GRAPHICS DEMO IN 128KB

Martin Záleta

Bachelor Degree Programme (4), FIT BUT

E-mail: xzalet00@stud.fit.vutbr.cz

Supervised by: Juraj Vanek

E-mail: ivanek@fit.vutbr.cz

Abstract: This paper describes the key elements in the process of making simple 3D graphic animations (demos) with the main focus on minimalizing the amount of external resources such as dynamic libraries, textures and models. The resulting application consists of a WinAPI window filled with a fully operational OpenGL scene, that uses OpenGL Shading Language for the control over lightning and textures.

Keywords: demo, WinAPI, OpenGL, GLSL

1. INTRODUCTION

This work deals with peculiarities of creating a spatially constrained standalone graphics demo, the decisions that have to be faced in the making of 3D scenes and their effect on the resulting feel on the animation as whole. This raises series of issues to be solved with the use of educational books.[1]

The demo in its final stages has to stay under 128kB, have its length above 3 minutes and be accompanied with an appropriate music score. Other than that, the main aim of this undertaking remains to demonstrate and underline the overall straightforwardness and simplicity in the way of making dynamic animations in the OpenGL environment.

2. DESIGN AND IMPLEMENTATION

To make a demo in the OpenGL environment from scratch requires basic establishment of the programming approaches, which are to be preferred over others with similar or identical purpose. This includes the area of object modelling, lightning models, texture and font usage, particle systems etc. By putting all of these elements together, I am creating multiple animation scenes to construct the final demo application while using Windows API as a baseline of my implementation to avoid usage of any dynamic libraries, which aren't directly available in standard clean installations of Windows XP.

2.1. LIGHTING MODEL

One of the most important parts of every 3D scene is its lightning model, to the creation of which there are countless possibilities. This raises an open suggestion on whether or not to use OpenGL Shading Language (GLSL), which is an extension of most contemporary graphic cards as means of more direct and better control over the rendering pipeline. That makes a huge advantage in design of demos and in my implementation I chose to opt for it, which on one hand made the range of programming possibilities even wider, but restricted the number of graphic cards on which the demo will run.

1 In my case in addition to these online tutorials: Molofee, J.: *NeHe Productions.* URL: http://nehe.gamedev.net/ (march 2011). and Fernandes A. R.: *OpenGL @ Lighthouse 3D – A Resource for Programmers.* URL: http://www.lighthouse3d.com/opengl/glsl/ (march 2011).

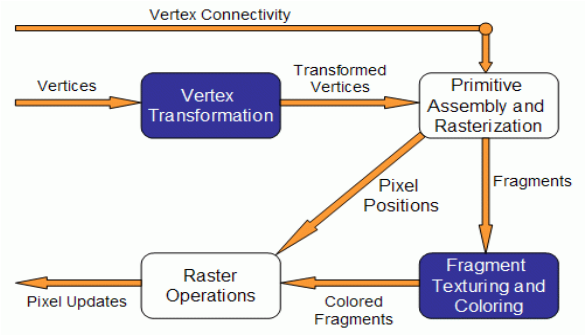

Fig.1: Rendering pipeline; blue boxes can be directly controlled with GLSL.

In my experience, for most of the scenes in demos, the Phong model can bring satisfactory results, and with the use of GLSL, there even is an option of using the Blinn-Phong lightning model instead, because it makes the necessary calculations a bit shorter due to the usage of half-vector, which is the vector between eye and light vectors and has values directly available in GLSL, making it a fair choice in my implementation.

2.2. OBJECTS AND TEXTURES

The use of various objects and textures is the cornerstone of almost every 3D animation, so in a demo it is essential to apply a simple yet useful way of their rendering without having to import them from external files.

As for the objects, one of the most straightforward options is the use of quadric surface objects provided by the OpenGL environment as routines to directly model some of the commonly used 2D or 3D objects such as circles, spheres or cylinders. Making their parameters easily adjustable by granting the programmer a direct control over their size and length gives an opportunity to make the objects of the animation more complex and dynamic than in other direct approaches.

Next, in order to avoid using externally defined textures, they need to be replaced with some appropriate kind of procedurally generated ones, such as those defined by fractals or noise functions. This includes the so-called Perlin noise, which is an algorithm for converting raw noise functions into more organised arrays of values, that can be used precisely for this sort of a task.

After subjecting the aforementioned approaches to a series of practical tests, I have reached a conclusion, that even in real-time circumstances they can manage to form variously shaped objects and materials in a reasonable computing time.

2.3. PARTICLE SYSTEM

To make the animation more interesting for the viewer, it was necessary for me to consider the use of particles in it and devise an appropriate particle system. After closer examination, it became obvious it isn't enough to put quads in their place because of their inability to maintain their orientation towards camera. On the other hand, points and their adjustment called point sprites are much more appropriate for this task. For their characteristics, I am using them in my animations to make them feel more dynamic and lifelike, while storing their position through time by using the translation matrix.

2.4. FLOATING TEXT

For improved visual effect, I decided to provide a component of objects formed out of fonts and spelling out a description of what exactly is happening on the scene. For this purpose, there is a function in OpenGL, which makes it possible to create display lists out of windows fonts called outline fonts. This makes their initialization and usage very simple and flexible.

3. CONCLUSION

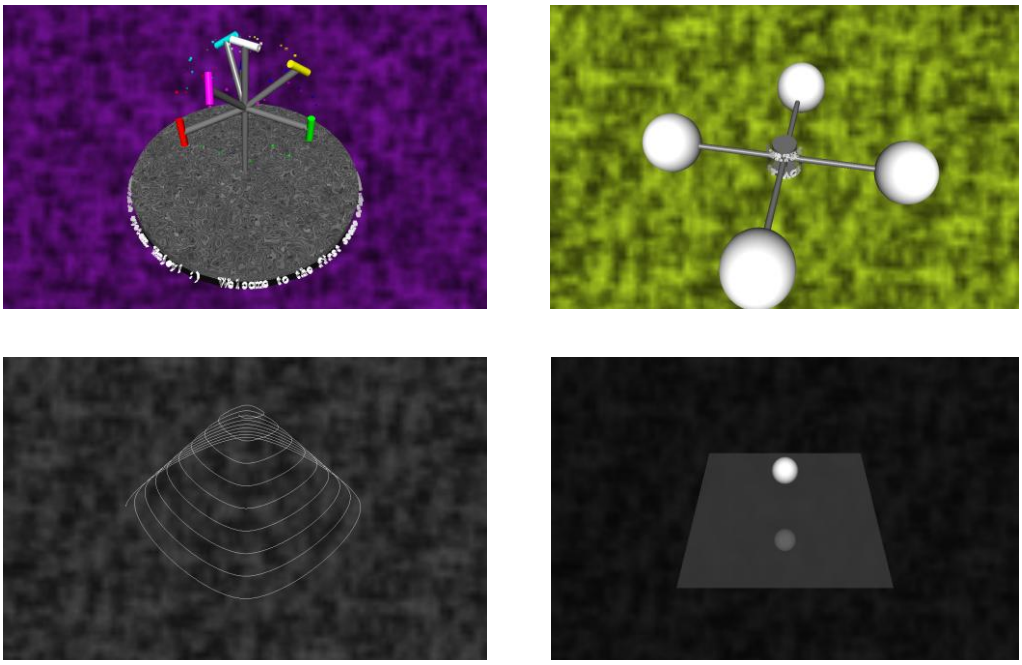

Fig.2: Project status as of march 2011.

For now, the compiled demo takes little less than 25kB of space, opening possibilities of using more complex music synthesizers in its audial part other than the simplest one, which is any standard MIDI synthesizer. But other than that, I am still planning to avoid external storage of models through header files and remain in the application of procedural approaches.

As I mentioned earlier, my implementation will in its final stages contain multiple scenes with different themes, namely vertex displacement in GLSL, use of splines in a scene (specifically Catmull-Rom splines), advanced object modelling approaches (such as Catmull-Clark subdivision) and finally, the use of stencil buffer in creation of reflective surfaces below objects behaving according to elementary physical laws. To form a 3 minute long demo out of these scenes is achieved by adding transitions in between them.

REFERENCES

[1] Schreiner, D.: *OpenGL programming guide : the official guide to learning OpenGL, versions 3.0 and 3.1*. Boston: Pearson Education, 2010.

[2] Rost, R. J.: *OpenGL shading language - 3rd ed*. Boston: Pearson Education, 2010.

[3] Randima, F.: *GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics*. Boston: Pearson Education, 2004.