Crossroads
The ACM Student Magazine

Graphics Libraries for Windows Programming

By *M. Carmen Juan Lizandra*

Introduction

Any new software project, be it a word processor or a high end graphics rendering software, presents its creator with a multitude of options. Considerations such as code reuse, portability and performance are important to every application. Often, it is impossible to achieve all these considerations, and tradeoffs are required. At times, one must chose between ready-to-use code libraries, SDKs or APIs that can speed up the development process, eliminate portability problems and ensure efficiency.

For those who prefer not to reinvent the wheel, several graphics libraries are available for 2D and 3D development. After all, it is not efficient to create functions or methods that have already been implemented in commercial libraries. This article describes two libraries for use with C and/or C++. The sidebar introduces **basic definitions and concepts**. The first section of the paper discusses the more relevant features of the two libraries. Some sample images created using these packages and Visual C++ are included.

Nugraf Developer's 3D Toolkit

Nugraf from Okino [2] is a library that allows the user to create highly realistic scenes that are at times comparable to photographs. Nugraf provides a C interface. This allows a programmer to easily use Nugraf functions which are themselves written in C. By passing the appropriate parameters it is possible to perform an operation in the hierarchical database that Nugraf uses to store scene information. This library includes functions to specify most of the scene essentials such as a scene camera, lights and options related to the rendering mode. Functions to define geometric primitives, to declare the material features and to define a texture are also provided.

Geometric primitives

Geometric primitives allow the definition of an object, however complex, using one easy function call with few parameters. They can be geometric structures that are normally combined to create objects (cubes, spheres, etc.) and might get quite frustrating to define vertex by vertex. Complex structures can be divided in several classes: straight polyhedrons, curved Polyhedrons, 3D Surfaces and polygonal primitives.

Scene definition

A scene, which later will be rendered, needs to coherently represent all the information that is available. At times several geometric structures could be repeated. Nugraf uses a hierarchical database to store scene information. This tree is structured in terms of objects and instances. Do not confuse the terminology used by Nugraf with OOP terminology, remember that the package provides a C interface (not C++).

An object in Nugraf is a named encapsulation of one or more geometries. This entity is generic and can not be directly rendered. To render an object one must create an instance of each object and add these entities to the scene. The instances inherit the geometric structure of their parent object. However, modifications (scales, rotates, translations) done to an instance will not affect the original object. Instances of an object do not belong to a scene until they are explicitly added to the hierarchical structure. The instance tree is born out of a special instance named *world*. When a scene has to be rendered, the tree is visited starting from the root node world. All accessible instances are rendered in a top down manner.

Rendering

In a Nugraf scene many cameras can be positioned, however only one can be active at a time. Several rendering methods can be used to produce very different results. One can choose from wireframe or hidden-line wireframe methods. Nugraf also provides a method of its own in which you can render anything from simple images, composed of few polygons with only a few colors, to photographs with high realism of complex scenes composed of thousands of textured polygons and shades. This method performs elimination of hidden surfaces using a hybrid algorithm (somewhere between scanline-zbuffer and area-buffer.) Finally, flat, Gouraud or Phong shading models can be applied to the scene. The lights can be of four different types: indirect light, sun, light bulb or spotlight (each one simulates: ambient light, directional light, punctual light and spotlight, respectively). All lights have a parameter, *intensity*, which expresses the power of each light.

It is possible to simulate a great number of materials using the definition of a surface. The attributes of a surface are reflectance properties, color of each component of the reflected light, textures, opacity and the reflection map. Features like the reflectance and absorption of a material are expressed in the Nugraf shading model using four components: ambient reflex, diffused reflex, spotlight reflex and environment reflex. Reflection maps can be with planes, spheres or cubes. Several general aspects related to the rendering process can easily be defined. Nugraf offers a solution to the aliasing problem. The antialiasing algorithm used is based on a technique known as over-sampling.

The resultant output is always an image. This image can be seen on the screen or stored in a bitmap file. One or several destinations of the rendered image can be specified. When an image is generated, its result is stored in all the destinations specified.

To sum up, good features of this library are:

1. *Excellent realism*. Near ray-tracing quality, with shading and normal interpolation, texture mapping, material features, different kind of lights, thrown shades, reflection calculations, etc.
2. *Minimum Temporal Cost*. The rendering method has several capabilities such as culling, clipping, storing shade information and reflection maps that accelerate the process to a great extent. Scenes that with non-optimized ray-tracing take several hours, can be generated by Nugraf within a few minutes.
3. *Low Hardware Demand* This library works very well with low-end or mid-level PCs.

On the other hand:

1. *Non Flexible* There is little control over the user interface. Nugraf works as a *black box* between the main application and the output device.
2. *Only 3D* Nugraf deals only with 3D surfaces. No 2D entities are provided.
3. All the objects in a scene have to be declared with a set of primitives included in Nugraf, and sometimes they are not very easy to use.

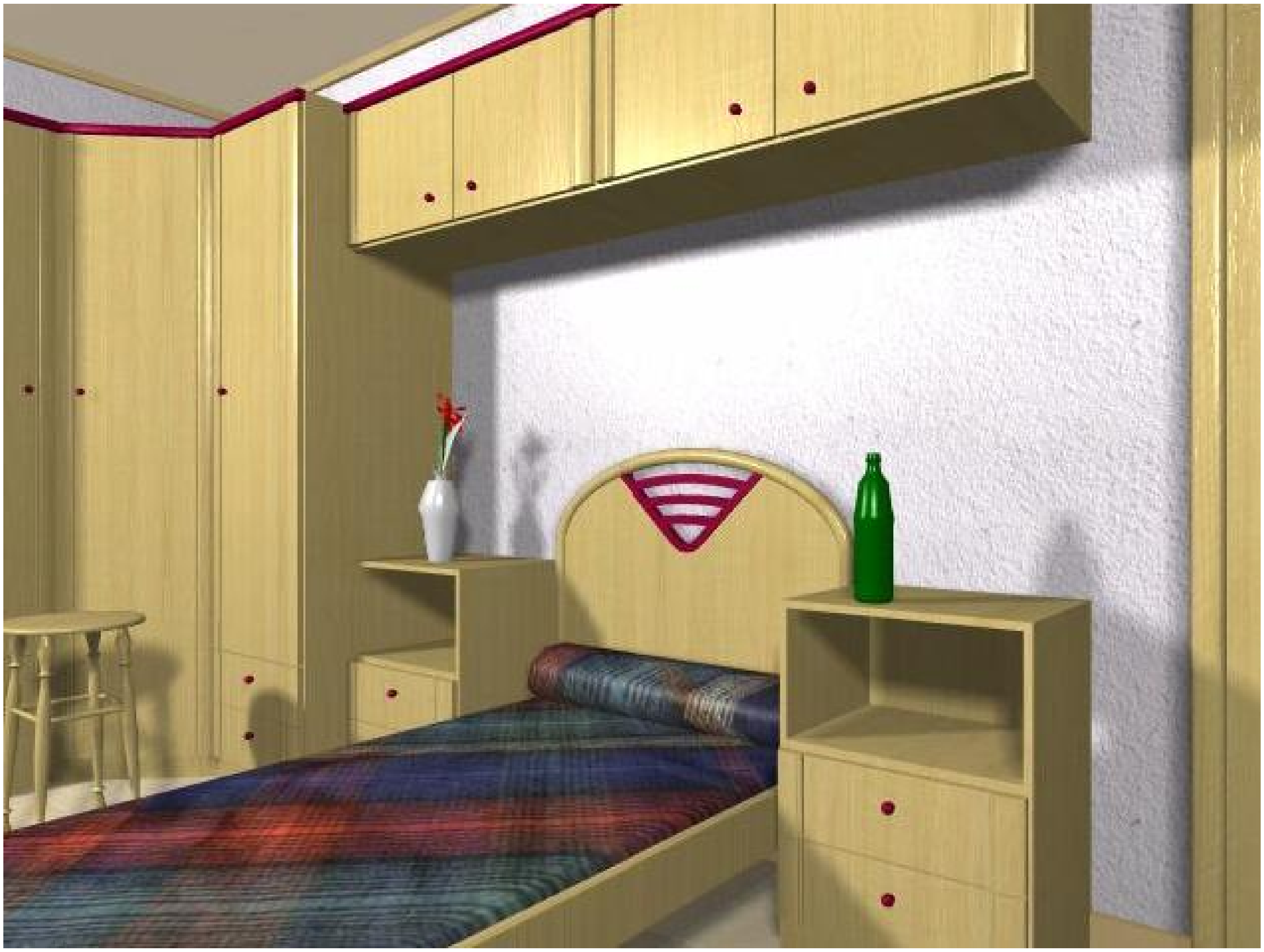
Nugraf examples

Figure 1 shows several examples generated using Nugraf. These have been extracted from an application that is used to design houses. Using this application it is very easy to visualize changed tiles or added furniture. In this application you can design the rooms in 2D, then add furniture, lights, etc. In the next step you can render the room in 3D using Gouraud or Wireframe rendering. Finally you can 'photograph' the scene from any angle. In this case, though, OpenGL has been used to achieve this 3D visualization, allowing the freedom of navigating through the room. The point of view is selected before the 3D visualization.

It can be seen that the quality of the images created using Nugraf is excellent. The disadvantage is that the rendering is not in real time. That is, you can not use it to navigate through the room. (a) and (b) show some of Nugraf features. (c)-(f) show several scenes from different houses. Examples use different types of white light sources: ambient, punctual and spotlight. All the images have thrown shades and texture mapping. A reflection map with maximum index has been applied in the sphere that appears in (b), and in the mirrors that appear in (a), (c) and (e). These reflection maps are: spherical in (b) and flat in (a), (c) and (e).

(a) (b)

(c) (d)



(e) (f)

Figure 1. Different examples created using Nugraf

OpenGL

OpenGL from Silicon Graphics [3] is a software interface used to create objects and operations for interactive, three-dimensional applications. OpenGL also provides a C interface. This library includes functions to specify most of the scene essentials such a scene camera, geometric primitives, lights, textures, etc.

It would be in order to list the more relevant capabilities for OpenGL:

- **Accumulation buffer** A buffer in which multiple rendered frames can be composited to produce a single blended image. Used for effects such as depth of field, motion blur, and full-scene anti-aliasing.
- **Alpha blending**. Provides a means to create transparent objects.

- **Automatic rescaling of vertex normals** changed by the modeling matrix.
- **BGRA pixel formats and packed pixel formats** to directly support more external file and hardware frame buffer types.
- **Color-index mode**. Color buffers store color indices rather than red, green, blue, and alpha color components.
- **Immediate mode**. Execution of OpenGL commands when they're called, rather than from a display list.
- **Display list**. A named list of OpenGL commands. The contents of a display list may be preprocessed and might therefore execute more efficiently than the same set of OpenGL commands executed in immediate mode.
- **Double buffering**. Used to provide smooth animation of objects. Each successive scene of an object in motion can be constructed in the back or "hidden" buffer and then displayed. This allows only complete images to ever be displayed on the screen.
- **Feedback**. A mode where OpenGL will return the processed geometric information (colors, pixel positions, and so on) to the application as compared to rendering them into the frame buffer.
- **Level of detail control** for mipmap textures to allow loading only a subset of levels.
- **Materials lighting and shading**. The ability to accurately compute the color of any point given the material properties for the surface.
- **Pixel operations**. Storing, transforming, mapping, zooming.
- **Polynomial evaluators**. To support non-uniform rational B-splines (NURBS).
- **Primitives**. A point, line, polygon, bitmap, or image.
- **Raster primitives**. Bitmaps and pixel rectangles.
- **RGBA mode**. Color buffers store red, green, blue, and alpha color components, rather than indices.
- **Selection and picking**. A mode in which OpenGL determines whether certain user-identified graphics primitives are rendered into a region of interest in the frame buffer.
- **Specular Highlights**. Application of specular highlights after texturing for more realistic lighting effects.
- **Stencil planes**. A buffer used to mask individual pixels in the color frame buffer.
- **Texture coordinate edge clamping** to avoid blending border and image texels during texturing.
- **Texture mapping**. The process of applying an image to a graphics primitive. This technique is used to generate realism in images.
- **Three Dimensional Texturing**. Three-dimensional texturing for supporting hardware-accelerated volume rendering.
- **Transformation**. The ability to change the rotation, size, and perspective of an object in 3D coordinate space.
- **Vertex array enhancements** to specify a subrange of the array and draw geometry from that subrange in one operation.
- **Z-buffering**. The Z-buffer is used to keep track of whether one part of an object is closer to the viewer than another.

OpenGL-related libraries

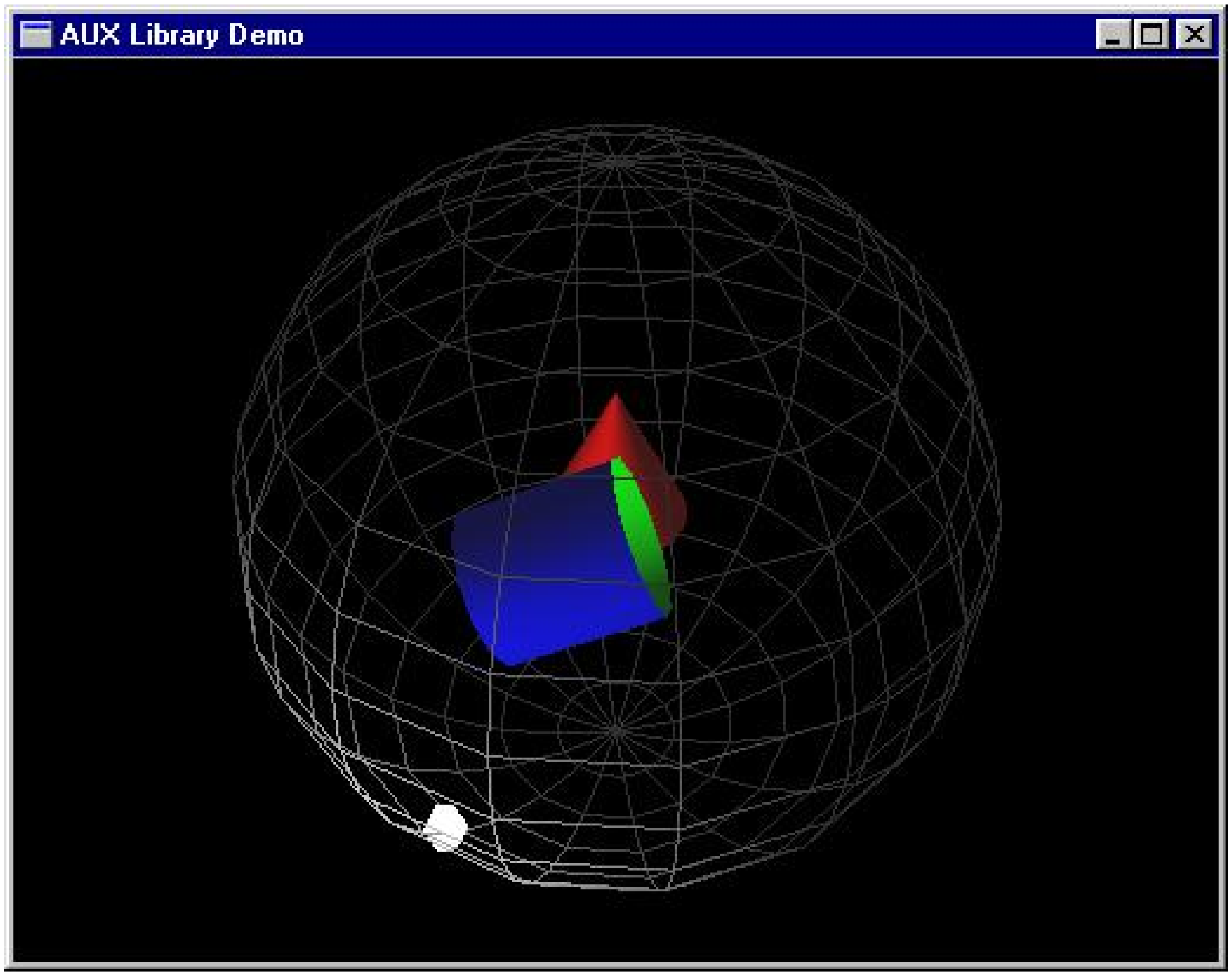
The OpenGL Utility Library (GLU) [4] contains several routines that use lower-level OpenGL commands to perform tasks such as setting up matrices for specific viewing orientations and projections, performing polygon tessellation, and rendering surfaces. The OpenGL Extension to the X Window System (GLX) provides a means of creating an OpenGL context and associating it with a draw-able window on a machine that uses the X Window System. Open Inventor [5] is an object-oriented toolkit based on OpenGL that provides objects and methods for creating interactive three-dimensional graphics applications. Open Inventor is written in C++ and it provides pre-built objects and a built-in event model for user interaction, high-level application components for creating and editing three-dimensional scenes, and the ability to print objects and exchange data in other graphics formats.

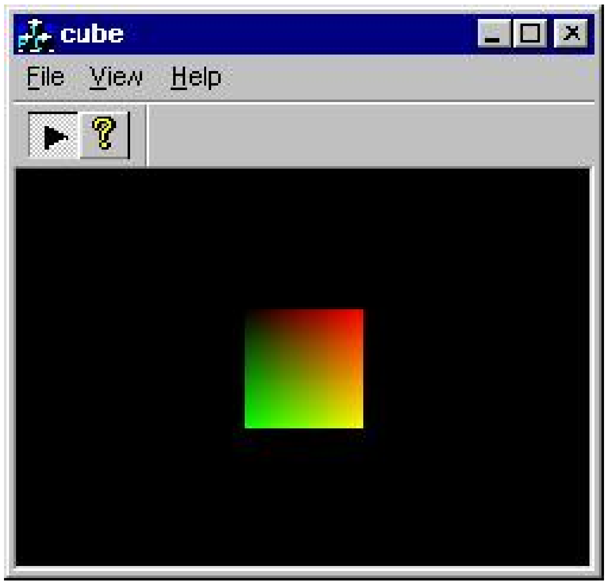
Mesa [6] is a free implementation of the OpenGL API for Linux, designed and written by Brian Paul, with contributions from many others. Its performance is competitive, and while it is not officially certified, it is an almost fully compliant OpenGL implementation conforming to the ARB specifications. You can find more information on Mesa home page.

The OpenGL Utility Toolkit (GLUT) [7] is a programming interface with ANSI C and FORTRAN bindings for writing window system independent OpenGL programs. The toolkit supports the following functionality: multiple windows for OpenGL rendering, callback driven event processing, sophisticated input devices, a "idle" routine and timers, a simple, cascading pop-up menu facility, utility routines to generate various solid and wire frame objects, support for bitmap and stroke fonts, and miscellaneous window management functions, including managing overlays.

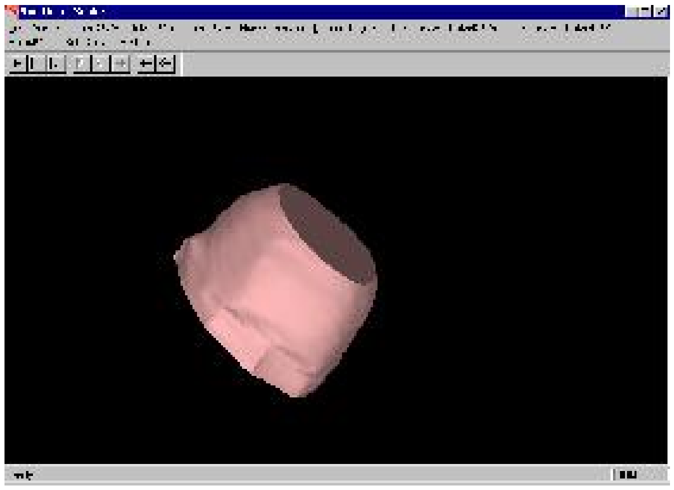
OpenGL examples

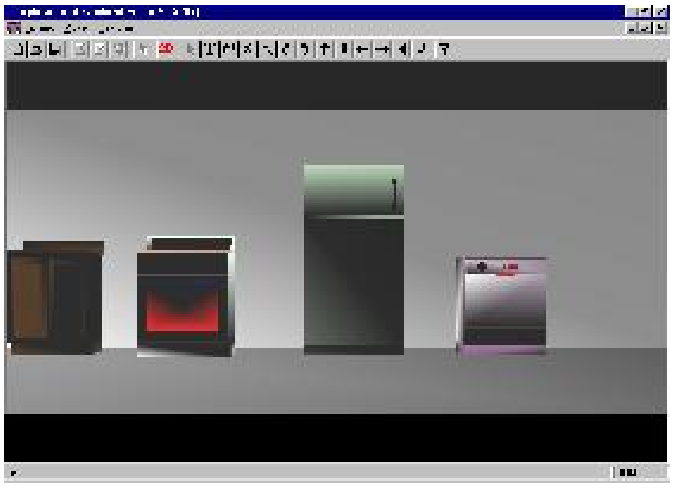
Figure 2 shows some examples. All of them perform 3D navigation., (d)-(f) in an interactive manner. (a) presents an OpenGL demo. (b) shows an OpenGL and Visual C++ example. In (c) an application that can visualize, rotate, move and zoom objects in 3D can be seen. The user can change the point of view using the mouse. Another application that can help you to design your house or add new furniture can be seen in (d)-(f). This application uses shading and textures to achieve realism. Though it does not achieve the quality of the examples showed in figure 1, it allows navigation through the room. (d) shows a scene with furniture. (e) and (f) present a textured floor.



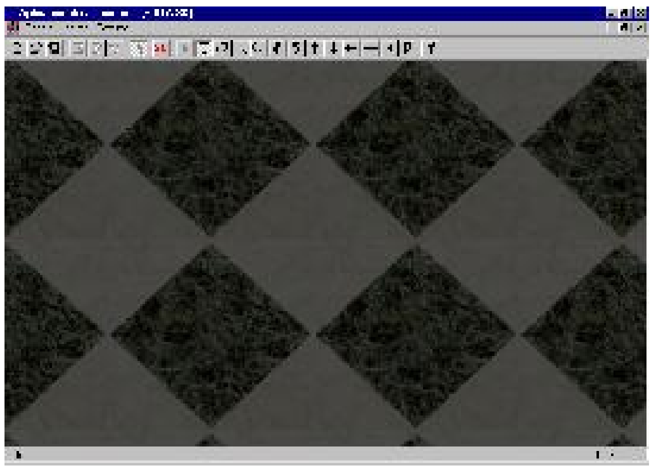


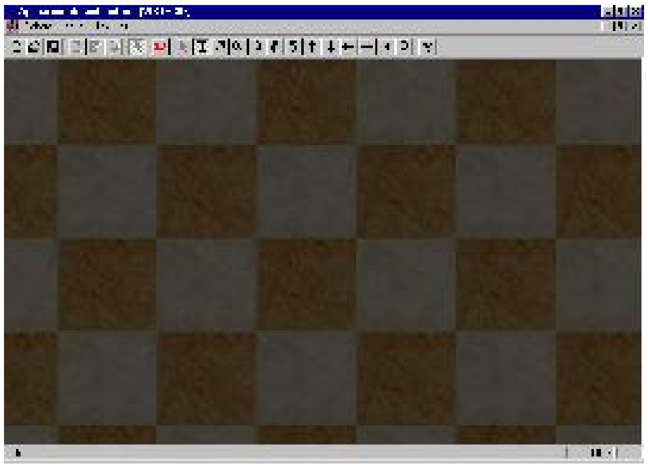
(a) (b)





(c) (d)





(e) (f)

Figure 2. OpenGL examples

Conclusions

There are several libraries that can help save a graphics programmer a lot of work. They help improve the quality of the finished applications and reduce the time needed to create them. From our discussion, it emerges that Nugraf is an excellent library if you want to create applications that take high realism 'photographs' of scenes. OpenGL can help you to create 2D and 3D interactive applications. In some cases the scenes achieved could be quite real. For interactive navigation in a 3D textured scene, you could possibly need OpenGL accelerator hardware.

References

1
Foley, van Dam, Feiner, Hughes. Computer Graphics: Principle and Practice. Ed. Addison Wesley, 1987.

2
Nugraf, Okino. **http://www.okino.com**, 1999.

3
OpenGL, **http://www.opengl.org**, 1999.

4
GLU, **http://www.opengl.org/Documentation/GLX.html?glu#first_hit**, 1999.

5
Open Inventor, **http://sal.jyu.fi/F/3/TGS_OPENINVENTOR.html**, 1999.

6
Mesa, **http://www.ssec.wisc.edu/~brianp/Mesa.html**, 1999.

7
GLUT, **http://reality.sgi.com/opengl/spec3/spec3.html**, 1999.