimented in laboratory within one or two weeks.

Sung and Shirley [14] compare the top-down and bottom-up approaches in teaching introductory graphics courses. They argue that a top-down approach, that is, starting from the functional modules of applications (rather than from foundational algorithms such as triangle rasterization) are more suitable for mature students.

3. Teaching goals, methods and requirements

This project was undertaken to help students understand and acquire an intuitive notion of several concepts related with geometric transformations from a learn-by-practice approach.

The specific topics we wanted the students to practice were those transform-related topics included in most introductory courses on Computer Graphics. In particular, the tutor was conceived to address the following topics:

- Viewing transforms: camera analogy, orthographic, and perspective camera models.
- Modeling transforms: basic affine transforms (translation, rotation, scaling, and shearing) and concatenation of transforms.
- Matrix stacks and their use in computer animation.
- Alias and alibi interpretations [15].
- Camera and object animation.

Competences and problem-solving skills concerning the above topics students should develop include:

- Define a suitable camera using OpenGL commands for a given model and camera constraints.
- Describe the image on the viewplane resulting from a given model and camera definition.
- Implement basic camera animation techniques such as rotation, zooming, panning, and flythrough.
- Implement simple and hierarchical object animation.

An additional goal was to provide the students with an easy-to-use graphical tool for debugging transformation code, along with an interactive tool for producing, practicing and checking practical exercises involving the above skills.

The analysis of the above goals lead us to identify the application requirements described below. The application should allow the user to:

- Import a 3D model and define a suitable default camera.
- View and edit the OpenGL code defining the modelview and projection matrices.
- Change interactively the parameters of OpenGL commands and view the effect on the camera and the objects.
- Write and edit small programs using a scripting language to assign variables usable as OpenGL function parameters.
- View the model from the current camera's viewpoint and from an exocentric viewpoint showing the camera's frustum with respect to the scene objects.
- Browse a tutorial guiding the students through different lessons/exercises.
- Export OpenGL commands and script functions into C/C++ code.

Two key features of this application concept are the programmability (via a script language) and the possibility of writing interactive tutorials for guiding the learning process. These tutorials comprise both code and step-by-step instructions.

Additional requirements of the application comprise a simple and intuitive user interface, and the use of free, multiplatform software libraries.

4. The programmable tutor

One of the main goals in the design of the application was to achieve a suitable balance in the flexibility/simplicity trade-off. The key idea was to provide instructors and students with a flexible, programmable tutor while abstracting students from distracting code. Abstracting the student from other programming details (model import, window creation, input management, user interface) promotes the focus on code dealing with transforms.

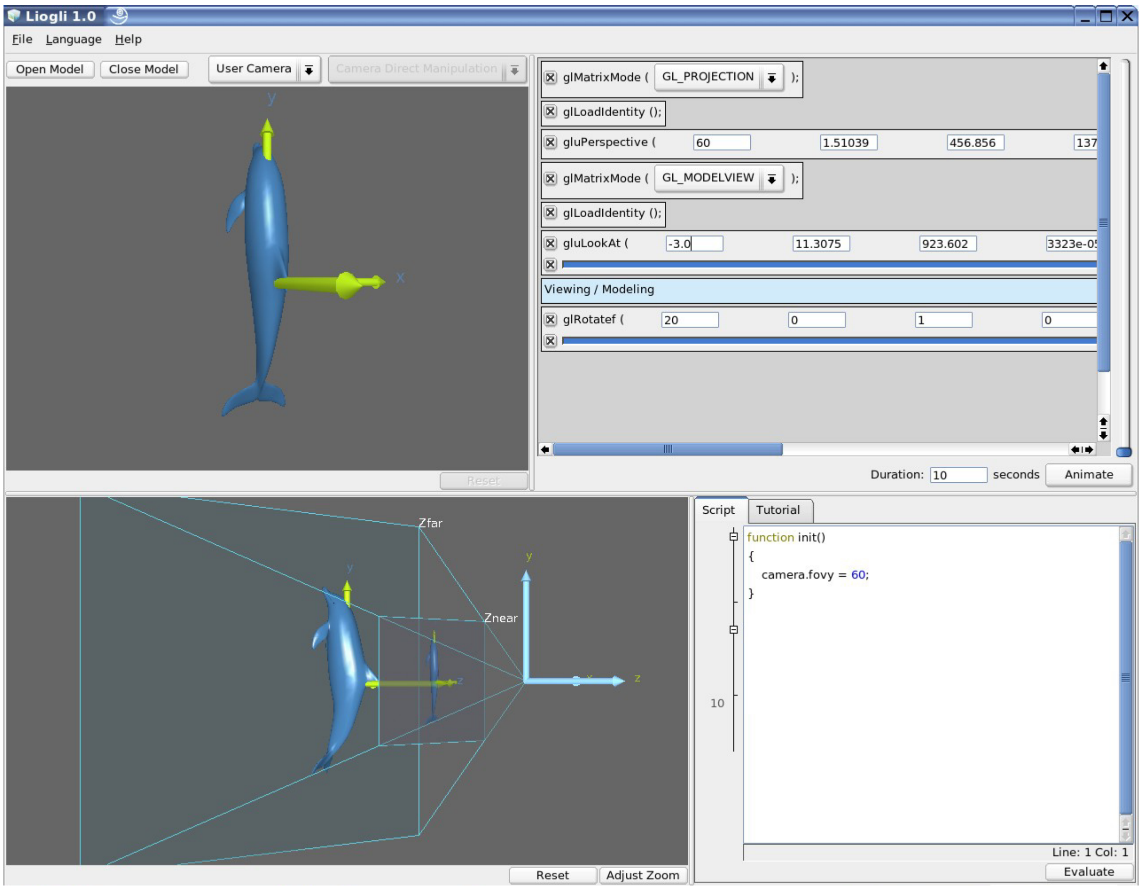
Regarding the user interface design, we strongly believe that presenting the student with simultaneous views of the OpenGL code and the associated effects on the camera and scene objects would increase the pedagogic value of the application. We also argue that separating actual OpenGL functions from the code computing parameters for these functions promotes a better understanding of the matrix operations involved.

As a result, the application's user interface is divided into five areas (Figure 2):

Camera view This view shows the projection of the (possibly transformed) objects onto the viewing plane corresponding to the current camera. From the user's point of view, this view works with two cameras. The *free camera* allows the interactive exploration of the model using direct manipulation and preprogrammed camera rotations, zooming and panning. This camera is completely independent of the user-provided OpenGL commands and it is provided as a simple and direct way for exploring the model. The *user camera* corresponds to the OpenGL commands provided by the user through the OpenGL command editor (see below).

Exocentric view This view shows the model and the current camera's frustum from an exocentric viewpoint. The user is able to rotate, zoom, and pan the exocentric camera at any moment through direct manipulation.

OpenGL command editor This window shows a user-editable collection of OpenGL functions used for defining



```
glMatrixMode ( GL_PROJECTION );
glLoadIdentity ();
gluPerspective ( 60  1.51039  456.856  137
glMatrixMode ( GL_MODELVIEW );
glLoadIdentity ();
gluLookAt ( -3.0  11.3075  923.602  3323e-0
glRotatef ( 20  0  1  0
```

Duration: 10 seconds

```
function init()
{
   camera.fovy = 60;
}
```

Figure 2: *Application's user interface. Camera view (top left), Exocentric view (bottom left), OpenGL command editor (top right), Script editor and Tutorial view (bottom right).*

both OpenGL's MODELVIEW and PROJECTION matrices of the user camera. The functionality of this key component is described in Section 4.1.

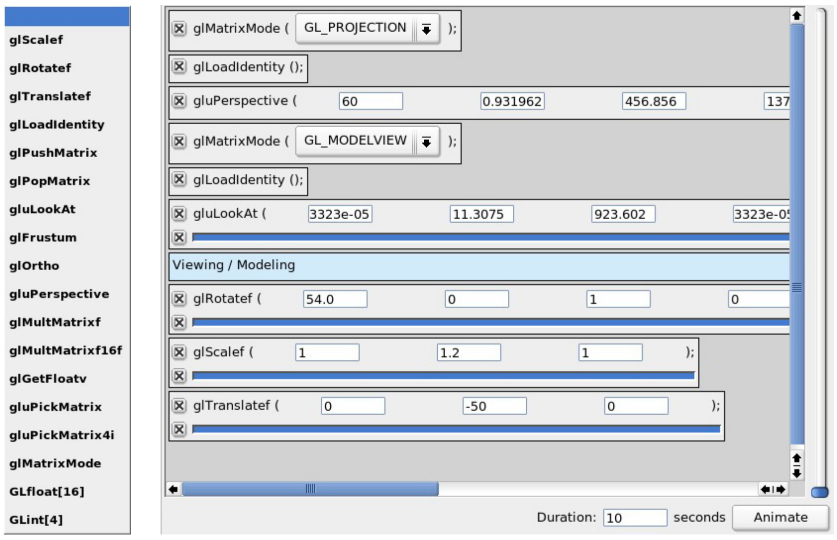
Script editor This component allows users to define their own functions to compute values usable as OpenGL function parameters (Section 4.2).

Tutorial view Our application allows instructors to design custom-tailored tutorials to make the students work on a specific topic. This component basically displays an HTML document which can include an introduction to the topic and a series of experiments. For example, instructors can provide a tutorial for comparing different alternatives for implementing a zoom operation.

The user interface is completed with a toolbar showing the list of supported OpenGL instructions (see Figure 3). Note that this subset is enough to implement complex camera and model animation effects.

4.1. The OpenGL command editor

This component represents the sequence of OpenGL calls for defining the modelview and projection matrices of a user camera (Figure 3). For the sake of simplicity, students do not need to type function names; commands can be inserted, removed and moved using the drag-and-drop technique. Function parameters can be either literal values or an object identifier (constant or variable). Each parameter has an associated tooltip describing its type (e.g. *GLfloat*) and purpose (e.g. *eyex*). This tooltip is automatically displayed when the mouse pointer moves over the parameter. Literal values can be changed interactively. Numeric values can be adjusted either by typing the desired value or through direct manipulation. The user simply clicks over the desired parameter and drags the mouse to increment/decrement the value. The support of object identifiers as parameters greatly enhances the possibilities of the application, allowing the student to program camera and object animations (see Section 4.2). Each OpenGL command on the list can be temporarily disabled through a checkbox. This feature helps students understand



```
glScalef
glRotatef
glTranslatef
glLoadIdentity
glPushMatrix
glPopMatrix
gluLookAt
glFrustum
glOrtho
gluPerspective
glMultMatrixf
glMultMatrixf16f
glGetFloatv
gluPickMatrix
gluPickMatrix4i
glMatrixMode
GLfloat[16]
GLint[4]
```

```
glMatrixMode ( GL_PROJECTION );
glLoadIdentity ();
gluPerspective ( 60  0.931962  456.856  137
glMatrixMode ( GL_MODELVIEW );
glLoadIdentity ();
gluLookAt ( 3323e-05  11.3075  923.602  3323e-05
Viewing / Modeling
glRotatef ( 54.0  0  1  0
glScalef ( 1  1.2  1 );
glTranslatef ( 0  -50  0 );
Duration: 10 seconds  Animate
```

Figure 3: *Interface components. Toolbar with supported OpenGL commands (left). OpenGL command editor (right)*

the role of each transformation in the final results. For instance, the student can disable a *glLoadIdentity()* command and check the result.

In order to distinguish viewing from modeling transformations, the user can freely move a separator labeled *Viewing/modeling* on the OpenGL command view (Figure 3). Transforms above and below the separator are considered respectively camera transforms and modeling transforms. Students can move the separator above and below and analyze the changes on the exocentric view. Obviously the separator placement has no effect on the camera view as the resulting modelview matrix is the same.

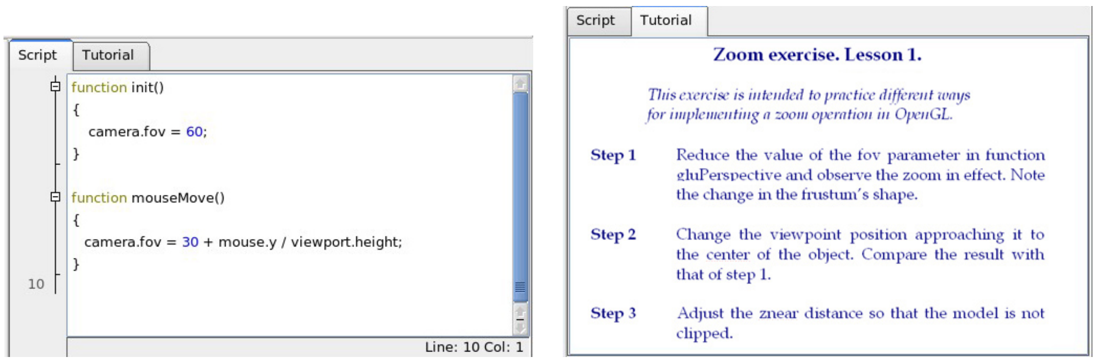
Modeling transformations and hierarchical animations often involve a long list of OpenGL commands. In this case it can be pedagogical to visualize the effect of applying each OpenGL command incrementally. This can also be used by the students to check their solutions of self-tests. The application supports the incremental application of the transforms using interpolation during a user-defined time interval. Suppose the OpenGL commands define the matrix product $T = T_1T_2\dots T_n$. At a given instant t, the incremental matrix is computed by $T'i(t)T_{i+1}\dots T_n$ where $T_i'(t)$ is computed as a smooth interpolation from the identity matrix to T_i. For translate, rotate an scale commands, a straightforward linear interpolation of the associated parameters is used. For the gluLookAt function[16], we have opted for animating its decomposition into $R(-h,0,1,0)R(-p,1,0,0)R(-r,0,0,1)T(-\text{eye})$, where h, p, and r are resp. the head, pitch and roll rotation angles, and *eye* is the viewpoint location. For glLoadMatrix() and glMultMatrix commands we decompose the matrix into a sequence of basic transforms using the algorithm described in [17] and animate them simultaneously. A progress bar below each animatable instruction (the blue rectangles in Figure 3) shows to which extend the current command is affecting the final matrix. The animation order can be reversed, i.e. computing $T_1T_2\dots T_{i-1}T_i'(t)$. This feature helps students understand better the alias and alibi interpretations of geometric transforms.

4.2. The script editor

Allowing the user to interactively change the sequence of OpenGL commands and their parameters offers many teaching possibilities. However, without the support of a full programming language some exercises like implementation of navigation modes cannot be demonstrated.

The script editor allows instructors and students to define a collection of functions which compute values that can be used as OpenGL command parameters. In our prototype implementation the scripting toolkit adopted is QSA (Qt Script for Applications) [18], an easy-to-learn, cross-platform interpreted scripting language. Qt Script is based on the ECMAScript standard (like JavaScript) and its syntax is somewhat similar to C/C++.

Communication between OpenGL commands and script functions and object persistence between different executions are two major issues. One possibility is to allow the user to place a function call inside an OpenGL command parameter. We discarded this option because it is not clear *when* this function has to be evaluated. We have decided to implement a simple to understand communication between OpenGL commands and script functions. Besides user-defined functions, a few function names are reserved



```
function init()
{
  camera.fov = 60;
}

function mouseMove()
{
  camera.fov = 30 + mouse.y / viewport.height;
}
```

Line: 10 Col: 1

Zoom exercise. Lesson 1.

This exercise is intended to practice different ways for implementing a zoom operation in OpenGL.

Step 1 Reduce the value of the fov parameter in function gluPerspective and observe the zoom in effect. Note the change in the frustum's shape.

Step 2 Change the viewpoint position approaching it to the center of the object. Compare the result with that of step 1.

Step 3 Adjust the znear distance so that the model is not clipped.

Figure 4: *Interface components. Script editor (left). Tutorial view (right)*

and have a well-defined application meaning. These functions are:

- init() - this function is called once every time a new model is loaded; the user can also force its execution explicitly through a *reset* option.
- paint() - this function is called every time the OpenGL window needs to be repainted. The user can activate a timer so that this function is called in user-defined intervals.
- mousePress(), mouseMove(), mouseRelease() - these functions are called in response to mouse events.
- keyPress(), keyRelease() - these functions are called in response to keyboard events.

These functions in turn can include calls to user-defined functions. The application maintains some objects whose properties can be accessed from the script functions and used as OpenGL command parameters. Some objects are read-only. Some examples of read-only objects are the default camera (with properties such as the viewpoint position and target), the OpenGL window (size, aspect ratio), the current model (radius, bounding box), and the mouse state. Read-write objects include the user camera and user-defined variables.

The main benefit of the described approach is that the user knows exactly when the script code is executed and that it allows the implementation of interactive animations.

4.3. The tutorial view

In order to guide the students into their learning process, it is not enough to provide a tool where they can practice the different possibilities. It is also necessary to give them a set of exercises that help them both to explore the possibilities of the tool, and the solution to different Computer Graphics problems.

One of the benefits of our approach is the possibility of writing complete tutorials for training on specific topics. Each tutorial consists of a collection of lessons, and each lesson is a HTML document with step-by-step instructions (Figure 4), a list of OpenGL commands and a script program.

Our user studies have demonstrated the importance of providing this guidance to the students in order to promote the understanding of commonly misunderstood concepts.

4.4. Additional features

The application can export the OpenGL command list and the script functions to C++ code. This feature further encourages the use of the tutor by the students as a test bed for implementing camera and model animations.

5. Results, experiences and evaluation

We have implemented a prototype version of the tutor over OpenGL and Qt. As both technologies are freely available and portable, it can be installed into different platforms. The source code can be downloaded from `http://www.lsi.upc.edu/~virtual/tutorGT`.

The following experiment can be conducted to measure the effect of the use of interactive tutors in student performance. Subjects (undergraduate students) are randomly assigned to one of three groups. All subjects are instructed to follow a tutorial and then complete several practical exercises involving geometric transforms. Subjects in the first group are given a sample OpenGL application that they can modify to undertake the exercises (that is the current approach we follow). Subjects in the second group use the Nate Robins' transformation tutor [11], and subjects in the third group use our application both to follow the interactive tutorials and to perform the exercises. Analysis of variance (ANOVA) can be used to test hypotheses about differences between the means, considering completion times and student marks as the dependent variables and supporting software and exercise type as independent variables.

We plan to perform this experiment during the sixth week of the next semester course during a lab class. At this point students will have already been taught transformations. Obtained marks will only be considered for tool evaluation purposes. Once the experiment is completed, all students will have access to the developed tool.

We are also planning a long-term experiment that consists in recording user usage of the tool including number and duration of each session along with a log file of all actions performed during the session. This study will be conducted on a reduced group of volunteer students. This second experiment will help teachers to identify typical errors and common difficulties of the tool usage.

Although we have not introduced yet our system into our Computer Graphics course, we have already carried out a preliminary user study amongst 10 undergraduate students near the end of the semester. Our objective was to have an initial evaluation of the user acceptance through a simple questionnaire. Students were asked for their opinion, and if they found some desirable feature missing. We showed them an initial version of our tutorial which was still lacking some functionality.

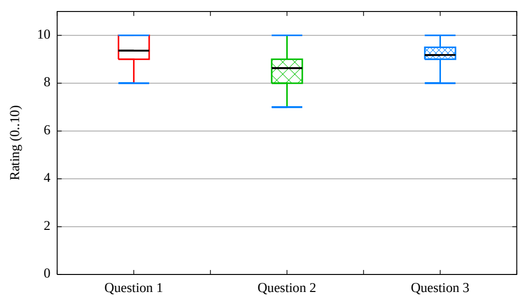


Figure 5: *Answers given by the students about the tool.*

All of them found the developed tool very useful to learn and practice. Figure 5 shows the answers to the three main questions in a 0 to 10 scale:

- Rate the usefulness of the tool.
- Does the tool help you to better understand modeling and viewing transformations?
- Does the tool help you to develop the final project?

The final project referred to in the third question consists in the implementation of a complete application for inspecting interactively 3D models with simple animations, which is carried out during the second half of the semester.

We are convinced that the introduction of our transformation tutorial will lead to improve the outcomes of students and to a faster mastering of OpenGL transformation operations.

6. Conclusions and future work

We have built a system for teaching and practicing geometric transformations in OpenGL. It has several advantages over previously proposed solutions including:

- Flexible addition and removal of OpenGL transformation commands.
- Interactive animation of the camera configuration process: The transformations involved into the camera configuration and positioning can be executed step by step, and transformations such as translations are animated by rendering all the translation path.
- Changing parameter values with direct manipulation and check the effects in the different views.
- A script language that allows us the definition of variables and methods that can be used to implement navigation modes.

Although the tutor can be freely used to experiment with OpenGL transforms, it is important to build a set of exercises to guide the student into the learning process. These exercises should cover different kinds of camera and modeling operations, such as translation, scaling, panning, rotations, etc. The tutorial documentation provides, when possible, different approaches to obtain the same result. This way, the students become familiarized with more manipulation possibilities.

As a future work, we plan to add a new command *drawModel(model)* that effectively sends the geometry of a particular model to OpenGL. That would allow to have several model transforms associated with the same model and to create several instances of the model at different positions.

We also plan to to build a set of other tutorial applications that cover other Computer Graphics concepts in OpenGL. Currently, an application for teaching the management of OpenGL stencil and accumulation buffers is almost finished. Our objectives are to implement other tutorials for: OpenGL primitives, texturing, ray tracing, illumination, and introduction to GPU programming. In some of the cases, this would simply imply to extend the set of supported commands.

Finally, we would like to convert the application into a class library so that existing applications can instantiate it in a separate window, in order to integrate it to the programming process. This can be easily done due to the features that allow extensibility of Qt.

7. Acknowledgments

The authors would like to thank Marc Lopez for the implementation of the described application.

References

[1] S. Grissom. uisGL: A c++ library to support graphics education. *Computer Graphics*, 30(3), 1996. 2

[2] A. C. Naiman. Interactive teaching modules for computer graphics. *Computer Graphics*, 30(3):33–37, 1996. 2

[3] Wernert E. A unified environment for presenting, developing and analyzing graphics algorithms. *Computer Graphics*, 31(3):26–33, 1997. 2

[4] Klein R., Hanisch F., and Strasser W. Web-based teaching of computer graphics: concepts and realization of an interactive online course. In F. Flückinger and A. Ninck, editors, *Computer Graphics Annual Conference Series, Conference Abstracts and Applications*, pages 88–93. NDIT/FPIT Bern, 1998. 2

[5] Trolltech. http://www.trolltech.com. 2

[6] Z. Pan, H.P. Lun, and R. Gao. Interactive learning of computer graphics algorithms. In S. Cunningham and D. Martin, editors, *Eurographics*, pages 1–6. Blackwell Publisher, 2003. 2

[7] Ullrich T. and D. W. Fellner. Computer graphics courseware. In J. J. Bourdin and H. McCabe, editors, *Eurographics*, pages 11–17. Blackwell Publisher, 2005. 2

[8] Reinhard Klein and L. Miguel Encarnação. An interactive computer graphics theory and programming course for distance education on the web. In *8th Int. PEG Conf.*, Sozopol, Bulgaria, May/June 1997. 2

[9] Guzdial M. Shabo, A. and J. Stasko. Addressing student problems in learning computer graphics. *Computer Graphics*, 29(3):38–40, 1996. 2

[10] Dieter W. Fellner. Minimal rendering tool: A research and teaching platform for 3d image synthesis. *IEEE Comput. Graph. Appl.*, 16(3), 1996. 2

[11] N. Robins. Nate robins - opengl tutors. http://www.xmission.com/ nate/tutors.html. 2, 6

[12] S. Cunningham. Re-inventing the introductory computer graphics course:. providing tools for a wider audience. *Computers and Graphics*, 24(12):293–296, 2000. 2

[13] E Angel. *Interactive Computer Graphics: a Top-Down Approach with OpenGL*. Addison-Wesley, 1997. 3

[14] Kelvin Sung and Peter Shirley. A top-down approach to teaching introductory computer graphics. In *SIGGRAPH '03: Educators program from the 30th annual conference on Computer graphics and interactive techniques*, pages 1–4, New York, NY, USA, 2003. ACM Press. 3

[15] Eric W. Weisstein. Alias transformation. alibi transformation. From MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/AliasTransformation.html. 3

[16] M. Woo, J. Neider, T. Davis, and D. Shreiner. OpenGL programming guide, 3rd edition, 2000. Specification document, available from http://www.opengl.org. 5

[17] Ken Shoemake. Polar matrix decomposition. pages 207–221, 1994. 5

[18] Trolltech. Qsa, qt script for applications. http://www.trolltech.com. 5