A Course in Programming and Computer Graphics Using Visual C++

R.W. Mayne

Professor
Department of Mechanical and Aerospace Engineering
University at Buffalo
State University of New York
Buffalo, NY 14260
Phone: 716-645-2593 ext. 2254
Fax: 716-645-3875
Email: mayne@eng.buffalo.edu

Abstract

This paper describes a course in computer graphics for seniors and graduate students in mechanical and aerospace engineering at the University at Buffalo. The course involves 3D graphics theory but also focuses on programming for computer graphics. It is taught in a PC Windows environment with Microsoft's Visual C++.

The paper provides a brief history of the course and its relationship to our other computer aided design offerings. We discuss our strategy for introducing students to programming with VC++ including initial object-oriented exercises without graphics and then programming approaches for basic 2D graphics operations in windows. This is followed by an implementation of 3D graphics programming using an object-oriented format and, lastly, our approach to introducing OpenGL for the PC.

Introduction

For many years, we have taught a computer graphics course for seniors and graduate students in mechanical and aerospace engineering at the University at Buffalo. This course is one in our series of courses in computer aided design and computer graphics. Other courses in the series include a mechanical design course using AutoCad and ProEngineer for design (recently made a requirement for BSME students), a second ProEngineer course considering finite elements, mechanisms and manufacturing, and a virtual reality graphics programming course based on workstation programming and including World Tool Kit. The course we are discussing here is normally a prerequisite for the virtual reality course.

Our computer graphics coursework originated in the days of Fortran programming and Tektronix "green screen" computer terminals well before the popularity of CAD packages. This introductory course to computer graphics programming has evolved through various

instructors, and hardware and software environments including UNIX and DOS. In recent years we have made a commitment to windows programming on personal computers using Microsoft's Visual C++.

The VC++ programming environment has proven to be very satisfactory. Students who have very little computer science background are able to adapt to C++ object-oriented programming and to the Application Wizard of VC++. They are able to do their programming in our department PC laboratories, they can take their work home easily using their own computers, and the programs that they produce look very much like the commercial programs they use routinely. At many points in the course there is the common reaction: "so this is the way it's done". The course develops specific programming and theoretical skills in computer graphics which transfer to any computing environment. But, by exposing students to PC windows programming, it also considerably broadens their computing skills.

This paper summarizes the nature of our computer graphics course and the various topics that we cover. It also is intended to provide some insight into Visual C++ programming and the way that it can be integrated into a computer graphics course.

Assumptions and Basics

This course is a cross-listed course offered at the undergraduate level (MAE 473) and at the graduate level (MAE 573) under the title Computer Graphics for CAD. Except for an occasional special case, the programming backgrounds of the undergraduates and graduates are much the same. Typically they will have had an exposure to C programming (probably as freshmen or sophomores) but have not done much programming recently. So we begin by reviewing C programming at the same time as the Visual C++ environment is introduced. The Applications Wizard of VC++ is used for windows programming in the course to facilitate programming without requiring students to become involved in the complete nitty-gritty of windows details. However, we initially use the VC++ "Console Application" format. In this form, when a compiled and linked program is executed, it produces a DOS-like window with an alphanumeric display without graphics capability. The programming, however, is quite simple. It can be done in a single file within the Integrated Development Environment of VC++. Code can be conveniently edited, compiled and linked. And program execution can be easily performed during program development without leaving the environment.

The Console Application format is very much appropriate for starting out in Visual C++. It provides the place to revisit arrays and pointers (where students usually need work) and to discuss object-oriented programming (which students only vaguely understand). This is the way Horton's[1] popular Visual C++ book begins. Other approaches, for example Gurewich and Gurewich[2], that move quickly into windows programming without first developing these concepts tend to be less suitable for engineering applications. Two of our example programs using objects are shown in Figures 1 and 2. These are the DOS windows of the Console Application format. The first of the programs in Figure 1 shows outputs from a "rectangles class" and a "circles class" by using appropriate member functions. In the program of Figure 2, the rectangles and circles classes are derived from a "TwoDobjects class". Virtual functions are used here so that an array of TwoDobjects containing both the rectangle and circle objects can be manipulated easily. Member functions designed for rectangles or circles are automatically called as needed for specific objects.

Windows Programming Concepts

The windows programming world is, of course, highly dependent on object-oriented methodology not only for programming but for basic organization itself. Writing programs for windows involves an incredible amount of detail, most of which is provided by functions and organization built into the compiler and the operating system. Like many Microsoft products, nearly everything is available in VC++ but in such abundance that it can be hard to find the most directly useful and interesting tools for an application. In this course we have tried to focus on those things which are most basic for engineering graphics and calculations. As students learn these basic tools and understand the VC++ environment they can move rather easily into other VC++ capabilities.

We have used the VC++ AppWizard executable form for our windows programs including the Document/View architecture. In this format, VC++ creates a program/project structure which serves as a template for the programmer's application. If the six step AppWizard start up process is followed, accepting all defaults for creating an AppWizard project, twenty five files of C++ code are created by the AppWizard alone. Without any added code, the resulting project can be compiled and linked producing additional files including an executable file. When executed, it will create a normal looking but blank window on the screen where nothing happens. Basically all of the VC++ prewritten code is used to put the blank window on the screen and provide the resources so that application programming can begin. In many ways, this makes programming for windows much more like writing a part of someone else's program rather than writing your own. Of course, plenty of work must still be done, but it requires an understanding of the environment and how to work within it.

The Document/View architecture of the AppWizard provides a program organization where data is considered to be part of a "document". Most programming takes place in two C++ classes. One class is a Document class for storing and manipulating the data in the document. The other class is a View class for visually displaying the document data and for handling interactions with a program user. The Document and View classes are both derived from higher level classes and have access to many convenient functions, classes and resources through the Visual C++ libraries and the Microsoft Foundation Classes (MFC). These include the graphics capabilities of OpenGL and Microsoft's DirectX which both facilitate software/hardware interaction through advanced graphics cards.

Beginning Windows Programming

The View class of a project contains the functions designed to place information on the screen and to interact with the user. The "OnDraw" function of the View class is called by the windows operating system as a program begins execution and also every time that it needs to update or redraw the program window. Figure 3 shows the window for our first windows program called "FirstWin". It contains both graphics and text drawn by the OnDraw function listed below. OnDraw is a member function of the View class. In the FirstWin program, the View class was automatically given the specific name CFirstWinView by the VC++ AppWizard as shown in line 01 (the prefix "C" simply means "class"). The OnDraw function was also prepared automatically in template form by the AppWizard and originally consisted of lines 01 – 05 and line 17 below. Line 05 provides the cue for the programmer to add his/her own code. We have added lines 06 – 16 and these are the only lines of code that have been added anywhere in

the FirstWin program. In the code, line 01 indicates that the pointer pDC is passed into the OnDraw function as it is called by the windows operating system. This pointer identifies the device context (i.e., the current window) where OnDraw should be drawing. Line 08 uses the pDC pointer to activate the TextOut function, placing "This is a Box and a Dot!" in the window beginning 25 pixels from the top of the window and 50 pixels from the left edge. Line 07 declares a CString object called “Buffer” and line 09 formats Buffer to contain some text and the two integer variables that define the box center. Line 10 again uses the pointer pDC to identify the window and then writes Buffer just below the box and dot phrase. Line 11 sets the pixel at the box center and lines 12 – 16 actually draw the box.

```
01    void CFirstWinView::OnDraw(CDC* pDC)
02    {
03          CFirstWinDoc* pDoc = GetDocument();
04          ASSERT_VALID(pDoc);
05          // TODO: add draw code for native data here
06          int CenterX = 300, CenterY =50;
07          CString Buffer;
08          pDC->TextOut(50,25,"This is a Box and a Dot!");
09          Buffer.Format("The Dot is at %d, %d",CenterX, CenterY);
10          pDC->TextOut(50,50,Buffer);
11          pDC->SetPixel(CenterX,CenterY,0);
12          pDC->MoveTo(CenterX - 25, CenterY - 25);
13          pDC->LineTo  (CenterX + 25, CenterY - 25);
14          pDC->LineTo  (CenterX + 25, CenterY + 25);
15          pDC->LineTo  (CenterX - 25, CenterY + 25);
16          pDC->LineTo  (CenterX - 25, CenterY - 25);
17    }
```

The OnDraw function above draws the window shown in Figure 3 very nicely but does not do anything else. In order to interact with a user, windows messages (which result, for example, when a key is pressed or a mouse is clicked) must be processed or command functions must be developed to respond to clicks on the menu or tool bars. The appropriate functions can be created through the VC++ ClassWizard and can be automatically introduced into the View class or Document class as desired. The function below is shown as an example of a mouse message handler in a simple drawing program called SimpleDraw. The View class in this case is called CSimpleDrawView and this particular function (called OnLButtonDown) will be executed whenever a left mouse click is received in the SimpleDraw window. Lines 01 – 03 and line 14 were furnished by VC++ as the function was created. Lines 04 – 13 contain the code which has been added.

```
01    void CSimpleDrawView::OnLButtonDown(UINT nFlags, CPoint point)
02    {
03          // TODO: Add your message handler code here and/or call default
04          CSimpleDrawDoc* pDoc = GetDocument();
05          CClientDC aDC(this);
06          aDC.SelectObject(&m_BluePen);
```

```
07              int i = pDoc->NumLines;
08              aDC.MoveTo(pDoc->FPoint[i]);
09              aDC.LineTo(point);
10              pDoc->LPoint[i] = point;
11              pDoc->FPoint[i+1] = point;
12              pDoc->NumLines++;
13              CView::OnLButtonDown(nFlags, point);
14      }
```

Line 04 obtains a pointer pDoc to the "document" that is being used to store data for the drawing being made. Line 05 associates the "client device context" aDC with the current window. Line 06 gets a blue pen ready for drawing. Line 07 uses the document pointer to find the current number of lines in the drawing. Line 08 moves to the first point on the current line (already stored in the document). Line 09 draws a line to the point just clicked. The variable "point" takes on the mouse cursor position when the mouse is clicked as shown in the argument list of line 01. Then, lines 10 and 11 store the new point in the document both as the last point on the current line and also as the first point on the next line. Finally, line 12 increments the line counter in the document. In this arrangement, the OnDraw function of the SimpleDraw program is coded so that it redraws the whole set of lines from first to last when it is called by Windows as a resizing, minimizing or adjustment of the program window takes place.

Figure 4 is an example of a mouse drawing program that has been extended to demonstrate the development of specialized menu bar and tool bar items. The VC++ resource package makes each of these items readily definable and allows them to be associated with functions that will be activated by clicking on them. In this program, mouse clicks on menu items and drop down menu items activate functions which can change parameter values to adjust line color, line thickness, etc. The tool bar items are normally defined to act as shortcuts to the drop down menu items.

Further into Windows

While the programming above becomes rapidly intricate, students are able to understand the concepts very well. Although it's not practical to have them program completely from scratch, student assignments have been typically defined to begin with an example program from the wide set of examples that we have been developing. For 2D graphics, we have often asked students to develop a mini CAD system including circles, arcs, rubber banding, and even including snap-to-grid operations for accurate drawings. Saving data to file is also included using the Serialize function from the Document class which provides access to the normal windows filing process. Programs similar to Figure 5 might often result from student projects. This contains adjustable snap-to-grid, a multidocument format and a customized file extension As shown in Figure 5, there are two open documents (2Sq.snp is the active one) and the file window is Open ready to give access to another file.

It is also possible to expand student horizons by including functions available for their use in programming. In the 2D graphics world, a possibility that we will explore in the next offering of MAE 473/573 will be the inclusion of classes and functions for drawing graphs as part of the example program files available to students. We currently have a basic 2D curve plotting function as well as a color contour plotter to use, for example, in plotting temperature or

stress distributions. Figure 6 contains representative graphs for an end-loaded beam. The beam sketch was created with a version of the mouse drawing program. The plot of beam tensile stress and the stress contour plot were drawn in the View class OnDraw function using the graph plotting tools and based on data generated by program code in the OnNewDocument function of the Document class. This function executes as the program starts.

Graphics in 3D

One difficulty in teaching computer graphics is to decide how much canned software to make available to students and how much individual programming is required by students. If students are required to program everything from scratch, too much effort is required even for modestly interesting programs. On the other hand, if too many "black box" functions are used, it is possible for students to create impressive programs while avoiding important programming issues and theoretical details of computer graphics such as those in Foley and van Dam[3].

In dealing with 3D graphics we have made a successful compromise by making basic 3D classes and functions available to students. The complete coding for these classes and functions is contained in example programs and is used for in-class discussion of theory and programming. Students are also asked, as part of their programming projects, to modify and extend the code in the existing examples. These modifications have included the removal of hidden lines in a projection algorithm, the improvement of 3D rotation functions, and the creation of 3D solid representations for extrusions and bodies of revolution. The object-oriented programming structure of C++ used in the examples makes it quite easy to focus on particular parts of the program and consider the applicable theory.

To generate 3D solid objects, we have used a series of derived classes to represent the object. The first class is the BasePoint class which contains the global coordinates of the object in three dimensions. The second class is the Points class which contains all of the vertices representing the object in local coordinates (relative to the BasePoint). The third class is the Polygons class which defines each of the polygons forming the solid in terms of the vertices. Finally, the Shapes class is used to describe any specific shape by defining its base point, vertices and polygons. A shape can be easily translated by changing its base point, it can be rotated by rotating its vertices and it can be displayed by projecting and drawing its polygons. The code fragment below is taken from the declaration of the Shapes class to provide an indication of the organization. In line 01 you can see that Shapes is derived from Polygons (which is derived from Points and BasePoint). Line 03 is the empty Shapes constructor function and lines 04, 05 and 06 show the argument list for the function which actually defines an object shape. The base point coordinates are included in the argument list, the number of vertices (NPts), pointers to the vertex coordinates, number of polygons, number of points per polygon, etc. Coding is not shown

```
01    class Shapes : Polygons {
02    public:
03          Shapes(){};
04          void Shapes::AdjustShape (double Xbp, double Ybp, double Zbp, int NPts,
05                double* ptX, double* ptY, double* ptZ, int NPolys, int* ptNPPts,
06                int* ptMPPts){…….}
07          void Shapes::RotateX(double thetaX){
08          Points::PRotateX(thetaX);}
```

```
09      void Shapes::RotateY(double thetaY){
10      Points::PRotateY(thetaY);}
11      void Shapes::RotateZ(double thetaZ){
12      Points::PRotateZ(thetaZ);}
13      void Shapes::Move(double nx, double ny, double nz){
14      Basepoint::MoveBP(dx,dy,dz);}
15      void Shapes::Dmove(double dx, double dy, double dz){
16      Basepoint::DmoveBP(nx,ny,nz);}
17      void Shapes::Show(CDC* aDC){
18      Polygons::Trace(aDC);       }
```

for this function. However, lines 07 – 18 do show complete functions for the rest of the Shapes class. For example, lines 07 and 08 will rotate a shape about an X axis through its base point by simply calling the Points function which rotates all of its vertices. Lines 13 and 14 can move a shape in three dimensions by calling the BasePoint function which changes the base point coordinates. The Show function of lines 17 and 18 displays the shape on the screen by calling the Polygons function which carries out the drawing of each polygon.

This program structure has proven to be very robust and useful. Projection algorithms can be studied by focusing on the Polygons function "Trace". This has been done for hidden line removal and for stereo projection. Higher level shape constructors have been easily written for cylinders and bodies of revolution. And a Bezier curve class has been written which makes use of the BasePoint and Points classes. Students can explore the coding in particular functions and assignments can be conveniently made to modify or improve the code. Figure 7 shows some simple 3D projections made from the basic programs. Figure 8 shows a display from a version with hidden line removal and Bezier curve capability.

OpenGL

For advanced level graphics on a personal computer, it is ultimately necessary to use OpenGL (Silicon Graphics) or DirectX (Microsoft). These software packages operate in conjunction with compatible graphics cards which allow hardware implementation of graphics functions. The software will let the hardware function at its limit and fill in with software implementation where necessary. We have been successful in incorporating OpenGL capability into this course and into our set of program examples for the course. We have found the official OpenGL guide [4] to be useful for OpenGL programming issues. The SuperBible [5] and the tutorial by Oursland [6] are especially useful for working with OpenGL in a PC environment. These later references offer strategies for inserting OpenGL graphics into the AppWizard. Interestingly it has been possible to adapt our object-oriented 3D graphics discussed above almost directly into OpenGL graphics and achieve impressively quick rendering. We will not attempt to discuss the implementation detail here. But, with modest modification of our Polygon "Trace" function, OpenGL commands can be used directly. The overall result is that 3D shapes can be easily generated, moved, rotated and displayed with our object-oriented programming tools. The higher level OpenGL programming tends to control viewing position and angle, lighting, background color, etc.

Figure 9 shows a first OpenGL program with the cylinder and box of Figure 7. In Figure 9, the OpenGL shading quality and ability to remove hidden surfaces by depth buffering is very

clear. With this capability it is reasonably easy to produce quality graphics and animations. The driving simulator of Figure 10 is an example discussed in class and which students can modify for their own projects. The number on the car hood in Figure 10 is an example of a student modification where the texture mapping necessary to place the number on the hood was introduced as part of a final project for the course. The roller coaster animation of Figure 11 is another example of a student course project. In this animation, a six car roller coaster follows the track and can be viewed from adjustable positions including the front seat. The robot of Figure 12 is a further example. It can be moved in steps or animated and can pick up the small object shown near its gripper.

Conclusions

This paper has presented an approach to teaching computer graphics in a PC Windows environment. The course has been quite successful and of interest to both seniors and graduate students. It provides an exposure to the process of windows programming in general and into the operation of both CAD and computer animation programs. In addition, the course serves the purpose of preparing students for further study and research in computer graphics.

The paper is also intended to provide some insights into the process of Visual C++ programming and an indication of programming approaches for basic two-D graphics operations. A strategy for using object-oriented 3D graphics in instruction has also been described. This process is attractive for being able to allow students to inspect and modify functions which service different aspects of creation, manipulation and display of 3D representations. The strategy extends directly into the higher-level graphics capabilities of OpenGL where significant animations and simulations can be conveniently developed.

In terms of scheduling - students typically complete the 2D graphics project about half way through the course. Lecturing on 3D graphics begins at about the 40 percent point. Two 3D graphics projects are also required. One is a specifically defined project performed without OpenGL and one is a student selected project using OpenGL.

Bibliography

1. Horton, I., *Beginning Visual C++ 6*, Wrox Press Ltd, Birmingham, UK, 1998.
2. Gurewich, O. and Gurewich, N., *Teach Yourself Visual C++ in 21 Days*, Fourth Edition, Sams Publishing, 1997.
3. Foley, J.D., et al., *Computer Graphics Principles and Practice*, Second Edition, Addison Wesley, 1990.
4. Wright, R.S. and Sweet, M., *OpenGL SuperBible*, Second Edition, Waite Group Press, 2000.
5. Woo, M., et al., *OpenGL Programming Guide*, Third Edition, Addison Wesley, 1999.
6. Oursland, N.A., *Using OpenGL in Visual C++*, www.DevCentral.Iftech.com.

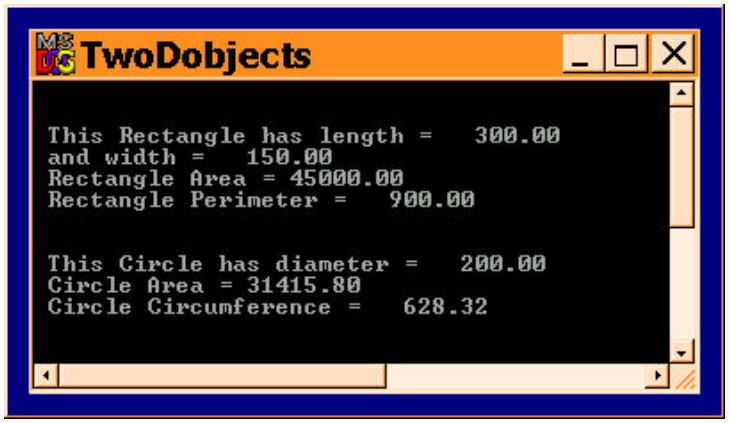

Figure 1. Object-Oriented Concepts

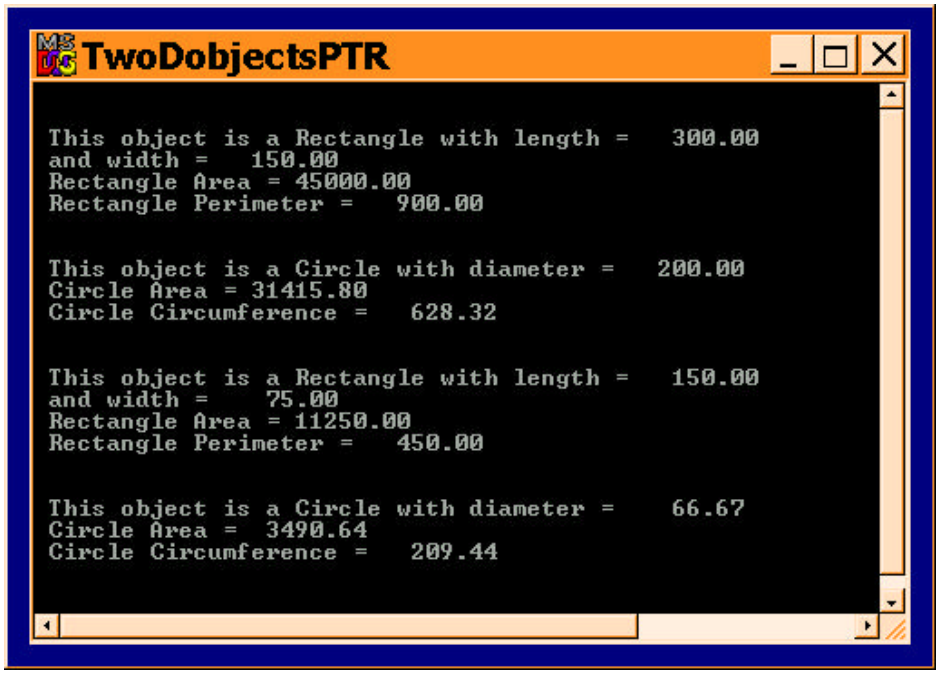

Figure 2. Object-Oriented with Pointers and Virtual Functions

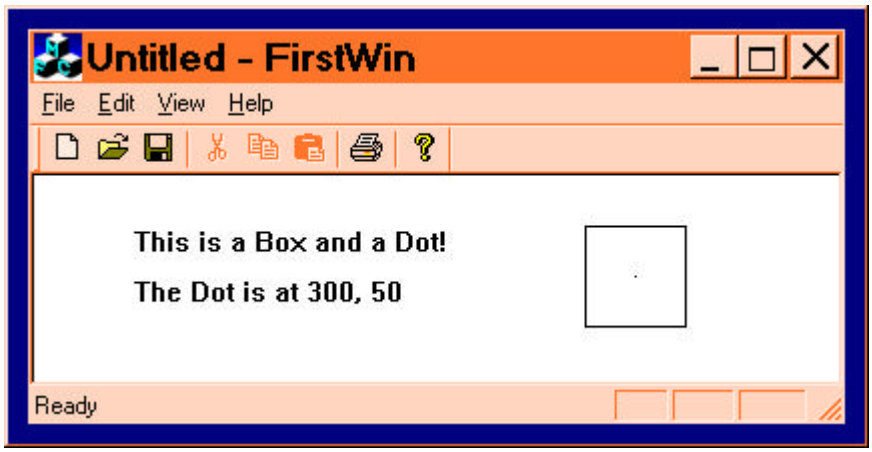

Figure 3. First Graphics in a Window

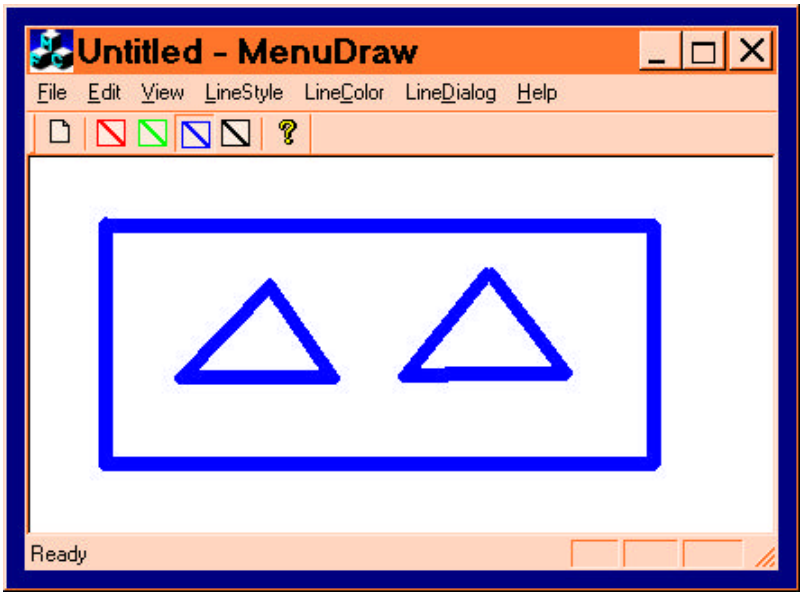

Figure 4. Drawing with the Mouse and Creating Menu and Toolbar Items

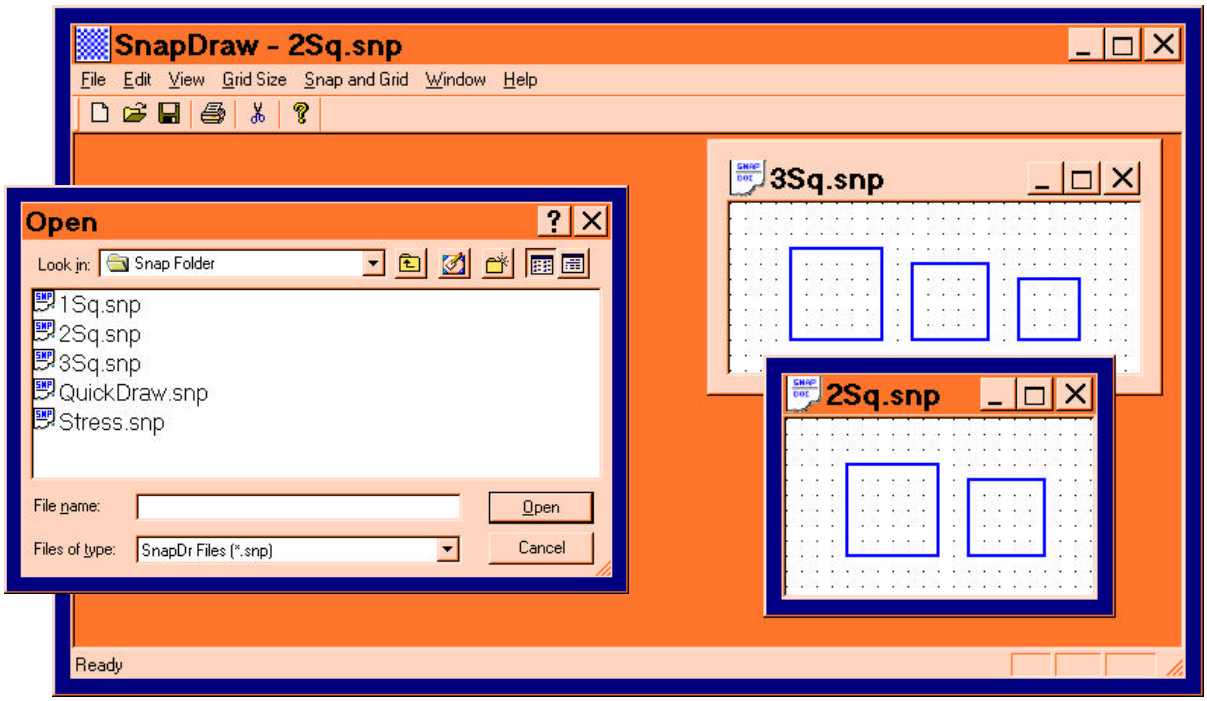

Figure 5. Drawing in a MultiWindow Environment

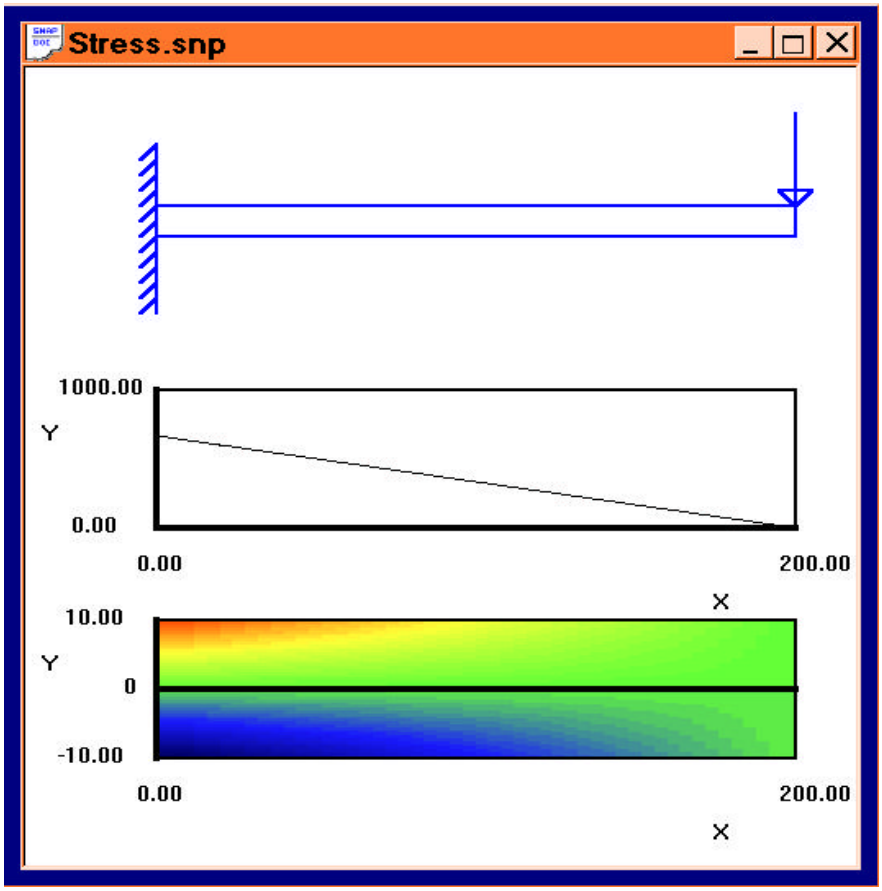

Figure 6. Mouse Drawn Sketch with Program Drawn Graphs

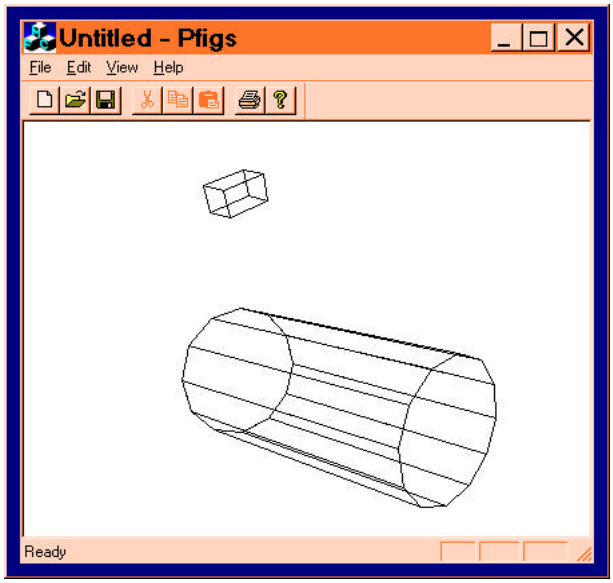

Figure 7. Simple 3D Projections

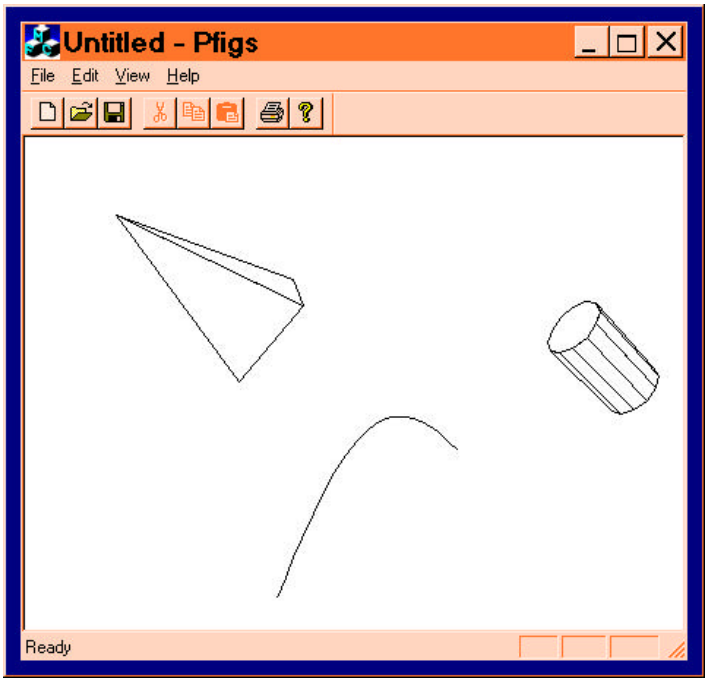

Figure 8. A Bezier Curve and Removed Hidden Lines

Figure 9. Starting with OpenGL

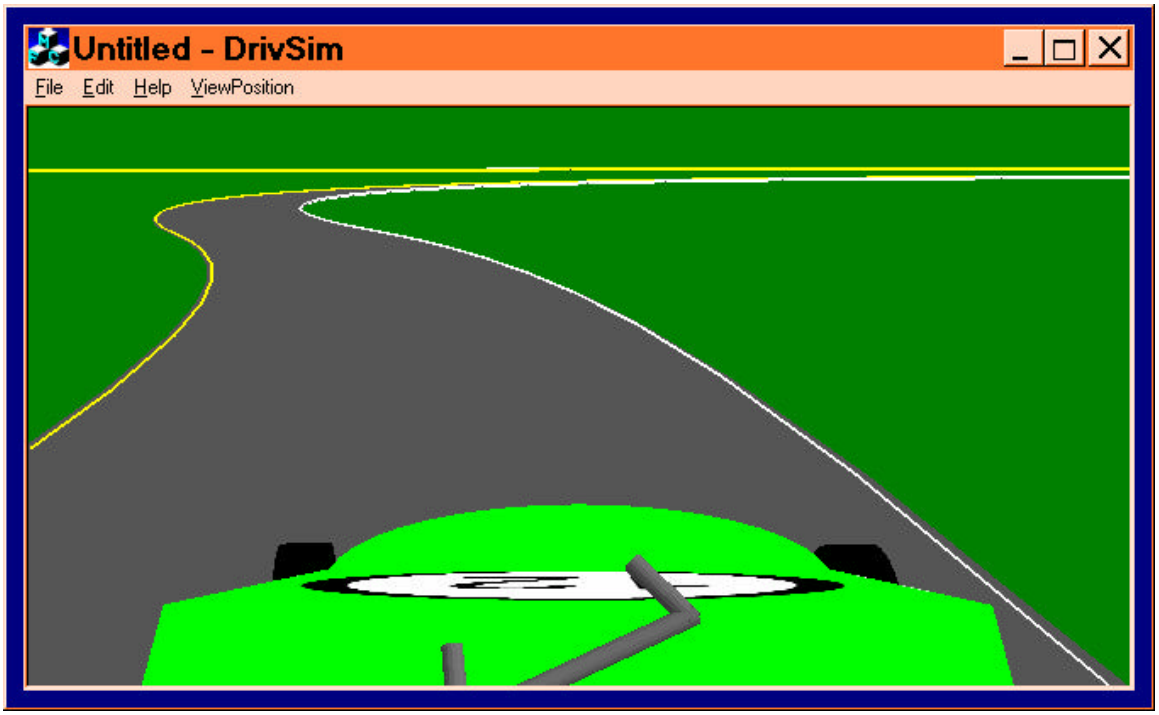

Figure 10. Basics of Driving Simulation

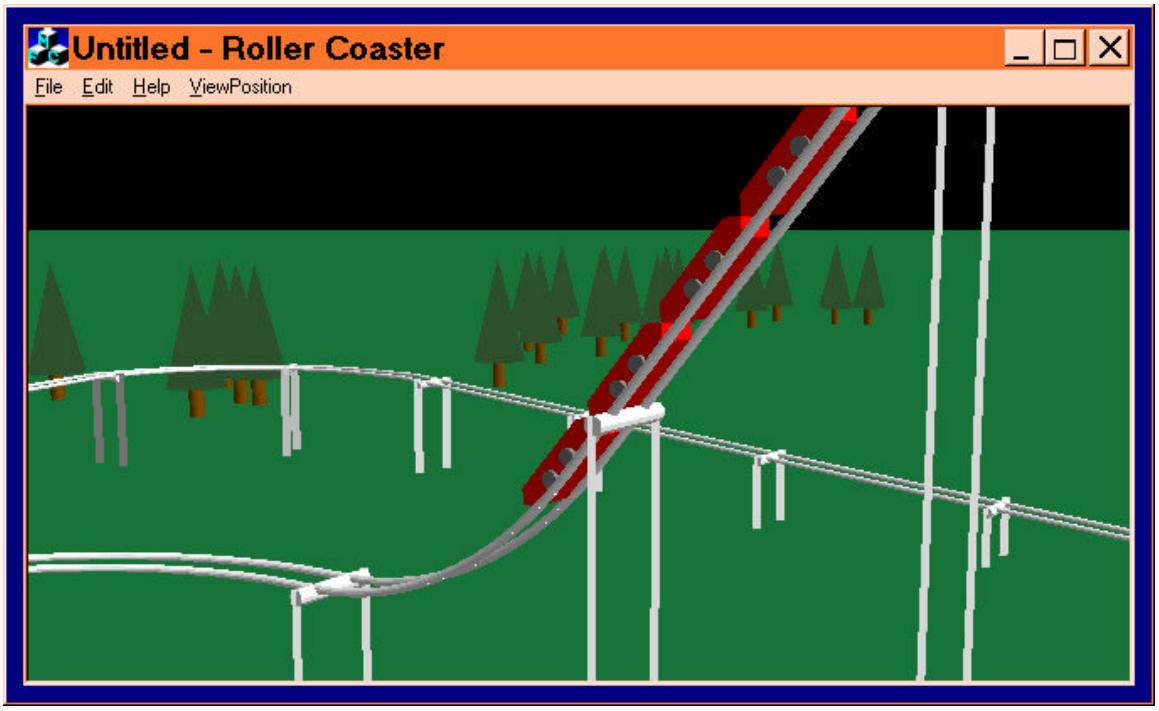

Figure 11. Student Roller Coaster Project

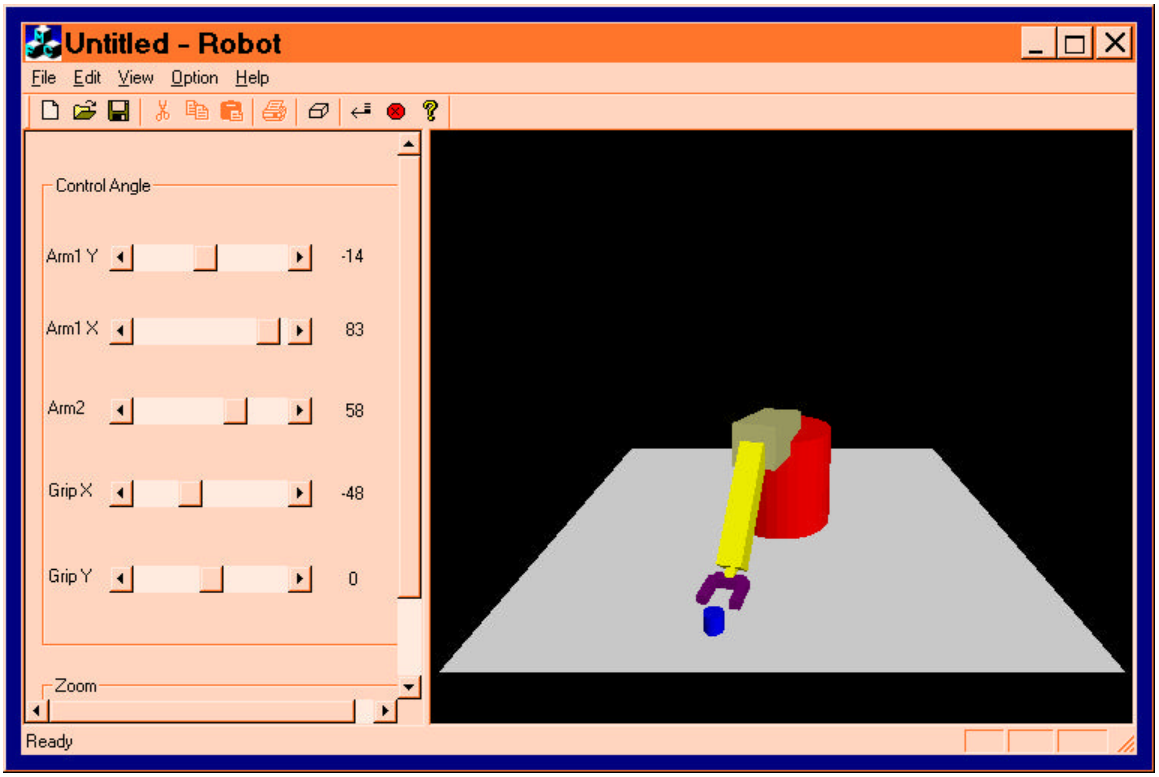

Figure 12. Student Robot Project