Fast Application Development to Demonstrate Computer Graphics Concepts

Pedro Pablo Gómez-Martín
Group for Artificial Intelligence Applications
Facultad de Informática
Universidad Complutense de Madrid
pedrop@fdi.ucm.es

Marco Antonio Gómez-Martín
Facultad de Informática
Universidad Complutense de Madrid
C/ Prof. José García Santesmases, s/n.
28040 Madrid (España)
marcoa@fdi.ucm.es

ABSTRACT

Computer graphics concepts have a high visual component. For that reason, teaching this subject should be enriched with the use of small applications showing concepts like near and far plane or objects hierarchy in real time. However, their development is usually time consuming. In this paper we propose the use of an open source engine called Nebula to create such examples. We will show that Nebula applications are easy to develop and modify, presenting three small applications that help to explain important concepts in Computer Graphics curriculum.

Categories and Subject Descriptors

I.3 [**Computing Methodologies**]: Computer Graphics; K.3.2 [**Computers and Education**]: Computer and Information Science Education—*Computer science education*

General Terms

Algorithms, Human Factors, Theory

Keywords

Computer graphics, game engines, education

1. INTRODUCTION

The Computer Graphics subject covers several concepts related to creation, representation and render of three dimensional environments. Unfortunately, most of them are hard to explain using just board and slides. The main limitations of these resources are their two dimensional nature and their lack of dynamism.

These restrictions often force the teacher to use rough examples. Teaching the coordinates hierarchy often ends with the lecturers using their arms, elbows and wrists to illustrate how the rotation of one of them affects the other ones.

The explanation of other concepts is even more difficult. The effect of the variation of the near and far planes is much better understood presenting some snapshots of a virtual environment rendered with different values of these parameters. Even better, teachers could use some interactive program where they modify the values at will during the explanation.

Nowadays, the use of laptops during lectures is widespread. Teachers are using them more and more to present small applications to teach concepts related to the current subject. For example, there are several simulators of small computers to teach Computer Architecture and Operating Systems [2], resources to show how algorithms work [3] or to teach typical AI techniques [1]. Computer Graphics lecturers should also use small applications like these ones, to exemplify the concepts they are explaining.

However, the amount of work developing such applications is usually prohibitive. In this paper we detail the use of an open source graphics engine that, though it is not meant to be used for that purpose, it eases the development of such applications.

The paper runs as follow. Next Section details the characteristics we think that applications should have to increase their pedagogical value. Section 3 describes Nebula engine, and the next one compares the two available versions. Next, it follows by a description of three small applications developed using Nebula that reveals that the amount of code needed is rather small[1]. Section 6 shows other resources available to improve the interactivity in the Computer Graphics lectures. The paper ends with some conclusions.

2. DEMOSTRATION FEATURES

Teaching computer graphics using just the board is pretty hard. Explanations are rather theoretical and teachers have to face with the task of drawing schematic pictures with no help.

Slides ease the task, because they allow lecturers to create more elaborated pictures *off-line* and present them at the right moments without the delay of drawing them. Nowa-

[1]Applications described in this paper are available in the first author's web page, `http://gaia.fdi.ucm.es/people/pedrop/development/CGDemos`

ITiCSE'06, June 26–28, 2006, Bologna, Italy.

days, the majority of teachers have laptops and overhead projectors available to enriched their lectures. They can get rid of the acetate slides and make use of applications such as Microsoft Power Point or Open Office Impress. These programs even allow to enrich slides with small animations.

Laptops make also possible the execution of applications in class. Lecturer can show the execution of the theoretical concepts she is teaching. For example, instead of present two or three static pictures about the variation of the rendered images when both near and far planes change, she can execute a program that allows her to change dynamically these parameters showing the result on real time.

We think that these programs should have these features in order to make feasible their use in a classroom:

- **Interactive:** the main advantage of these applications compared to the static slides is that they can render the images in real time. It is desirable to be able to change the parameters involved in the concepts been taught at execution time using widgets as sliders.

- **Easy to develop:** creating these applications requires time. If the effort is excessive we can hardly expect teachers to develop them. It would be great that a huge amount of that applications would be available but, as we will see in Section 6, the set of applications is still small.

- **Easy to modify:** when showing these applications, students use to pay more attention and they understand the concepts quicker. For that reason, new questions related to the current subject appear. Pedagogically, it is better to answer these doubts at that very moment, instead of postpone them to the next class. If teachers could not only change the foreseen parameters but also the application to adapt it to the new pedagogical requirement, the learner experience will improve even more.

- **Availability:** all the resources used in class should be available for the students. Ideally, they should not consider these programs just as plain "executable examples" of the slides but as applications to play with. If their source code is available and easy to understand and modify, students could easily make their own versions of the application, consolidating their knowledge.

Next Section describes the Nebula game engine. Once its core concepts has been introduced, Section 4 compares the two main versions available. Later, Section 5 describes three small application that we have developed with them.

3. NEBULA DESCRIPTION

Nebula is a graphic engine used to develop interactive 3D games, distributed under LGPL license by RadonLabs [4]. There are two different versions available, known as Nebula 1 and Nebula 2. The latter has advanced features, needed for almost every modern 3D game (as programmable *shaders*); nevertheless, for our more modest goal, Nebula 1 (last released in January, 2004) has enough interesting characteristics. We have developed the three application using both versions. Section 4 details the advantages and drawbacks of them.

Nebula is written in C++, and designed to help developing real time 3D applications. Graphic API[2] is hidden under an object oriented facade. It is available for different platforms, such as Windows and GNU/Linux[3]

Nebula is also able to control user input (mouse, keyboard and joystick) and play sounds. It is in charge of creating the main window, and it helps in 3D collision detection, and network control. Besides, it contains some helpful classes to create a *Nebula application*. Programmers can extend this application to create their games.

One of the main Nebula advantages is the duality of its API: Nebula framework can be extended using C++ and *scripts*. In that sense, all the Nebula state can be accessed and modified using languages such as Tcl, Python or LUA.

All the Nebula C++ classes are arranged in a well–designed object oriented architecture. For instance, in Nebula 1, the `nMeshNode` class stores an object geometry, and `nShaderNode` keeps information about the shaded process in a 3D object, such as colour or texture.

Every C++ created object can be accessed from the *scripts* and vice versa. Nebula kernel manages the so called *Named Object Hierarchy* (NOH), a kind of *virtual file system* where the "directories" are in fact *Nebula objects* without any relation to a real directory. NOH is used in order to access to any existing object using its name. Some objects contains other ones as attributes. In that case, the "directory" representing the container object will have "subdirectories", one per contained object. For example the object `/usr/scene/table` representing a table would be formed by its 3D object, and by `/usr/scene/table/lamp` and `/usr/scene/table/glass`, that are objects whose coordinate systems depend on the table one.

When a new Nebula object is created using C++ code, it is provided with a name in the NOH. Using the C++ variable, the object can be, obviously, manipulated; the interesting point here is that its position in the NOH can be used to recover it later (when the variable has gone out of scope) or to access it from any *script*. In that sense, the Nebula NOH builds the bridge between the C++ code and the *script* code.

One of the most interesting features that converts Nebula in a good tool to support the Computer Graphics lectures is its *console* . When an application is running, pressing down the *scape* key shows the *console* on the foreground. It works in a similar way to the command line interface of Unix and MS-DOS. The "working directory" is related to the NOH, so, for example, `dir` lists the content ("subdirectories") of the current object, and `sel` is used in a similar way to the Unix `cd` command. As said before, each "directory" links a C++ object; when in a directory, we can send messages to the related object using the console[4].

Available messages will depend on the class the object is instance of (we can list all of them using the `.getcmds` command in the "directory"). For example, if the current directory represents a Nebula 1 `n3dnode` object (which contains a reference system), we can type `.txyz 10 10 10` to translate its reference system to that location. Commands

[2]DirectX and, in Nebula 1, also OpenGL

[3]Nebula 2, however, is not able to render in Linux, besides the complete system compiles on it.

[4]In that sense, the current directory is in some way similar to the `this` C++ pointer

are immediately executed and their repercutions are shown in real time. Usually, almost every C++ method is available using one of the commands in the console.

All the commands written down in the console are, in fact, staments in the current *script language*. So we can also use commands in that language in the console; for example we are used to programming using Tcl, so we can write `puts "Hello World"` to do an "echo" or use variables with `set`.

The Nebula Release provides a couple of programs to test the *framework*. One of them is too basic and it doesn't even open a graphical window. The second one is more interesting. An empty 3D scene is shown. The user can add objects using the Nebula console and the NOH or, better, load one of the several examples included. *All of them* have been written using Tcl, showing the big features available without needing known C++.

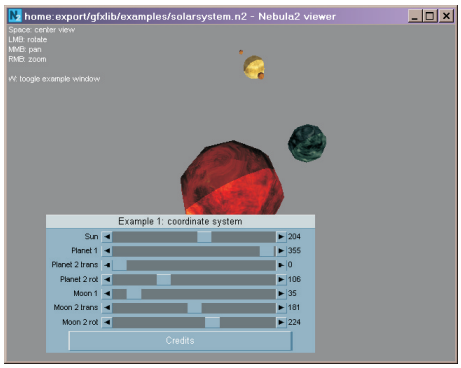

Figure 1: Scene hierarchy application (Nebula 2 version)

4. NEBULA 1 VERSUS NEBULA 2

As have been told in Section 3, there are two different versions of Nebula available. The final release of the first one is dated on January, 2004. The second version is still under development, though users are able to download it via anonymous CVS or binary builds.

Though Nebula 2 shares the design philosophy with Nebula 1, it is a more advanced framework. Its graphic subsystem has been completely rewritten to conform DirectX 9 specification. The main changes are related to *shaders* and programmable pipelines, and it support the High Level Shader Language (HLSL). It also has an animation and character subsystem, an important lack in previous version. It incorporates a GUI subsystem, that allows the inclusion of buttons, sliders and other widgets in the render window, allowing programmers to easily incorporate menus and HUDs (*Head-Up Display*) in their applications.

Another important difference between both versions is the support of Tcl. In order to limit the resources needed for the script language, and make the engine suitable for console environments, Nebula 2 uses an special version of Tcl (*MicroTcl*).

Though this decision is really appropriate in the game area, is not so convenient in our context. Using the complete set of Tcl as script language has an extra advantage: the Tk library. In order to made our interactive pedagogical applications, we definitively need widgets to manipulate some aspects of the render (for example colors, or weights in interpolations). If the lecturer has enough knowledge about Tcl/Tk, she can easily create extra windows with widgets to control those parameters.

This does not mean that Nebula 2 can not be used. Tk can be replaced by the new GUI subsystem. The final appearance is better than that using Tk, because the widgets appears integrated in the render window (see Figure 1), but its creation is not standard. Using Nebula 2 forces the teacher to first understand the GUI subsystem. If the lecturer has in mind the development of a lot of applications, that effort will be worthwhile; otherwise she should consider using Nebula 1.

5. EXAMPLE OF APPLICATIONS USING NEBULA

Just to show some examples of applications developed with Nebula, in this Section we present three small programs. All of them have been developed using both versions of Nebula, and can be used for teaching purposes. They also can be used as an inspiration or help to develop other little demos to show specific aspects of the Computer Graphics concepts.

To develop these examples, we have used a subset of the scripts, meshes and textures that are part of the demonstrations released with the framework. In that sense, we have taken advantage of the scenes and geometry of Nebula examples, modifying them to our own gain.

The developed applications make an intensive use of scripts. In fact, the entire applications are build through Tcl scripts, that load the resources (scene), create the widgets and associate their events to specific parameters of the render. The widgets are created using Tk in the Nebula 1 versions and the GUI subsystem in the Nebula 2 ones.

With these three examples we want to show the number of lines of code needed to build pedagogical applications. Next three subsections describe them. After that, in Section 5.4 we detail the amount of code required to build the Nebula 1 version. As we will see, creating the widget window in Tk became the most consuming task as regard the number of lines of code.

5.1 Scene hierarchy

We started the motivation of our paper talking about how to explain the hierarchy of coordinate systems. We have develop a small application based on an example distributed with Nebula 1 to exemplify the concept. The program shows a solar system where the coordinate system of each satellite depends on the reference system of its planet, and this one depends on the star's one.

Once the scene is created via scripts, Nebula allows modify during execution the properties of each reference system. For example, the user can rotate a coordinate system through the Nebula console using the `.ry <angle>` command. This change will affect the subordinate ones. Just for a quicker and easier modification of these parameters, we added the widgets that allow the rotation of some reference systems (see Figure 1).

Though the widget allows just translate and rotate some of the objects in the scene, the teacher is able to change other ones using the Nebula console if some students ask for that. For example, this application is not meant to show the

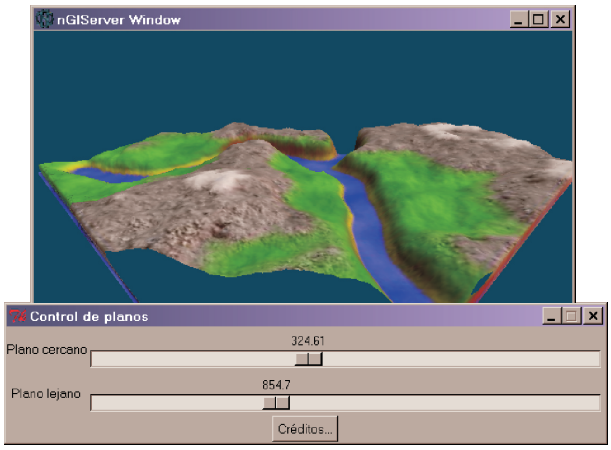

Figure 2: Near and far plane application (Nebula 1 version)

scale concept of the system coordinates. However, the lecturer can react to a student question about that concept by changing the value of the scale parameter using the console.

5.2 Near and far plane

Concepts in the previous examples are still possible to explain using movements of the arm, or the example of a solar system. But the existence of the near and far plane in the render pipeline is rather difficult to justify and even more complicated to illustrate.

Once the frustum concept is explained it is quite useful to show the execution of an application where the position of those planes can be changed.

Nebula allows the variation of the parameter via scripts. Our second application has been developed using also Tcl. It shows a scene with enough size to show the effect of changing the distance between the planes and the camera. Two sliders allows the user to change both distances even making the far plane stands nearest to the camera than the near plane. Figure 2 shows a snapshot of the Nebula 1 version.

5.3 Zoom and camera movements

When teaching camera concepts, frustum and projection matrix, teacher has to explain the meaning of the angles between left and right plane and between top and down plane. Usually students hardly understand that the value of these angles are related to the zoom of a real-life camera. During a lecture, one of the students asked us about the difference between zoom (or decrease the angle between planes) and move the camera towards the object. Though that question is related to real camera concepts more than computer graphics, we answer the question changing the angle using the console and moving the camera using the mouse.

However, we considered that the question was interesting by itself and we developed another Nebula application that allows the user to change the zoom and the camera distance to the terrain with two sliders.

5.4 Amount of code

Just to illustrate how difficult is to build these applications, we will inspect how they are built and show statistics about the lines of code required. We have done that using the Nebula 1 version of the three examples, because the construction of the widgets need more lines of code. So, these results can be considered as an upper limit of the amount of text written. Readers interested in the Nebula 2 version are invited to visit the web page.

Description	Hierarchy	Planes	Zoom
Specific of that application / Nebula inspired	189 (96)	28 (0)	36 (0)
Reused in one more application	0	80	80
Reused in two applications	150	150	150
Total lines	339	258	266

Table 1: Lines of code of the examples

All these programs are build using an application provided with Nebula (`nsh`). It starts the Nebula environment and launch a Tcl script specified by parameter. We have a script for each application. They create the render window, scene and Tk window with the widgets to modify the taught parameters.

The construction of the render window and other stuff is common to all applications, and the major part of that code is actually distributed with Nebula so we just had to adapt it with minor changes. The construction of the scenes (solar system in the first one and terrain in the other ones) are also part of that distribution, so we just created the code to place the camera. We implemented an "About..." window using Tk that also reused in every application.

So, the only pieces of code that we had to write from scratch and we could not reuse in other examples were the construction of the Tk windows that contain the widgets to control the parameters, and the event handlers. However, that code constitutes a small percentage of the total lines of each example. First row of Table 1 reveals that only a small amount of code is specific of each example (in brackets the number of those lines inspired in Nebula demos). Most of the code, however, is reused in one or both other examples (second and third row respectively).

6. OTHER RESOURCES

Previous sections have shown the use of Nebula to develop demo applications for showing Computer Graphics concepts. Although using this *framework* and its *scripting languages* can save time, for some specific aspects of the Computer Graphics curricula there are available pre-made applications that can be used directly with no effort.

For instance, Nate Robins developed a set of applications useful to explain, among others, projection, fog, material and texture concepts [5]. These programs usually split the main window in three *frames*; the first one shows the rendered image, the second one presents the same scene from an external point of view that let the user see the camera and its frustrum; finally, the third frame contains widgets where manipulate the important parameters for the concrete example. Figure 3 shows a snapshot of the application used to support the explanation of directional lights allowing to change the light and camera positions.

These tutorials have a similar goal to our applications, but they are meant to teach the OpenGL API (instead of just

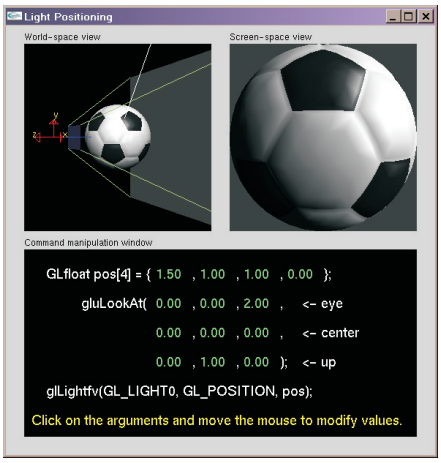

Figure 3: Snapshot of Nate Robins light tutorial

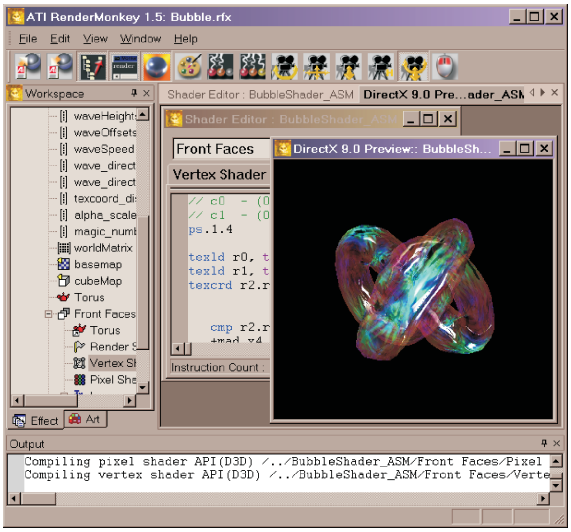

Figure 4: RenderMonkey™ snapshot

general concepts). In that sense the application explicitly shows the related OpenGL *functions* such as `glRotate` or `glLight`.

Both Nate Robins and our Nebula applications have being developed with the pedagogical goal in mind. There are some other general programs useful for artists that can also be used in lectures. Modelling tools such as 3DStudio Max® (from Autodesk®) is sometimes used. More recently, in order to teach the programmable shaders technology, two new applications shine: RenderMonkey™ from ATI (see Figure 4) and FX Composer from nVidia. The election between them usually depends on the teacher preferences more than in functionality; nevertheless RenderMonkey™ has an appreciated feature: it can simulate *pixel shaders* in computers where the hardware doesn't provide them.

Finally, most Computer Graphics students are videogames fanatics. Using stunning recent games has an important motivational impact that, on the other hand, arises their critical analysis when they play other games. Videogames can be used to explain how the *sprites* and billboards work, study the different kinds of camera (fixed, tracking camera, etc.), the selection of the mesh depending of the level of detail, or light maps.

Unfortunately, using games is not always an easy decision. Games are hardware eater applications, so good computers are needed. Another problem is that usually a game needs some time to initialise itself, and has menues. Since the teacher starts the game until she reaches the point wants to show, quite time could be spent. Besides, games cannot be legally provided to the students in order for them to make their own experiments. An intermediate alternative would be showing some screenshot in slides. The interactivity will be lost but, at least, the motivational aspects will be maintain because the students will feel nearer the theoretical concepts.

7. CONCLUSIONS

This paper defends the use of interactive applications to show Computer Graphics concepts during the lectures. Altough they are accepted to be a good pedagogical option, unfortunately their time consuming development makes them quite scarce.

We propose the use of a *framework* meant to develop 3D interactive games called Nebula as a starting point. Its more important advantages are the possibility of controlling almost all the engine using a *script* language, and the availability of a *console* that let the user introduce commands using that language in *immediate mode*. The Tk accessibility in Nebula 1, or the new GUI support in Nebula 2 are other appreciated advantages because they allow to show widgets that can be connected with render parameters which, when changed in real time, modify the 3D scene inmediately. All these features make Nebula a good option to fast prototyping and deployment of small pedagogical applications for the Computer Graphics lectures. To demonstrate this opinion, we have shown three of such applications.

8. REFERENCES

[1] *Effective Interactive AI Resources Workshop*, 2001.

[2] M. Goldweber, R. Davoli, and M. Morsiani. The Kaya OS project and the μMPS hardware emulator. In *ITiCSE '05: Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education*, pages 49–53, New York, NY, USA, 2005. ACM Press.

[3] A. Moreno, N. Myller, E. Sutinen, and M. Ben-Ari. Visualizing programs with jeliot 3. In *AVI '04: Proceedings of the working conference on Advanced visual interfaces*, pages 373–376, New York, NY, USA, 2004. ACM Press.

[4] Radon Labs GmbH Game Development. Nebula device, 2004. Available at `http://www.radonlabs.de/`.

[5] N. Robins. Opengl tutors. Available at `http://www.xmission.com/~nate/tutors.html`.