One approach to C++ look at DirectDraw and Direct3D

Victor A. Debelov, Yuri A. Tkachov
Institute of Computational Mathematics and Mathematical Geophysics
of SB RAS, Novosibirsk, Russia

Abstract

Microsoft DirectX [1] becomes more and more popular among users of Windows platforms. Our interests concern two main graphical parts, namely: DirectDraw (2D dynamic graphics) and Direct3D (only Retained Mode is considered). They are developed during several versions of DirectX almost continuously and their functionality grows version by version. Although DirectDraw and Direct3D are considered as tools for C/C++ programmers, they are really instead collections of COM-objects, or interfaces. The relationships between these interfaces are not clearly visible, even from a second look. For example, Direct3D's Retained Mode layer contains as many as 33 interfaces, and correspondingly thousand of functions as well as many other parameters that control the rendering process.

The system developed by authors and suggested in the given report, is based on the OO approach. It consists of a collection of basic classes (objects), contrary to COM-interfaces. These objects wrap and expose properly gathered and organized functions of DirectX thus hiding extra possibilities of DirectDraw and Direct3D. We implemented a kernel of the system in Microsoft Visual C++ environment. Construction of the system is transparent and can be considered as guidelines for further improvements or refinements.

A collection of basic classes for developing 2D applications includes two main classes. **MainObject** represents the «world» or the main frame where all other objects operate and cooperate with each other. The second main class is a **Sprite** that is an abstraction of an object that may have its' own size, view, position in the «world» and a couple of other attributes. Several helper classes for managing Sprites and MainObject are also developed. Analogous construction was done for 3D applications - **D3DMainObject** is a scene and several classes help to represent: geometry and transformations (second main class **D3DFrame**), lights, animations, etc.

In our opinion such a system should have a great impact to those who begin to study how to create 3D graphical dynamic applications on the base of DirectX.

Keywords: *DirectDraw, Direct3D, OO computer graphics system, MSDeveloper Studio.*

1. INTRODUCTION

We can recognize a lot of levels of computer graphics functionality in modern days. For example, let us consider the situation of Windows-95/NT in brief, assuming the following levels of complexity.

Level of driver capability. An application programmer has at his disposal the features that are the fastest and closest match to a particular hardware architecture. It means that (depending on his skills) he can reach an exceptionally high level performance and productivity of his graphical application. On the downside, it can require a major commitment of time and effort, that in some cases produces less than the desired or expected results.

Windows GDI (graphical device interface). As a general rule, GDI is somewhat easier to use. At least the problem of program portability is eliminated, e.g., in Windows environments. However, the set of available graphic primitives is still insufficient, as are a set of available set of transformations.

DirectDraw [2] of Microsoft DirectX. Another incarnation of GDI. It suggests very poor set of operations and primitives optimized for achievement of maximum drawing speed. In fact, there exists only one primitive – so called Drawing Surface and a set of operations to manipulate a number of such surfaces. The set of operations includes direct access (on pixel level) to a surface and bit block transfer operations (BitBlt) between two surfaces.

Let us consider the case of 3D graphics and focus on a layer structure of 3D functionality.

GDI or DirectDraw. All particulars of the 3D world are left to application programmers.

OpenGL [3] or *Direct3D's Immediate Mode* [4]. A user is still devoted to annoying procedures to convert the 3D world of the application to a restricted set of instructions. Nevertheless, he works already in a 3D environment (coordinating systems, camera, transformations, real light sources, etc.).

Direct3D's Retained Mode – D3DRM [5]. A user works in the real 3D world. The mechanism of frames allows him to construct very complex scenes quickly and easily. At the same time, he loses the productivity of his application but

... gains more time for its' development. Notice that the power of the primitives' set and a collection of transformations are increased, while we constrict our task (locally), and agree to lose some flexibility in an application.

In our opinion, the fastest preparation of a 3D scene, especially of its layout, is a critical point of most applications. Especially when a creator assigns proper places for all actors (geometric objects, lights, shadows, etc.). Very often, the design process begins from creating a prototype application, which is evaluated and then converted to the final application via revolutionary or evolutionary process [6].

One known attempt to increase the level of functionality of 3D graphics in the Windows'95 environment was described in [7]. The object-oriented library was created as a cover of COM-interfaces of DirectDraw and Direct3D.

We also developed the C++ shell [8, 9] for functions of D3DRM COM-objects. It is too difficult to choose one approach over the other, as we only list our several main design requests herein:

- to minimize the number of basic C++ classes;
- to hide particulars of concrete interfaces, especially "driver-dependent" types;
- to minimize the efforts necessary to learn and become familiar with the system, while simultaneously providing the significant functionality;
- to make the system architecture open, i.e. easy to extend and improve.

Thus we suggest the C++ 3D graphics tool which is situated just above D3DRM - less flexibility and functionality but less time to comprehend. So, why did we choose DirectDraw/Direct3D instead of OpenGL with its advanced development tools? Well, one of the features of the basic Direct3D object (frame) had convinced us – the frame's intrinsic velocity and rotational speed and automatic reaction to scene heartbeats.

Although we focussed mainly on 3D functionality and developed a tool for 3D graphics programming, we nevertheless needed to comb 2D graphics (DirectDraw) in the same manner. Below, the exposition of the system will be presented in the opposite order. The construction of the 2D layer is repeated during the construction of 3D case scenario.

2. 2D DYNAMIC GRAPHICS

2.1 Brief Overview of DirectDraw

DirectDraw is a set of COM-interfaces. In order to use it in your program you have to get pointer to the main (or primary) IDirectDraw interface by one means or another. Having this pointer you are able to gain access to other DirectDraw interfaces like IDirectDrawSurface or IDirectDrawPalette. A brief description of the role of the interfaces is as follows.

Interface DirectDraw is an abstraction (or representation) of a graphic hardware of the computer as a whole. The methods of the interface allow changes to graphic modes (resolution, color depth and even display refresh rate) and to access other interfaces that represent specific parts of a graphic hardware.

IDirectDrawSurface corresponds to the graphic memory. To be more precise, each pointer to this interface represents some part of the graphic memory. It allows direct access to the video buffer and several another methods of video memory manipulation. There is one main (or primary) surface among all other surfaces. It represents the current visible state of the application window (or a whole screen if the application is running in full screen mode). In other words, the contents of this primary surface defines contents of the application window. Therefore, there is only one way to change contents of the window - you have to change contents of the primary surface. The most exciting feature of the surfaces is the ability to perform bitblt operations between different surfaces with maximum possible efficiency.

It is important to understand that the application should use at least two surfaces. The primary surface holds the current state of the applications window (current frame). The second (or back) surface is used for preparation of the next frame. If the state of the primary surface has to be changed, the process of changing must be hidden from the user. Any application must prepare the next frame in the back surface and then blit the whole buffer or relevant part of it into the primary surface.

Actually any application will use a lot of surfaces. Besides the primary surface and the back surface, it is very convenient to use surfaces (usually of small sizes) for storing sprites. Each sprite represents some object that can have its' own size and position on the screen. When a position or another visual attribute of the sprite has changed, it is necessary to refresh the back surface by blitting of the sprite surface to the back surface and then to refresh the primary surface by blitting of the back surface to the primary surface.

The interface IDirectDrawPalette represents the color lookuptable. It allows direct changing of hardware palettes in palletized modes (4, 16 or 256 colors).

The last but not least interface is IDirectDrawClipper. It does not represent any part of a graphic hardware, but its role is very important in establishing the proper communication with windows of another currently running application in the windowed (not full screen) mode. A clipper object holds one or more clip lists. A clip list is composed of one bounding rectangle, or a list of several bounding rectangles that describe an area or areas of a

surface to which you are allowed to blit. So, if the application window is partially overlapped by another window (the usual situation in the windowed environment) then the clipper object looks after all windows and all blit operations in order to prevent drawing in the alien window. You have almost nothing to do with this object. You have just to create the object and connect your application window with it. It will automatically do the rest for you.

The model of DirectDraw functioning is quite simple. To develop a fully functional DirectDraw application it is necessary to:

1. Create a window just like in any other windowed application. This window will receive user input in the form of keyboard and mouse messages.
2. Create a DirectDraw main object and associate it with that window. The association with the window is absolutely necessary because it is the window that defines the position and sizes of the DirectDraw output on the screen.
3. Create a clipper and assign the window to it.
4. Create the primary surface and the back surface. The sizes of these surfaces must be equal to the sizes of the window, associated with the DirectDraw object.
5. Create a couple of additional surfaces for the sprites.
6. Make the main application loop that will change the state of the primary surface according to the user input, the states of the sprites, and internal logic of the application.

2.2 Basic DD Classes

The set of developed classes reflects the set of generic DirectDraw objects and logical interrelations among them.

The class CDDMainWnd provides the basic functionality of the application window with DirectDraw. The class holds the object of the class CDDMainObj and provides a proper cooperation between the window and this object. Specifically, it looks after window sizes and calls relevant methods of the CDDMainObj to ensure proper sizes of the primary surface which must be equal to window sizes. This class also receives user input messages from the mouse and keyboard.

The CDDMainObject keeps tracks of all subordinate objects like the primary and back surfaces, sprites, palettes, as well as the clipper, and organizes proper relationships between all of them. In particular, it refreshes the primary surface as soon as the state of some sprite or sprites has changed. It is also responsible for cleaning up of all objects when the main object is destroyed.

The class CDDSprite contains one or more surfaces which hold visual representations of the sprite. Surfaces of the sprite may be shared with other sprites. So you can instantiate any number of sprites that has the identical visual representation. It is useful when the application has many objects of the same type placed in different positions. This allows saving the video memory which is a costly and limited resource.

3. C++ ENVELOPE OF DIRECT3D'S RETAINED MODE

Screen snapshot, dynamic 3D application, using different classes of geometric forms, shadows, textures, and materials:

3.1 Brief Overview of D3DRM

First of all, let us consider what categories of information should a user know in order to prepare his D3DRM application (the list taken from [5]):

- Functions and Interfaces
  - Callback Functions
  - Interfaces
  - Nonmember Functions
- Data Types
  - Constants
  - Enumerated Types
  - Structures
- Other Topics
  - Return Values
  - Further Reading

This information is superimposed with the process of evolution of DirectX D3DRM SDK. If interfaces of SDK version 3.0 and SDK version 5.0 are allowed to be used jointly, then SDK version 6.0 rejects such a possibility to mix different versions of similar interfaces. The diagram beneath shows the interface hierarchy and the fact that the new version 6.0 of D3DRM supports only new interfaces

and renders old versions as obsolete or replaced (marked as **RP**). The next but last obstacles, which a newcomer encounters during his study of D3DRM, are different collections of examples supplied with SDK [10] or examples found on the Internet as in [11]. Note: Some of the examples include applications that use outdated interfaces.

The rendering process is controlled via many parameters, which are set using different interfaces. The relationship between these settings is not as obvious as one would expect or hope for.

We would like to point out that the design of Direct3D contains several constraints that are difficult to understand. For example: "Direct3D allows an application to have several viewports, but each light may be attached to only one of them".

Conversely, we suggest the approach to create a user- (or application-) oriented C++ shell for D3DRM but the complete system. This approach is supported by the implemented set of basic classes and capabilities to extend and improve them in any desired direction.

Direct3D Retained Mode Interface Hierarchy

```
IUnknown
|
+--IDirect3DRM (RP)
+--IDirect3DRM2 (RP)
+--IDirect3DRM3
|
+--IDirect3DRMArray
|  +--IDirect3DRMAnimationArray
|  +--IDirect3DRMDeviceArray
|  +--IDirect3DRMFaceArray
|  +--IDirect3DRMFrameArray
|  +--IDirect3DRMLightArray
|  +--IDirect3DRMObjectArray
|  +--IDirect3DRMPicked2Array
|  +--IDirect3DRMPickedArray
|  +--IDirect3DRMViewportArray
|  +--IDirect3DRMVisualArray
|
+--IDirect3DRMObject
|  +--IDirect3DRMAnimation (RP)
|  +--IDirect3DRMAnimation2
|  +--IDirect3DRMAnimationSet (RP)
|  +--IDirect3DRMAnimationSet2
|  |
|  +--IDirect3DRMDevice (RP)
|  |  +--IDirect3DRMDevice2 (RP)
|  |
|  +--IDirect3DRMDevice3
|  +--IDirect3DRMFace2
|  +--Iirect3DRMInterpolator
|  +--IDirect3DRMLight
|  +--IDirect3DRMMaterial (RP)
|  +--IDirect3DRMMaterial2
|  |
|  +--IDirect3DRMViewport (RP)
|  +--IDirect3DRMViewport2
|  |
|  +--IDirect3DRMVisual
|  |  +-IDirect3DRMFrame (RP)
|  |  |  +-IDirect3DRMFrame2 (RP)
|  |  +-IDirect3DRMFrame3
|  |  |
|  |  +-IDirect3DRMMesh
|  |  +-IDirect3DRMMeshBuilder (RP)
|  |  |  +-IDirect3DRMMeshBuilder2 (RP)
|  |  +--IDirect3DRMMeshBuilder3
|  |  |
|  |  +--IDirect3DRMProgressiveMesh
|  |  |
|  |  +--IDirect3DRMShadow (RP)
|  |  +--IDirect3DRMShadow2
|  |  |
|  |  +--IDirect3DRMTexture (RP)
|  |  |  +--IDirect3DRMTexture2 (RP)
|  |  +--IDirect3DRMTexture3
|  |  |
|  |  +--IDirect3DRMUserVisual
|  |
|  +--IDirect3DRMWinDevice
|  |
|  +--IDirect3DRMWrap
|
+--IDirect3DRMObject2
```

3.2 Main Notions

The main object of D3DRM is a scene – or 3D world. It consists of a set of geometric objects, a set of light sources, and a set of decals.

Scene as a whole is characterized by the following features:

1. Left-handed coordinate system
2. Camera and its parameters
3. Viewport, background color, background image
4. Ambient light, its intensity and color
5. Fog, by default is absent
6. Shading mode: flat or Gouraud
7. Fill mode: points, wireframe, solid
8. Tick – heartbeat of a scene (enabled, disabled)

To achieve this we need to apply several methods of different COM-interfaces of D3DRM, in order to implement such a complex object.

Geometric objects. Only one-sided surfaces are recognized by D3DRM. All surfaces are given as meshes. D3DRM suggests several ways to define meshes. In our system we select the only interface – *IDirect3DRMMeshBuilder (or *2, or *3).*

Light sources. All types of light sources are introduced in our system: point, parallelpoint, directional, spotlight. Ambient light was stated as a feature of a whole world, as was mentioned above.

Decals. Rectangular flat posters (analogs of sprites) filled with specified texture are called decals. A decal has a fixed orientation with respect to camera.

Also, we distinguish such qualities as follows.

Textures – images, which can be wrapped onto surfaces – assigned to meshes.

Materials – light reflection properties of scene surfaces.

Animations – descriptions of dynamic behaviors of world objects – time dependencies of position, size, orientation with respect to other objects. Time is governed by the heartbeats of a scene.

3.3 Basic 3D Classes

CD3DMainObj – the class of 3D world. Its hidden function is to be an owner of all allocated objects as a means to watch after memory usage. A user creates his own scene as an object of an inherited class and thus gets all its functionality: fog, ambient light, camera, etc.

CD3DFrame – the second important class in the design. This class defines the object *frame*.

1. A frame can be seen as a model space with its' own coordinate system
2. Only the frame contains the mesh.
3. The texture object, material object and color can be assigned only to a mesh (meshbuilder).
4. The mesh shadows are added to the frame of that mesh.
5. A frame with a mesh is a *leaf frame* of a geometric construction.
6. A frame can be included into another frame (*node frame*) with the help of some affine transformation.
7. Different kinds of frames are identical in most respects.
8. Each frame may have rotational and linear velocities, which allow Direct3D to automatically change frame position and orientation at each scene heartbeat.
9. A scene is a frame too.
10. All objects that have a position or a direction are frames also such as lights, camera, decals, etc..
11. It is recommended to build all basic geometric shapes as classes inhereted from CD3DFrame class, e.g., CD3DConeFrame - side surface of a cone.
12. CD3DMainObj class is declared as

```
class CD3DMainObj : public CD3DFrame
```

and therefore a user has access to all properties of frames while constructing his scene, the object inherited from the CD3DMainObj class.

CD3DTexture – the class that defines a texture object which can be applied to some leaf frames (i.e., directly to their meshes). Note that we do not create the special class that covers the Wrap interface of D3DRM. Simply, the frame class has special methods for applying textures to a mesh (flat, cylindrical, spherical, chrome), if it is a frame with a mesh. Moreover, there exists the only way of defining of a texture image. It is loading it from file. If a user wishes more ways, he can easily add them by extending the system.

CD3DDecal – this class is derived from frame and texture classes. Its position is controlled as a position of a frame. Its image and properties (transparency, etc.) are taken from the corresponding texture object.

CD3DMaterial – is the objects class, which defines specular, emissive and diffuse reflection properties, and can be applied to any frame, even if it has a tree-like structure. In the latter case, the material object is assigned to all of the leaf frames down the hierarchy.

CD3DAnimation – is the object, which can be attached to any frame.

Lights are considered as frames (special class *CD3DLightFrame)* in a case when they are assigned to necessary positions or directions.

CD3DFrame - generic class

CD3DLightFrame - resumptive light object

CD3DDirectionalLight - particular light object

CD3DParallelPointLight

CD3DPointLight

CD3DSpotLight

CD3DCameraFrame – is the artificial class. It was introduced in order to control camera movements and orientations as usual frame.

4. CONSLUSION

The given design of C++ library as a shell above DirectDraw and D3DRM is simple but powerful enough. The developed system of classes can serve as a good first 3D tool for newcomers of 3D graphics. It represents a tool, which allows a user to create an acceptable prototype of an application in a short enough period of time. Then, the prototype can be transformed to a final application by means of evolution when a programmer creates the necessary absent classes or extends functionality of existent ones. In the case of insufficient productivity of the prototype, a programmer may select the revolutionary way – the prototype should be re-programmed, e.g., using Direct3D's Immediate Mode.

Initially, we propose to create this C++ tool as an improvised means for a quick preparation of 3D scenes (rough estimates of geometry and lights) in our experiments with the radiosity equation. Just as we had

found its usefulness, our next step consisted in the development of the given tool.

The C++ shell described in the given report was used as a canvas of lectures and seminar lessons of the course "Computer Graphics" at the Novosibirsk State University.

5. ACKNOWLEDGEMENTS

This work is supported in part by the Russian Foundation for Basic Research under a grant No. 99-01-00577 and the special federal program "Integration of Science and Education" under a project No.274 (course "Computer Graphics").

Authors are grateful to Dr. Viktor Sirotin for his permanent interest to this work and valuable remarks.

6. REFERENCES

[1] http://www.microsoft.com/directx/developer/information/default.asp - Microsoft DirectX developer resources start page.

[2] B.Bargen, T.P.Donnelly. *Inside Directx.* Microsoft Press, 1998.

[3] http://www.opengl.org/Documentation/Specs.html *OpenGL 1.1 Specification.*

[4] M.Stein, E.Bowman, G.Pierce. *Direct3d : Professional Referenc*. New Riders Publishing, 1997.

[5] http://www.microsoft.com/directx/dxm/help/d3drm/c-frame.htm#default.htm

[6] H.R. Hartson, D. Hix. *Human-Computer Interface Development: Concepts and Systems for its Management*. ACM Computing Surveys, Vol.21, No.1, 1989, p. 5-92.

[7] Nigel Thompson. 3D Graphics Programming for Windows 95. Microsoft Press, 1996.

[8] V. Sirotin, V. Debeloff, Urri. *DirectX-Programmierung mit Visual C++ 6*. Addison-Wesley, 1999.

[9] В.А. Дебелов, Ю.А. Ткачев. *Объектно-ориентированная система машинной графики для Windows (C++ и Microsoft DirectX)*. Изд-во СО РАН, 1999, в печати (in Russian).

[10] http://www.microsoft.com/directx/dxm/help/d3drm/oview/samples.htm - up to date samples.

[11] http://www.geocities.com/~directx/articles.html

Authors:

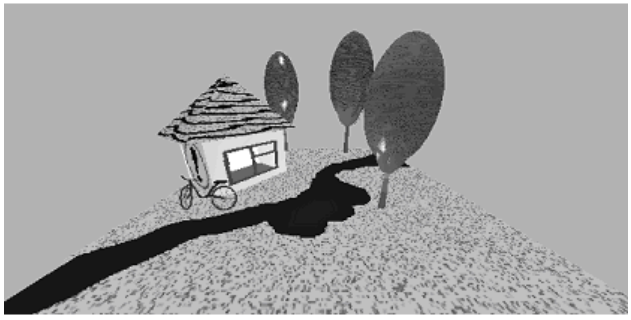
Laboratory of numerical analysis and computer graphics, Institute of Computational Mathematics and Mathematical Geophysics (former Novosibirsk Computing Center) of Siberian Branch of RAS.

Yuri A. Tkachov - researcher

E-mail: urri@oapmg.sscc.ru

Victor A. Debelov - Ph.D., senior researcher. A part-time professor of Novosibirsk State University

E-mail: debelov@oapmg.sscc.ru



Screen snapshot: static 3D application, usage of different classes of geometric forms, textures with transparency (i.e. a window, river), materials, decal with transparency (bicycle).