Using the Wiimote in Computer Graphics Projects

Joseph Clifton
Computer Science and Software Engineering
University of Wisconsin – Platteville
Platteville, WI 53818
clifton@uwplatt.edu

Abstract

The enrollment in the Computer Graphics course at our university had gradually declined over the past twenty years. It had developed a reputation of being difficult and too mathematical. In the fall of 2008, we reworked the course to focus more on 3-D concepts and less on mathematics and implementation of algorithms. In addition, we developed some game-based projects that used the Nintendo Wii Remote™ (Wiimote). We advertised these changes to the students registering for fall 2009 courses with the result that for the first time ever, the course filled and enrollment had to be limited. In this paper, we describe the changes to the Computer Graphics course and the use of the Wiimote as an input device for projects in the course.

1. INTRODUCTION

The Computer Graphics course at the undergraduate level takes many different forms and is taught at many different levels. Some small colleges are fortunate enough to be able to offer multiple courses in Computer Graphics, particularly those that have an emphasis in game programming. For those of us that only have resources for one such course, we struggle to decide on the best approach.

The Computer Graphics course at our university had been suffering from decreasing enrollments, especially over the past ten years. We had tried various changes in an attempt to turn this around, but none produced a sustained increase in enrollment. In fall 2008, we decided to make changes that were more substantial. We looked to a report from the working group on Computer Graphics in Computer Science at the SIGGRAPH/Eurographics Computer Graphics Education workshop in Zhejiang University, Hangzhou, China, June 2-6, 2004. In that report, the authors state:

"Computer graphics courses grew up following a very few key textbooks that were based on the technology of the 1970s and 1980s, when graphics standards were 2D and clumsy, graphics was an expensive and nonstandard capability, and there were no standard graphics tools. Faculty who taught graphics grew up in that environment and it was difficult to get people to think about the computer graphics course in a new and more general setting. Computer graphics had the reputation of being very difficult and very mathematical, as it originally was." [2]

They go on to affirm the agreements made at the 1999 and 2002 meetings, including the use of graphics pipelines and the importance of including interaction techniques. Then they provide a list of knowledge topics that students who take a Computer Graphics course should know and indicate the level at which they should be taught:

- Transformations
- Modeling: primitives, surfaces, and scene graphs
- Viewing and projection
- Perception and color models
- Lighting and shading
- Interaction, both event-driven and using selection
- Animation and time-dependent behavior
- Texture mapping

"These knowledge topics are intended to be independent of the graphics API and hardware used in the course, and should be largely independent of such problem-structuring tools as the scene graph. In general we believe that this knowledge should be conceptual and should not require that the student implement the various algorithms and processes that go into these topics." [2]

2. BACKGROUND

We have offered a Computer Graphics course at the upper-division level since the early 1980's. Our course suffered from some of the problems listed in the first quote in the previous section. The students considered the course difficult with too much mathematics. The prerequisites for the course were Data Structures and Linear Algebra, and Linear Algebra required two semesters of Calculus.

A significant portion of the course was spent on implementation details of algorithms such as scan conversion of lines and circles, line and polygon clipping, hidden line and surface removal, polygon fill routines, and anti-aliasing. We used texts such as those by Hearn & Baker [3] and Pokorny & Gerald [7].

We offer the Computer Graphics course in the fall of odd years. The author has taught the course since 1989. As seen in Table 1, there has been a gradual decrease in enrollment. During the past ten years, we have tried a number of changes; however, the math level remained about the same and a significant amount of implementation was still required. As a result, the success was generally limited and temporary. The low point came in 2007, when enrollment dropped to eight students.

Term	Course Enrollment	Number of Undergraduate Majors	***Approx % of Undergraduate Majors Taking Course
Fall 1989	22	132	17%
Fall 1991	14	119	12%
Fall 1993	22	106	21%
Fall 1995	17	134	13%
Fall 1997	18	132	14%
Fall 1999	15	186 **	8%
Fall 2001	14	233 **	6%
Fall 2003	7 / 7 *	210 **	3%
Fall 2005	13 / 6 *	211 **	6%
Fall 2007	7 / 1 *	257 **	3%
Fall 2009	31 / 7 *	291 **	11%

Table 1. Computer Graphics Enrollment

*The left number is the number of undergraduates enrolled. The right number is the number of Master's students enrolled. A Joint International Master's program was established with a partner school in Germany in 2003. Students from each institution are required to spend one semester at the partner institution. Most of the Master's students enrolled were from Germany.
**This includes both the Computer Science and the Software Engineering majors. The Software Engineering major was established in 1999.

*** Approximate since occasionally a Computer Science minor will take this course; however, we have few students who minor in Computer Science, and it is rare for one of them to take this course.

Due to state budget constraints over the past two or three years, administration at our university established minimum target enrollments for courses. The first established target was ten students. Soon after, this was raised to fifteen and now stands at twenty. A course with enrollment below the minimum is subject to cancellation, unless a strong case can be made to offer it. This also provided motivation to modify the course.

In the fall of 2008, we reworked the Computer Graphics course and presented the changes to the University Undergraduate Curriculum Commission in December of 2008. The changes are outlined in the next section. The continuing students started registering for fall 2009 courses in March of 2009. Prior to this enrollment period, we sent an email to all upper-division majors outlining the approved changes to the Computer Graphics course. Furthermore, we noted that there would be some games-based projects using the Wiimote. We were pleasantly surprised when the course filled at 38 students, which is the fire-code limit for the room.

3. COURSE CHANGES

One major change to the course was to lower the mathematics prerequisite. Calculus I was chosen as the new mathematics prerequisite, which we thought was sufficient maturity for the course. Although this meant that knowledge of matrices could no longer be assumed, the necessary material could be covered in about two lecture periods. Very little of the theory taught in linear algebra is really needed to be able to understand and use linear transformations.

Another major change was to align the course with the topics listed in Section 1. The previous course description and learning outcomes were:

> An introduction to computer graphics including raster hardware, standard graphics software packages and important algorithms such as window-to-viewport mapping; clipping; 2D and 3D transformations and hidden line/surface removal. In addition, topics such as 3-D modeling, illumination, fractals, and animation will be covered.
>
> Upon completion of this course, students should be able to:
>
> 1. Understand the basic principles behind graphics systems hardware and software
> 2. Understand the need for graphics standards
> 3. Write and implement scan conversions algorithms for lines and circles
> 4. Write algorithms for filling a polygon and clipping a line
> 5. Set up 2-D window-to-viewport mappings
> 6. Develop object representations and models for simple 3-D scenes

7. Understand and apply the underlying mathematics of rotations, translations, and scalings for 2-D and 3-D objects
8. Understand the underlying mathematics of 3-D viewing and view plane transformations
9. Understand the various techniques for hidden line and surface removal
10. Use OpenGL to render 3-D scenes

The new course description and learning outcomes are:

> An introduction to computer graphics including transformations; modeling; viewing and projection; color, lighting and shading; texture mapping; interaction; and animation. Use of a pipeline-based graphics library such as OpenGL. Several programming assignments, including some games-based projects.
>
> Upon completion of this course, students should be able to:
> 1. Understand and apply 2-D and 3-D Transformations
> 2. Understand and use 3-D Modeling: primitives, surfaces, and scene graphs
> 3. Understand and apply Viewing and Projection
> 4. Understand and use Color Models
> 5. Understand and use Lighting and Shading Models
> 6. Understand and apply Texture Mapping
> 7. Understand and implement Interaction, both event-driven and selection
> 8. Understand and implement Animation and time-dependent behavior
> 9. Use OpenGL to create graphics applications

The pipelined Graphics API chosen was OpenGL. The textbook chosen was Interactive Computer Graphics: A Top-Down Approach with OpenGL by Edward Angel [1]. In addition, we decided to use the Nintendo Wii Remote™ (Wiimote) as an input device for some of the graphics projects.

The tentative outline of topic coverage presented in the syllabus was:

Topic	**Text Chapter**
Graphics Introduction: Systems, Standards, Architectures	1
Graphics Programming Intro: OpenGL, C#, primitives, attributes, color, viewing	2 & supplements
Graphical User Interfaces, inputs, event-driven input	3 & supplements
Geometric Objects, Intro to Modeling, 2-D & 3-D Transformations	4
Viewing and Projection	5
Lighting, Shading	6
Modeling, Animation	10, 11
Texture Mapping	8.6 – 8.10
Programmable Shaders	9

All the topics were covered in reasonable depth except the last two. Those were only covered at a high level and were not used in any of the programming assignments.

Although coverage of algorithmic detail was not listed in the above topics, the author could not, in good conscience, leave it out altogether. Some coverage is "good" for the students, so we covered algorithms such as Bresenham's line generation, a few different ways to generate a circle, digital differential curve generation, and recursive as well as iterative generation of fractals. Moreover, besides fulfilling all the undergraduate requirements, those students taking the course for graduate credit were required to research a current 3-D graphics technique and/or algorithm and write a paper that:

- Explains the technique or algorithm, using appropriate mathematics
- Analyzes the advantages and disadvantages over existing techniques or algorithms
- Discusses implementation issues for at least two programming languages
- Proposes and informally justifies at least one modification that would improve it. This does not need to be formally proved. Moreover, it doesn't even need to be technically feasible at this time.

The reduction in mathematics was not as drastic as might be implied by the lower mathematics prerequisites. Vectors and matrices were covered in class and even those who had not had linear algebra were able to effectively use and understand them to set up transformations. Some quiz and test questions remained similar to those in the past and still required a reasonable amount of mathematics. Four sample questions are:

> Set up matrices that represent a 3-D rotation of the point (100, 200, 300) of 30 degrees about the x-axis followed by a 60^o rotation about the z-axis. Recall sine of 30^o is ½, cosine is $\sqrt{3} / 2$. It is the reverse for 60^o.
>
> Give the perspective projection matrix assuming the eyeball is at (0, 0, d) and the projection plane is at z = 0. Draw a picture using y and z (x and z will be analogous), work out the similar triangles, and then use this to make the matrix. Remember homogeneous coordinates – use that denominator! Show ALL work in deriving this.
>
> Set up the first 3 matrices, in order, to do a rotation about P1 = (50, 20, 30), P2 = (62, 23, 34). Do not set up the other 4 matrices. You must get all numbers, as shown in class, and show all work. Hint: 3, 4, 5 is a right triangle as is 5, 12, 13.
>
> Set up the first three (3) matrices, in order, that give the look-at matrix assuming: V = View Point (eye) = (50, 20, 30), R = Ref Point (look at) = (70, 29, 42), U= Up = (0, 1, 0). Do not set up the matrix for aligning the Up vector. You must get all numbers, as shown in class. Hint: 3, 4, 5 is a right triangle, as is any multiple, such as 9, 12, 15 and 15, 20, 25.

The students were not allowed to use books or notes for the tests and quizzes. However, for the last two questions above they were given the diagram in Figure 1 to assist in their derivations of the matrices.

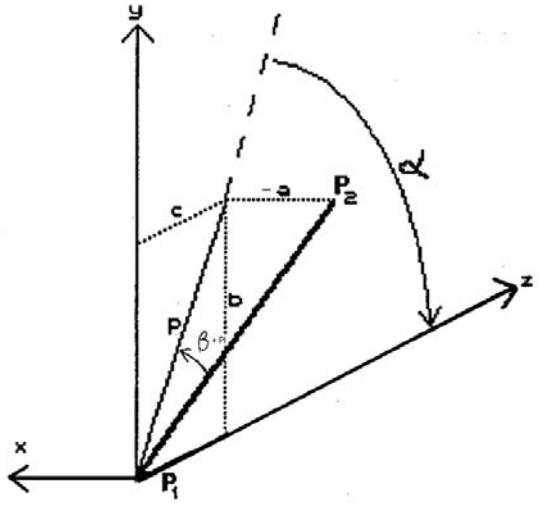

Figure 1: Diagram for Test and Quiz Problems

4. HARDWARE AND SOFTWARE

Johnny Chung Lee's head and finger tracking projects originally motivated us to use the Wiimote [5]. Johnny used C# and Visual Studio .NET, so we decided to adopt these for the project. Our students are exposed to Visual Studio .NET in the prerequisite course, albeit using C++, so we believed the switch to C# should be easy for upper-division students. Since our department belongs to the Microsoft Development Network Academic Alliance (MSDNAA), the software was on all of our Windows-based computers, and the students were allowed to install the software on their personal computers free of charge.

A freeware Managed Library for Nintendo's Wiimote, WiimoteLib, is available through the Coding4Fun project [6]. It is object-oriented and allows an instance of a Wiimote to be created and easily manipulated. The states of all the buttons and sensors can be queried, and the device outputs, such as lights and rumble, can be controlled.

Next we had to choose a graphics API. Both DirectX and OpenGL were considered, with OpenGL being chosen since it aligned well with the selected textbook. The Tao Framework for .NET was chosen as the OpenGL wrapper [4]. Besides libraries for OpenGL, GLU, and GLUT, it has libraries for game development, although as it turned out, we didn't use any of the gaming libraries.

For hardware, besides the Wiimote, a sensor bar and Bluetooth adapter are needed. The sensor bar consists of two infrared light sources that are read by the Wiimote's infrared cameras. Nyko makes a wireless sensor bar for the Wiimote that sells for about $15.

Finding compatible Bluetooth adapters is somewhat tricky. The access to the Wiimote was reverse engineered and getting it to work with an arbitrary Bluetooth stack is problematic. A list of working Bluetooth devices is given in [8]. We chose IOGEAR's Enhanced Data Rate Bluetooth USB Adapter, which sells for about $25. Another problem is the drivers. The latest drivers for IOGEAR's device require that a passkey be entered to connect to a Bluetooth device, which is compatible with the current Bluetooth standard. However, the Wiimote connection doesn't use a passkey and trying null or 0's doesn't work with these drivers. Therefore, this requires that either older IOGEAR drivers be used or the standard Windows BlueTooth drivers be installed. The sequence to install the IOGEAR device using Windows standard drivers is:

- Log into the computer with administrator rights
- Plug in IOGEAR USB Bluetooth adapter
- Let Windows XP automatically install drivers
- Click on Bluetooth icon to Add device, then click Add
- Check My device is ready/setup
- Press and release buttons "1" and "2" on WiiMote for discovery mode
- Wait for WiiMote discovery
- Click Nintendo device once found, then click Next
- Check "Do not use a passkey"

5. PROJECT DESCRIPTION

The project used OpenGL and was implemented in four phases. For the first phase, the students developed an object-oriented framework for displaying a 3-D figure stored in the Virtual Reality Modeling Language (VRML) 2.0 format. Classes were required for Face, Shape (a list of Faces), and Figure (list of Shapes). They were required to provide a mechanism to view the figure from any reference point. In addition, they were required to create at least one figure using any tool that exported VRML 2.0; however, the open source software Wings 3D was available on the departmental lab computers, so most students used it.

For the second phase, they added support for:

- Multiple figures on the screen
- Rotation, scaling, and translation of any and all figures
- A MovePattern class and subclasses that could be associated with Figure instances to allow them to move when triggered by a timer, thus providing animation

For the third phase, the students turned the program into a game, with a spaceship that moved around a universe hunting "space garbage" (the various VMRL figures created by the students). The spaceship was manipulated with the Wiimote and could fire projectiles at the space garbage. They were required to encapsulate the spaceship in a class. Furthermore, they were required to put a wrapper class around the functionality provided by the WiimoteLib.

The sparsely commented code below gives some pieces of a typical WiiMote wrapper class. Note that it uses the WiimoteLib, which has Wiimote and WiimoteState classes. In this example, the library class name differs from the wrapper class name by capitalization of a single letter. This is generally not good practice; however, as WiiMote is a wrapper class, the use of the Wiimote library class is limited to this class only.

```
using System;
using WiimoteLib;

public class WiiMote
{
  private Wiimote  wm;
  private WiimoteState  ws;

  public WiiMote()
  {
    wm = new Wiimote();    // Create a Wiimote instance

    // Set up the callback event to handle state changes
    wm.WiimoteChanged += wm_WiimoteChanged;

    // Connect to the Wiimote
    wm.Connect();

    // This report gives buttons, accelerometer, and IR data.
    // Other options are available, including extensions.
    wm.SetReportType(InputReport.IRAccel, true);
  }

  public void ShutDown()
  {
    wm.Disconnect();
  }

  // Called about 1000 times per second by WiimoteLib
  void wm_WiimoteChanged(object sender,
                                  WiimoteChangedEventArgs args)
  {
    // All the current state information
```

```
    ws = args.WiimoteState;

    // Sample button checks and actions – constants and
    // data members not shown.  In this example, the spaceship
    // was implemented as a Singleton.

    if (ws.ButtonState.Left)
      thetaChange = TURN_AMOUNT;
    else if ( ws.ButtonState.Right )
      thetaChange = -TURN_AMOUNT;
    if (ws.ButtonState.Up)
      phiChange = -TURN_AMOUNT;
    else if (ws.ButtonState.Down)
      phiChange = TURN_AMOUNT;

    if (phiChange != 0 || thetaChange != 0)
      Ship.Instance.ChangeDirection(thetaChange, phiChange);

    if (ws.ButtonState.B)
      Ship.Instance.Move(MOVE_AMOUNT);

    if (ws.ButtonState.A)
      projectileFired = true;

    if ( ws.IRState.Midpoint.X != 0 &&
         ws.IRState.Midpoint.Y != 0)
    {
      irTheta = ws.IRState.Midpoint.X - 0.5;
      irPhi = ws.IRState.Midpoint.Y - 0.5;
    }

   // Other button and sensor checks
  }
  // Other methods
}
```

In the sample above, the joystick pad is used to turn the ship left, right, up, or down. The infrared (IR) sensor is used to specify which way the captain is looking, assuming a forward-facing 180-degree panoramic window. The “A” button is used to fire a projectile. The “B” button is used to move forward in the pointed direction, which isn’t necessarily the direction that the captain is looking. An instance of the WiiMote class is created in the “Activated” event of the .NET Form.

In the fourth and final phase of the project, lighting and shading were added. In particular, they were required to add the following minimal functionality:

- Place five positional lights in your universe. Provide the capability to enable and disable each one individually. Show these lights right after setting the "LookAt", before showing the figures. Have different colors. At least two of the lights must use the linear and quadratic attenuation terms and at least two must not be attenuated.
- Have a spotlight at the ship's position, pointing in the ship's direction. Make it R = B = G = 1.0. Provide the capability to set the spot cutoff angle and spot exponent. Default to 30 and 0, respectively. Provide the capability to enable and disable it.
- Provide the capability to vary the global ambient, default it to 0.2.
- Have at least one emissive object, two shiny objects, and two non-shiny objects.
- Have at least two objects with smooth shading and two with flat.

It should be noted that besides the project, the students were also required to do a couple of smaller OpenGL programs. One was a "getting started" program that involved rubber-banding some shapes and storing them in a list for display. Another was a fractal-based program.

6. CONCLUSION

Given the author's mathematical background (Ph.D. in mathematics), the thought of significantly reducing the mathematics and eliminating the algorithm coverage was not a pleasant one. The author thought it would lead to a "watering down" of the course, but was willing to accept that the course had to be changed if it were to survive. In the end, the author was still able to preserve a reasonable amount of the mathematics and cover some of the algorithms, and the students were accepting of that. Moreover, the projects were very challenging, but the students were motivated to put in the necessary time, in part due to interest in integrating the Wiimote. The tests and quizzes focused more on concepts and less on mathematics and implementation, but as seen in Section 3, there were still challenging questions involving mathematics. As it might be expected, the evaluations for the course were mostly positive, although a couple of the graduate students who had had an introductory graphics course as undergraduates did indicate they wish we had covered more of the theory.

Below are a couple of student comments that show that even students not mathematically inclined can still be motivated to learn the material if other factors are favorable:

> "Even though I am terrible at calculus and math, the class was still lots of fun since we were learning something that is not covered anywhere else."
>
> "I knew this was going to be an awesome class when Dr. Clifton busted out the WiiMote on the first day. C# might be my favorite language now!"

We plan to continue with this approach when the course is taught again in fall 2011 with the intent of making some minor changes to the project. In particular, we plan to come up with a framework that allows more flexibility in phase three of the project, allowing for implementation of other types of games.

7. REFERENCES

[1] Angel, W., *Interactive Computer Graphics: A Top-Down Approach with OpenGL*, Addison Wesley, 2009, 5th edition, ISBN: 0-321-53586-3

[2] Cummingham, S., Hansmann, W., Laxer, C., and Shi, J., A Report from the Working Group on Computer Graphics in Computer Science, SIGGRAPH/Eurographics Computer Graphics Education Workshop, Zhejiang University, Hangzhou, China, June 2-6, 2004, http://www.siggraph.org/events/symposia/reports/Rep2004CGEworkshop.pdf

[3] Hearn, D. and Baker, M., *Computer Graphics, C Version*, Prentice Hall, 1997, Second Edition, ISBN 0-13-530924-7

[4] Hudson, D., et al., Tao Framework for .NET, http://sourceforge.net/projects/taoframework/files/

[5] Lee, J., Wiimote projects, http://johnnylee.net/projects/wii/

[6] Peek, B., Managed Library for Nintendo's Wiimote, http://blogs.msdn.com/b/coding4fun/archive/2007/03/14/1879033.aspx

[7] Pokorny, C. and Gerald, C., *Computer Graphics: The Principles Behind the Art and Science*, Franklin Beedle & Associates, 1989, ISBN 0-938-66103-5

[8] WiiBrew, Bluetooth Devices that work with Wiimote, http://wiibrew.org/wiki/List_of_Working_Bluetooth_Devices