AC 2007-1048: EDUCATIONAL COMPUTER SCIENCE FUN PROJECTS FOR INTEGRATING MULTIDISCIPLINARY CONCEPTS OF MATHEMATICS, SCIENCE, AND ENGINEERING

Mahmoud Quweider, University of Texas-Brownsville
Dr. M K Quweider is an Associate Professor and chair of the Computer Science/Computer Information Systems at University of Texas at Brownsville/Texas Southmost College. He received his Ph.D. in Engineering Science and an M.S. in Applied Mathematics, M.S. in Engineering Science, and M.S. in Biomedical Engineering all from the University of Toledo, Ohio. After graduation, he worked at several places including Pixera, a digital image processing company in Cupertino, CA, and 3COM, a networking and communication company in Schaumberg, IL. He joined the UTB/TSC in 2000. His areas of interest include Imaging, Visualization and Animation, Web Design and Graphics.

Juan Iglesias, University of Texas-Brownsville
Dr. J R Iglesias is an Assistant Professor in the Computer Science/Computer Information Systems at University of Texas at Brownsville/Texas Southmost College. He received his Ph.D. in Computer Science from New Mexico State University (NMSU), New Mexico, USA, with specialization in Databases, and the B.SC and M.S. in Computer Science from the National Autonomous University of Mexico. He has worked as an Associate Director for the Federal Electoral Institute (IFE), Mexico City, Mexico during the 1997 year. His areas of interest include Databases, Programming Languages, Data mining, and Web Design, and e-Commerce Systems.

Amajd Zaim, University of Texas-Brownsville
Dr. A T Zaim is an Assistant Professor and the director of the VIB (Vision, Intelligence and Bioinformatics) research group at University of Texas at Brownsville/Texas Southmost College. He received his Ph.D. in Biomedical Engineering from the University of Toledo, Ohio and an M.S. in Electrical Engineering, and M.S. in Biomedical Engineering Wright State University, Dayton, Ohio. Having worked on multidisciplinary projects in 3D image-guided surgery early in his PhD years, he developed a special research interest in medical image processing. Dr. Zaim also led and managed two IT firms providing consulting for business and healthcare organizations. He recently joined the computer science department of the UTB and is currently conducting collaborative research with members of the VIB group as well as external healthcare organizations.

Educational Computer Science Fun Projects for Integrating Multidisciplinary Concepts of Mathematics, Science, and Engineering

Abstract

In our continuous efforts to increase recruitment, retention and graduation rates of our, mainly minority, computer science and engineering students, we have recently embarked on an ambitious and comprehensive transformation of a major sector of our Computer Science and Engineering curriculum, the first stage of which is transforming the means by which major goals and objectives of three key courses, Data and Information Structures (COSC-3345), Digital Image Processing (COSC-4333), and Computer Graphics (COSC-4330) are achieved. The goal is to integrate in a rather "fun and games" way basic concepts from mathematics, statistics, signal and image processing, and computer graphics into a real-life game project. The three courses are meshed synergistically through a well thought-out 2-D/3-D gaming project, which is introduced in the junior level course and continues in the senior imaging and graphics courses.

In the new age of IPods, PlayStations, and Xboxes, it is hard to ignore the affinity young students have for 3-D action-based and visually intense games; so rather than villainizing games and ostracizing their use, we aim instead at using that inherent fondness of the games to the students' advantage by relating key computer, engineering, and mathematical concepts to the fundamental way games operate. By adhering to the guidelines and recommendations set forth by the ACME and the Accreditation Board for Engineering and Technology (ABET) Technology Criteria 2000 for the Computer Science and Engineering programs, the CS/CIS department at our university has continually modified and enhanced several facets of its programs to demonstrate that its graduates possess specific mathematics, physics, engineering, and computer science skills (outcomes) by their time of graduation. This paper describes our efforts to incorporate in a rather fun and entertaining ways how to integrate major concepts in the above described fields in an action-based game project which students find exciting and are easily able to relate to. Our new experience showed that a Game-based project quickly attracts students and fosters student communications, teamwork, and the development of analytical capabilities. The paper additionally details the interdisciplinary strategy implemented by the department's faculty in conjunction with other departments in the college of Science, Mathematics, and Technology (SMT) to integrate key concepts in the Mathematics and Physics areas in the game design project.

Introduction

Games are fun! Such a statement is hardly disputed by any one, especially young students at the college level, who are usually seen playing with their nifty and "cool" gadgets at dorms, bus-stops, and cafes. But games are also complex to design and construct, as they incorporate many disciplines especially those in the Science, Mathematics and Engineering areas. The question which then presents itself is: Is there a way to combine the fun aspects of gaming with the engineering and scientific principles they underline at the college level? Recent articles[1-5] have emphasized the role these games can play in "reversing the ongoing decline in computer science enrollment," by appealing to the "ubiquitous interest among teens and twentysomethings," in action-based games.

Additionally, games can present "real-world applications" to science and engineering majors rather than the unrealistic short programs, presented in programming classes for example, with their archaic number-sorting algorithms, and seemingly abstruse formulas; the same can be said about the mathematical functions and equations, or physics laws, which at times can be seen as void of a proper context to justify and reinforce their extreme importance. This paper presents our efforts to bring major concepts in Science, Mathematics, Physics and Computer Science together in a game-centric action-based project. The game consists of many modules, but we, as a first stage effort, specifically target the game modules which relate to the following:

1. Mathematics and Physics
    a. Relation to vector analysis, probability, transformations, integration and differentiation, physics motion equations, exponential and doubly exponential functions used in fogging.
    b. Matrix operations for basic translation, rotation, and scaling.
2. Computer Science
    a. Object Oriented Programming (OOP)
        i. Classes and objects as game components
    b. Data structures used in maintaining players information and statistics, game structure, game resources
    c. Hardware architecture of modern graphic-cards and game platforms
3. Signal and Image Processing
    a. Filtering techniques related to gaming
    b. Image pyramids, interpolation and zooming techniques
    c. Images and bitmaps
4. Computer Graphics
    a. Rendering techniques for basic geometric objects (lines, circles, polygons)
    b. Texture representation

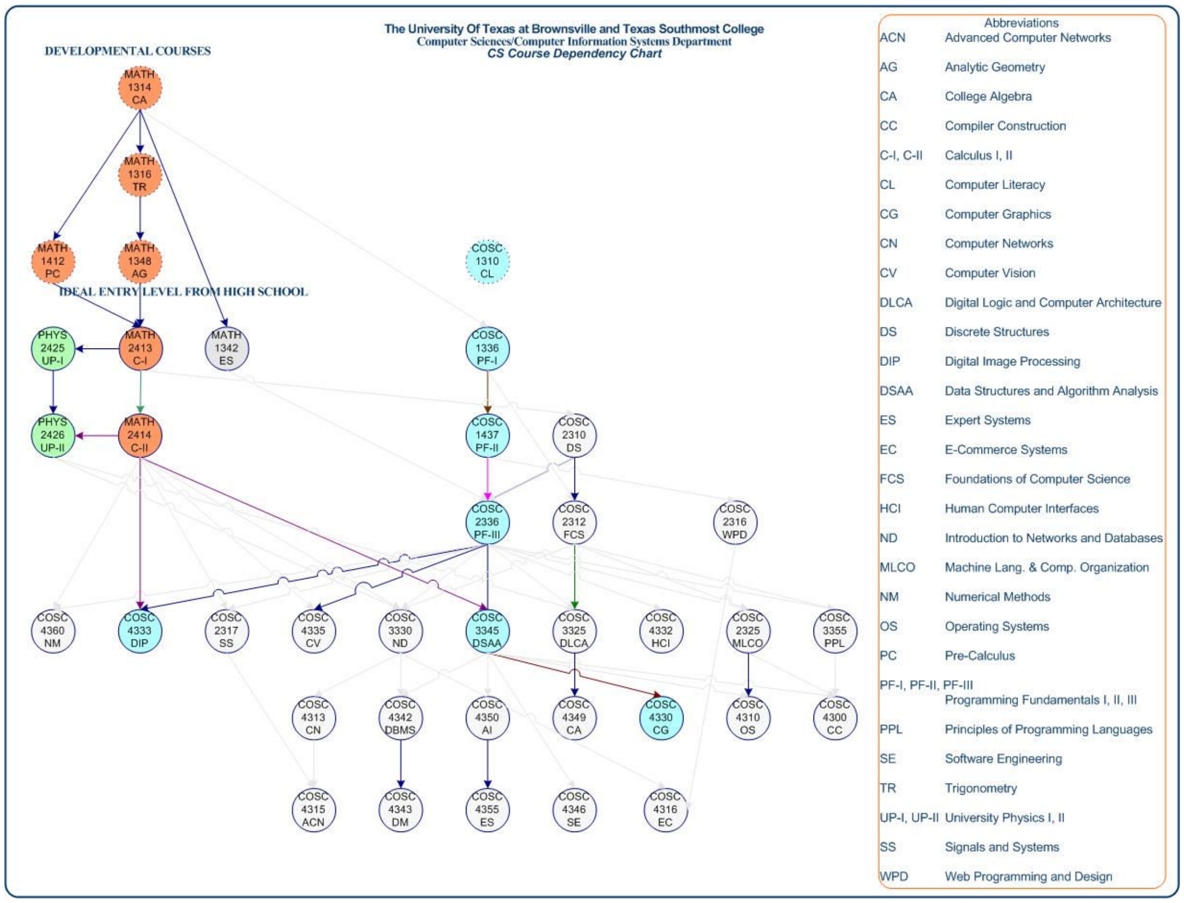
In the following sections we describe the overall structure of the game and the targeted modules as they relate to the game project, and give a suggestion as how to extend them to other design projects.

Game Components

The game requirements are introduced to the students in their sophomore year with preliminary work done on different pieces; however, the final game is actually implemented during the senior year as a capstone project for the Computer Graphics (COSC-4330) course. Our intention, once we have a good pool of completed projects, is to distribute, either in hand or though an on-line portal/website, the best project from the previous offering as an exemplary project to learn from and replicate in terms of best practices. This will serve two purposes: firstly, it will allow the junior student to see how the end product should look like; and secondly, it will allow him to relate the preliminary work done before the capstone project to where it meshes in the final project.

The project requires the students to implement an action-based game which incorporates multidisciplinary concepts from the following modules shown in figure 1. Before taking the Computer Graphics course, the student would have taken the following courses in the proper order, which are also shown in figure 1:

Calc I, II → Physics I, II, → Programming I, II, III, → Data and Information Structures → Digital Image Processing → Computer Graphics.
Course Sequence Recommendation



ACN	Advanced Computer Networks
AG	Analytic Geometry
CA	College Algebra
CC	Compiler Construction
C-I, C-II	Calculus I, II
CL	Computer Literacy
CG	Computer Graphics
CN	Computer Networks
CV	Computer Vision
DLCA	Digital Logic and Computer Architecture
DS	Discrete Structures
DIP	Digital Image Processing
DSAA	Data Structures and Algorithm Analysis
ES	Expert Systems
EC	E-Commerce Systems
FCS	Foundations of Computer Science
HCI	Human Computer Interfaces
ND	Introduction to Networks and Databases
MLCO	Machine Lang. & Comp. Organization
NM	Numerical Methods
OS	Operating Systems
PC	Pre-Calculus
PF-I, PF-II, PF-III	Programming Fundamentals I, II, III
PPL	Principles of Programming Languages
SE	Software Engineering
TR	Trigonometry
UP-I, UP-II	University Physics I, II
SS	Signals and Systems
WPD	Web Programming and Design

Figure 1. Course Sequence Recommendation

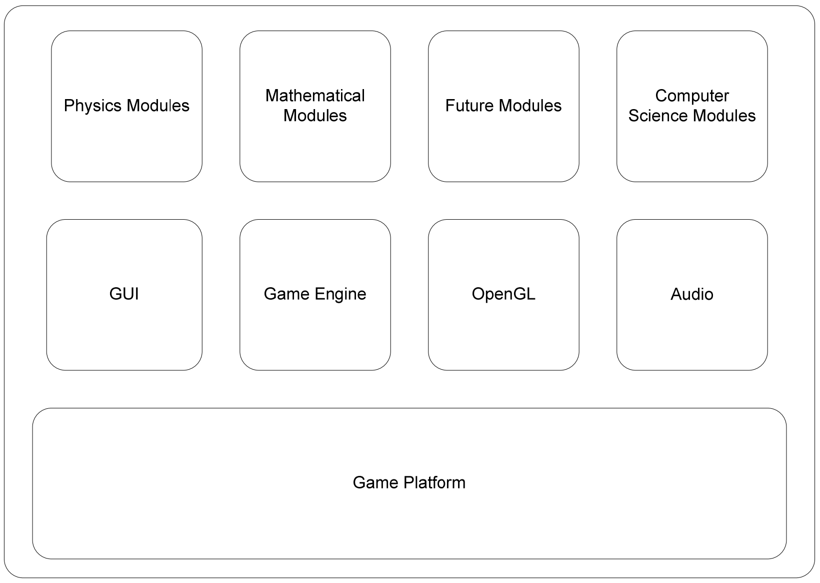


Figure 2. Game Overall Diagram

Figure 2 shows a diagram of the major components of the game as they relate to our targeted areas. In the following we discuss these modules and present related material implemented in the final game project.

1. Mathematics & Physics Modules

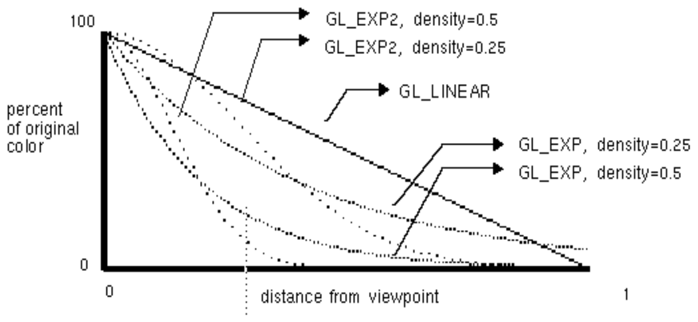
As mathematics represents a major part of any game, many mathematical concepts were successfully integrated into the final game design.

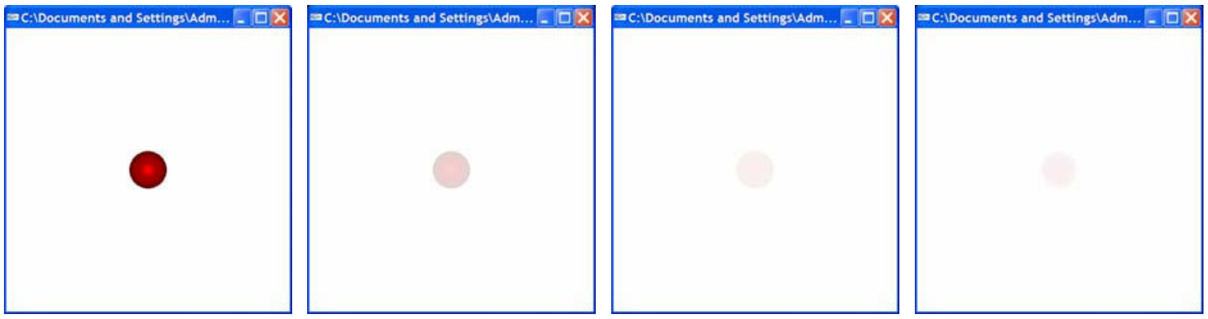
1.1. Mathematical Functions

Many basic formulas and functions are put in perspective in the context of gaming. We discuss two examples among many others that exist in almost every stage of the game. The first is collision detection, such as when a plane is hit, is approached as a Euclidean distance measure between two vectors (2-D or 3-D points), which falls within a given distance (the sum of the two planes radii, considered as simple spheres). The algorithmic code for the collision detection can be implemented as follows:

```
Function: Boolean bCollideTest
Parameters: Vector* Plane1, Vector* Plane2, float P1Radius, float P2Radius
{
  // Computer Difference Vector between the two planes
  Vector DistanceVector = Plane1->prPosition - obj2-> Plane2;
  // Find Scalar Distance
  dist =       DistanceVector.x * DistanceVector.x +
               DistanceVector.y * DistanceVector.y +
               DistanceVector.z * DistanceVector.z;
  // Compare distance to the
  minDist = P1Radius + P2Radius;
  return dist <= minDist * minDist;
}
```

Another example where mathematical functions make absolute sense within the game context is the difference between exponential functions, and the drastic effects they have on performance. We use Fogging to demonstrate these differences. Fog is a general term that describes similar forms of atmospheric effects. It can be used to simulate haze, mist, smoke, or pollution. For gaming, Fog is essential in visual-simulation applications, where limited visibility needs to be approximated and when fog is enabled, objects that are farther from the viewpoint begin to fade into the fog color. Mathematical functions are used to control the density of the fog, which determines the rate at which objects fade as the distance increases, as well as the fog's color. As the graph below shows, three models can be used for fogging (GL_EXP2, GL_EXP1, GL_LINEAR) each with drastically different effect on how objects are seen from a distance.




C:\Documents and Settings\Adm...
C:\Documents and Settings\Adm...
C:\Documents and Settings\Adm...
C:\Documents and Settings\Adm...

Fogging with different models, No Fog, GL_EXP2, GL_EXP1, GL_LINEAR, (affecting distance from viewer)

1.2. Motion Equations

Many physical phenomena in gaming have their origin in Newton's motion laws. Therefore, we strived to derive these equations from differential calculus point of view, and then use them in different aspects of the game. For example, a fountain simulation was implemented by some of the students based on these laws. All students were required to use Newton's laws in vector form; the vector-form equation was used to update the player's fighter plane and enemy's planes as well. In the final game design, each plane takes the following form

– $P(t)=P_0 + s(t)\,\mathbf{d}$

Where P_0 is the initial location of the plane, **d** is a direction vector deciding the path of the plane, and s(t) is the speed of the plane.

1.3. Vector Transformations

In the game, a player's fighter plane has the freedom to move paced on the feedback from the input devices (mouse, keyboard, etc.). Basic left and right movements were implemented as vector transformation operations. For generality, we also allowed a plane to go through the following transformation:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = T \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} a_{xx} & a_{xy} & b_x \\ a_{yx} & a_{yy} & b_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Where for Translation, T= $\begin{bmatrix} 1 & 0 & b_x \\ 0 & 1 & b_y \\ 0 & 0 & 1 \end{bmatrix}$, Rotation, T= $\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$, and for Scaling T= $\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$.

Additionally, graphics hardware employs a sequence of coordinate systems; the location of the geometry is expressed through the transformation in each coordinate system in turn, and modified along the way. The movement of geometry through these spaces is considered a pipeline, and is subject to different transformations.

We also use the modeling transformation to position and orient the main objects in the game including the player's fighter plane, the enemy's planes, the bullets, and the surrounding terrains. OpenGl represents these transformations with the following functions:

void glRotate{fd}(TYPE *angle*, TYPE *x*, TYPE *y*, TYPE *z*);
void glTranslate{fd}(TYPE *x*, TYPE *y*, TYPE *z*);
void glScale{fd}(TYPE *x*, TYPE *y*, TYPE *z*);

Where the {} indicates that either an f, for float, or d, for integer, must be supplied; TYPE is the type of variable (i.e. float, integer, etc.).

2. Signal Image Processing Modules

Images can simply be thought of as two-dimensional arrays of positive integers (pixels) representing either the digitized color components or the gray levels. Image processing deals with the acquisition, representation, and processing of these arrays to achieve a specific goal such as removing noise, sharpening an image, etc. Three of the gaming areas that heavily relate to image processing are bit maps and images, zooming and interpolation, and texturing.

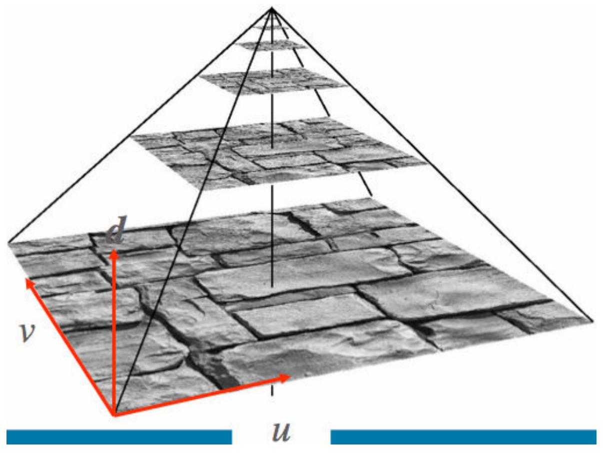
2.1. Bit Maps and Images

Bit images and maps are at the heart of any game as they are used to display the environment, terrains, character, weapons, and special effects. To display images at different places in the buffer, OpenGL provides operations for reading, copying and drawing pixels. These commands use the following functions respectively: glReadPixels() - reads a region of the frame buffer into off-screen (processor) memory; glCopyPixels()- copies a region of the frame buffer into another part of the frame buffer; glDrawPixels()- draws a given pixmap into the frame buffer.

In the process of transferring and copying pixels OpenGL provides the capability to magnify, reduce, or flip (reflect) an image. The function glPixelZoom() can be used for these purposes.

2.2. Zooming and Interpolation

OpenGL provides the capability to magnify, reduce or flip (reflect) an image. Of course in image processing, magnifying and zooming are achieved based on different types of interpolation techniques. Gaming provides a perfect context in which these operations are used to display objects at different distances and scales, thus making scenes more realistic. We used the function glPixelZoom()to demonstrate this capability; the parameters, the functions used, are already covered in the image processing class. Another technique related to interpolation and texturing is mipmapping. Mipmapping, shown in the following figure, is a technique that allows us to calculate a texture map at each resolution level. We Pre-calculate how the texture should look at various distances, and then use the appropriate texture at each distance. Each mipmap (each image) represents a level of depth (LOD). Mipmapping, of course, is known as image pyramid in image processing, and is used for different purposes including image compression and vision applications. The gaming project presents another solid example where a pyramid is used to create a realistic application.



2.3. Texturing

Texture is simply the appearance and feel of a surface. Usually an image is used to define those characteristics of the surface. In gaming, many objects are dressed with a texture in the form of an image; texturing an object involves mapping (pasting) 2D images onto geometrical objects in order to enhance the realism of a scene and making drawings a bit more interesting. Surface texture can be made from Bitmaps, Digital images, Clip arts, as well as from computed functions. While implementing the game, students had the choice to create new art work or use freely existing ones for their fighter planes, bombs, enemy characters, and trains. As they are already familiar with image formats they were able to easily load the texture images and bind them to the appropriate objects. The following set of partial functions taken from one of the student's project show the basic steps used in this process, which are loading the image, generating the texture using the glGenTextures() function, binding the texture to an object using the glBindTexture()function, and building the mipmap structure of textures.

```
void LoadTextures(…)
{
     glGenTextures((sizeof(resource_id)/2), &texture[0]);
     for (int i=0; i<sizeof(resource_id)/2; i++) {
          LoadBMPTexture(NULL, resource_id[i], i);
     }
}

bool LoadBMPTexture(char* file, int resource_id, int num_texture)
{
     LoadImage(…);
     glPixelStorei(…);
     glBindTexture(…);
     glTexParameteri(…);
     gluBuild2DMipmaps(...);
}

void Fighter::DrawFighter(float fadelevel)
{
     glPushMatrix();
          glColor4f (1.0f, 1.0f, 1.0f, fadelevel);
          glBindTexture (GL_TEXTURE_2D, texture[2]);
          int size=3;
          int direction=0; // may try adding in the future
          glBegin (GL_QUADS);
               glTexCoord2f (0,1);   glVertex2f (X - size, Y + size);
               glTexCoord2f (0,0);   glVertex2f (X - size, Y - size);
               glTexCoord2f (1,0);   glVertex2f (X + size, Y - size);
               glTexCoord2f (1,1);   glVertex2f (X + size, Y + size);
          glEnd ();
     glPopMatrix();
}
```

3. Computer Science

We felt that the major emphasis in the area of computer science should be on relating many of hardware issues, data structures, and OOP concepts, taught in the lower level courses, to real world applications as they present themselves in a gaming project.

3.1. Object Oriented programming (OOP)

OOP deals with the creation of classes, or templates, from which we can create many objects. A class is similar to a blueprint of a car, and the objects are the actual cars built from this blueprint. OOP is easy to deal with as it resembles in many aspects the way we think in real life. For our game, students were required to construct an object oriented model of the game in which they treated players, enemies, resources, and terrains as objects. As an example, we show the major parts of the class definition of a fighter as implemented by one of the students:

```
class Fighter
{
private:

      void DrawFighter(float fadelevel);
      void DrawFaded(GLfloat x,GLfloat fadelevel);
      void DrawExplosion(GLfloat BlastRadius);
      void Draw();
      void Draw(GLfloat x);
      void Draw(GLfloat x, GLfloat y);

      GLfloat X;
      GLfloat Y;

      int NewFighterDelay;
      int DefaultNewFighterDelay;
      GLfloat FighterFadeIndex;
      // Flags
      bool Explodef;
      bool Fighterf;
      bool GameOver;

      GLfloat ExplosionSize;
      GLfloat CurrentExplosionRadius;
      GLint ShootCounter;

      // Defaults - Defined in Constructor
      GLfloat DefaultExplosionSize;
      GLfloat DefaultX;
      GLfloat DefaultY;

      GLint ProjectileMax;

      /////////////////////////////////////////////
      //    Projectile Nested Class;
      class Projectile
      {
      private:
            GLfloat X;
            GLfloat Y;
            GLfloat VanishPoint;
            GLfloat StartPoint;
            GLfloat StartSpeed;
            GLfloat VanishSpeed;
```

```
		bool Projectilef;

	public:
		Projectile();
		void New(GLfloat x, GLfloat y);
		void UpdatePosition();
		void Draw();
		void Explode();
		void Remove();
		void Loop();
		bool Active();
		float ReturnX();
		float ReturnY();
	};
	//	End Nested Class
	Projectile projectile[MAX_PROJECTILES];

public:

	Fighter();
	void New();
	void Remove();
	void RemoveAll();
	bool Active();
	…
	void Loop();
	void Shoot();
	void Explode();
	bool CheckCollision(GLfloat x, GLfloat y, GLfloat r);
	bool Ready();
	void SetGameOver(bool flag);
	void Shoot2();
};
```

What is amazing is the ease with which students were able to construct the classes, and come up with logically sound members and functions for each class; in probing about this, we found that this ease was mainly based on their real-life experience with actual game playing and game settings. The classes illustrate many of the solid OOP principles including modularity, data encapsulation, inheritance, and function overriding just to mention a few.

3.2. Data Structures

Data structures are one of the basic tools programmers use to create solid and efficient software. Gaming is full of areas in which data structures are used efficiently whether for fast performance or for efficient storage. While many specialized data structures exist for graphics and professional gaming, we focused, instead, on usual and basic structures introduced in the lower level classes. Here are some examples where data structures were efficiently used and explained within the game context:

- 1-D arrays: one-dimensional arrays were used to store basic textures, or number of enemy planes, or light information. A typical definition would look like:
  GLuint texture[No_Textures]; or
  Enemy minion[MAX_MINIONS];
  float lightPos[] = {x,y,z,1};
  float lightDirection[] = {0.0,0.0,-1.0,1.0};
- 2-D arrays: two-dimensional arrays were used to store images, Bomb data, A typical definition would look like:
  GLubyte image [256][256]
  float BombData[MAX_BOMBS][2];
- Structures: structures were used in places were a class would be considered less efficient (as structures can be considered specialized classes). For example some students modeled bomb data using a structure. A typical definition would look like:

```
struct bullet
{
          float x, y;           // center of bullet
          float velocity;       // velocity
          bool exist;           // exist detection 0/1
};
```

3.3. Hardware architecture of modern graphic-cards and game platforms

One of the most important factors for a successful game is the hardware and the platform on which it is runs. As students were running there games at the lab, on their lab top, or in the lecture room, it was hard not to notice the fervor and enthusiasm with which they were discussing, proudly, the hardware pieces relating to the game. Of particular importance were the following three parts:

- Processors
- Graphic Cards,
- Memory

Although hardware is not officially part of any of the targeted course, we discovered that students were quickly tutoring each other, and at many times the instructor, to the latest technologies in each of the above categories. Whether it is the latest features form AMD or Intel processors, or the newest graphics and sound cards from NVIDIA or ATT, or the latest DDR memory in the market, a game seems just the perfect topic to stir interest in hardware. Of course, in the process, we found our students brushing up on their computer architecture or machine language knowledge to gain better understanding of their game performance; for example, how double buffering, texture support, or shading and lighting routine are supported at the hardware level.

4. Integration Into Design Projects:

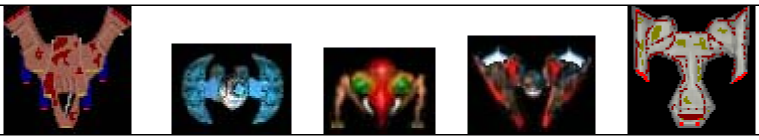
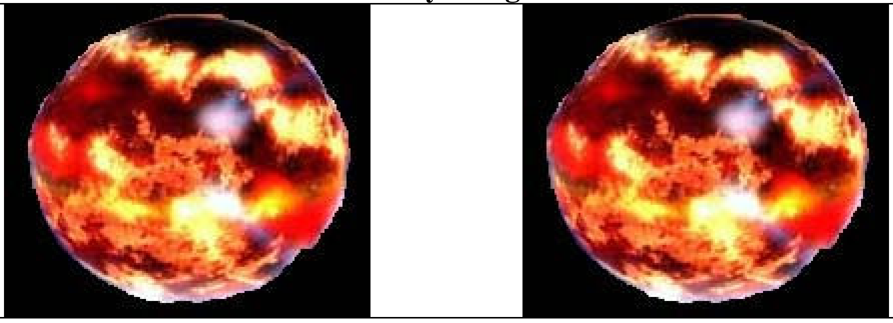
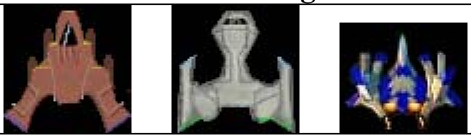
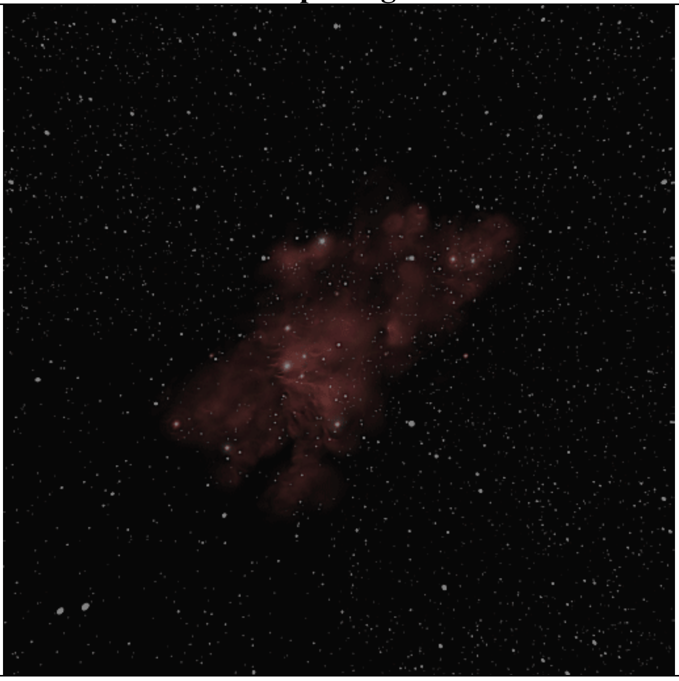
We believe it is fairly simple for other disciplines to follow a similar path in integrating key concepts of mathematics, science and engineering into similar entertaining projects. Our experience is reproducible not only for games but also for other projects that involve simulation or visualization. The key is early coordination within and across departments. Within the department one or more senior projects can be created every cycle of 2 to 4 years, and given to juniors, who will then execute them through their senior year. A set of targeted courses are chosen to emphasize the aspects of the senior project though programming, simulation, and homework problems. Across-department coordination is also vital. The mathematics and physics departments in particular can be involved early by working preliminary examples from the senior project through their teachings and assignments. A department-specific set of targeted courses can be chosen based on their instructional goals and how they can mesh synergistically in a senior project. As the table for our targeted courses show, a game (or a simulation/visualization) project easily attains many of the instructional goals for the courses. Other departments can decide on a project in a similar way.

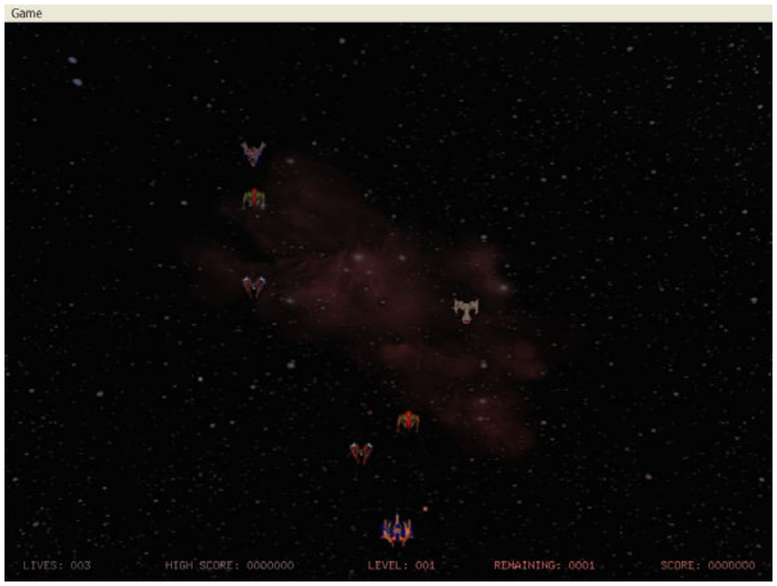
Selected courses Instructional Goals

Course #	**Course Name**	**Students should be able to demonstrate knowledge of information structures.**	**Students should be able to understand data representation and the transformation of data.**	**Students should be able to understand the relationship between hardware and software.**	**Students should be able to apply their understanding of software and hardware structures in scientific or industrial applications.**	**How the requirements are met and assessed.**
COSC 3345	Data and Information Structures	X	x		X	Programming assignments, quizzes, exams, & comprehensive final exam
COSC4330	Computer Graphics	X	X	X	X	Programming assignments, quizzes, exams, & comprehensive final exam
COSC 4333	Digital Image Processing	X	X		X	Programming assignments, quizzes, exams, comprehensive final exam, & several projects

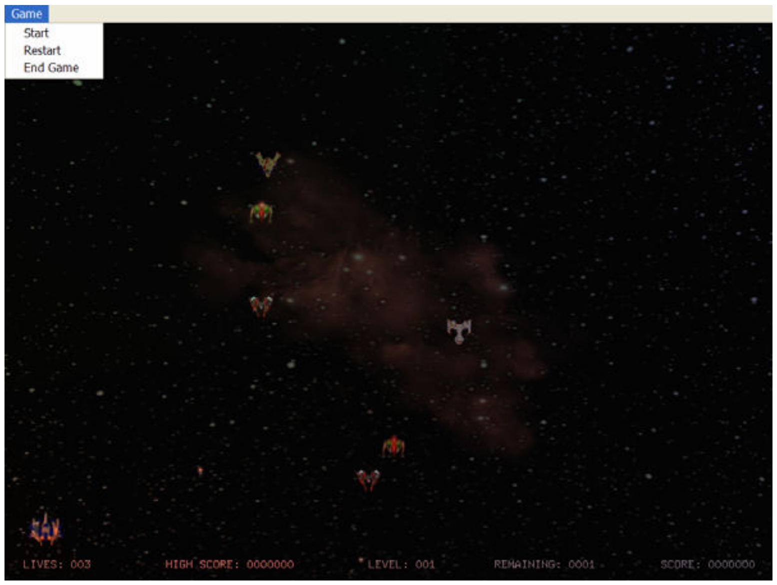
Examples and Screen Shots

The complete code for one game designed by Mark Haven will be posted on the department website. Following are images and screen shots from the game while in action.

Images Used for Game Construction (Mark Haven)

Enemy Images

Fireball Images

Ship Images

Space Image



Game Start Screen Shot



Human Computer Interaction (GUI) Options for game

Students Testimonials

As we present the students testimonials, either as they were ascertained orally or during the end of class evaluation, it is imperative to emphasize that our experience showed that this project did not compromise any of the goals and objectives of our courses, while at the same time gaining instant popularity among them.

The Computer Graphics class in which the gaming project was designed and implemented was offered in fall of 2005, and has an enrollment of 11 students. The Data structures class was offered every semester, except summers, and the image processing class was offered one semester before the Computer graphics was offered.

Statistics for COSC-43330, Computer Graphics Fall-2005				
No. Of Students	11			
Students overall Grading of course	A 9	B 2	C 0	D 0
Comments from students	“I love to have a career in gaming”			
	“I learned more than in any other class”			
	“I really liked this class! It has put everything I learned in focus”			
	“The instructor demonstrated the course very efficiently, and made it very interesting and practical”			
	“Great Course”			
	“I think that there should be an additional course only for the GUI programming in Windows; It is very interesting.			
	“We really need a whole track in Game programming; I wouldn’t mind taking 2 or more additional course in that filed”			

Conclusion

This paper detailed an innovative way to mesh in a synergetic way major concepts from Mathematics, Physics, Computer Science, and Engineering in an action-based gaming project. Many modules of the game were discussed and related to a set of targeted courses, and the actual game design was done in a senior level Computer Graphics course. The experience proved extremely rewarding to the students as well as to the instructors while at the same time insuring that the goals and objectives from these courses are attained by the students. Such an endeavor has proven to be a major one, and while we achieved substantial milestones, we stress that there is a myriad of ways to enhance on the project. For example, several concepts from others courses can be easily integrated into the game. From Software Engineering, to Networking, to Artificial Intelligence, to Databases, and Operating systems, we believe that different modules can easily interface with the game. That is a goal we have set to ourselves, and we will report on our efforts in a future conference. The importance of the realistic and fun factors in these projects are hard to oversee, and they can be prove to be a major factor in retaining students in the Engineering and Science fields, which have been suffering an alarming decline in enrollment. Additional benefits from such a project include improving analytical skills, team work, and better communication among students and instructors.

Acknowledgements

Many of the students in the computer graphics class were partially funded by the CS/CIS department through employment as Lab Assistants. The authors would like also to thank all the students who participated in this innovative project, and provided invaluable feedback, most of which is currently being incorporated in the current offering of the course.

References:

[1]. Anonymous, "More than Fun and Games: New Computer Science Courses Attract Students with Educational Games," http://www.ccnmag.com/news.php?id=3720.

[2]. Claypool, K., and Claypool, M., "Teaching Software Engineering through Game Design," Annual Joint Conference Integrating Technology into Computer Science Education, ITiCSE, Costa De Caparica, Portugal, pp. 123-127, 2005.

[3]. Chamillard, A. T., "Introductory Game Creation: No Programming Required," Technical Symposium on Computer Science Education, Proceedings of the 37th SIGCSE technical symposium on Computer science education, pp: 515-519, 2006.

[4]. Bishop, L., Eberly, D., Whitted, T., Finch, M., and Shantz, M. "Designing a PC Game Engine," IEEE Computer Graphics and Applications, V. 18 (1), pp. 46-53, 1998.

[5]. Schaefer, S. and Warren, J. "Teaching Computer Game Design and Construction," Computer-Aided Design V. 36 (14), pp.1501-1510, 2004.