C S 81A: 3-D GRAPHICS PROGRAMMING

Foothill College Course Outline of Record

Heading	Value
Effective Term:	Summer 2021
Units:	4.5
Hours:	4 lecture, 2 laboratory per week (72 total per quarter)
Advisory:	One of the following: C S 1B, 2B, 20A, 21B.
Degree & Credit Status:	Degree-Applicable Credit Course
Foothill GE:	Non-GE
Transferable:	CSU
Grade Type:	Letter Grade (Request for Pass/No Pass)
Repeatability:	Not Repeatable

Student Learning Outcomes

- A successful student will be able to write code using a specific 3D API such as OpenGL that generates 3-D images and motion. Aspects of the API that will be mastered include setting up the configuration space, specifying the projection, camera positions and lighting parameters, and attaching material properties to the scene members.
- A successful student will apply the mathematical tools of matrices, normal vectors and linear transformations to the design of graphics programs.

Description

Introduction to 3-D graphics programming using OpenGL, intended for anyone interested in gaining 3-D expertise for games, scientific visualization, desktop and mobile apps. Coding topics include a systematic study of the OpenGL API in conjunction with any of these programming languages: Java, C++, C# or Objective C (student's choice). Concept topics include viewports, graphics primitives, 3-D motion matrices, normal vectors, shaders, fragment and pixel buffers, light simulation, polygons, virtual cameras, image pipelines, texture mapping and alpha blending.

Course Objectives

The student will be able to:
A. Install and access the OpenGL API in an integrated development environment (IDE).
B. Create an OpenGL graphics context within an operating system's native user interface (UI).
C. Describe distinct roles of the graphics initialization and drawing operations.
D. Define the meaning of glBegin()/glEnd() and how they are used to define points, lines and polygons.
E. Write programs that use normal vectors and orthographic projections to simulate naturally lit 3-D scenes.
F. Use frame buffers to effect real-time 3-D animation.
G. Incorporate vertex arrays into programs.
H. Define buffer objects and give examples of their use.
I. Explain the difference between viewing, modeling and motion transformations.
J. Produce programs that use matrix stacks to control complex, hierarchical motion.
K. Incorporate color and alpha blending into programs to generate transparency effects.
L. Control scene lighting through camera position, material parameters, spotlights, and ambient parameters.
M. Use OOP and class inheritance to design flexible objects, materials, lighting and animation structures.
N. Write programs that incorporate texture mapping to simulate complex surfaces.
O. Describe the OpenGL shading language and shaders.
P. Download and install third party font libraries for OpenGL.

Course Content

A. The OpenGL Development Platform
1. OpenGL and C++
2. OpenGL and C#
3. OpenGL and Java
4. OpenGL and Objective C
B. OpenGL Windows
1. The OpenGL graphics context
2. Opening and closing graphics windows
3. Threads and the graphics context
C. The Rendering Lifecycle
1. Initializing the pixel formats, clear colors and depth buffers
2. Making the graphics context current
3. Drawing preparation by loading identity matrices, positioning the camera and drawing simple polygons
4. Deleting the graphics context
D. Graphics Primitives
1. glBegin()/glEnd()
2. Point and line commands
3. Triangle commands
4. Quad commands
5. Polygon commands
E. 3-D Scene Building
1. glVertex()
2. glNormal()
3. Computing normal vectors
4. Camera placement
5. Orthographic vs. normal projections
6. Local vs. non-local cameras
F. Animation
1. Defining framebuffers
2. glSwapBuffers()
3. Making the graphics context current in buffer swapping
G. Vertex Arrays
1. VAOs (vertex array objects)
2. Using buffers to store vertex data
3. Speed considerations
H. Frame Buffer Objects (FBOs)
1. Creating and filling buffers
2. Texture buffer objects
3. Renderbuffer objects
I. Transformations
1. Eye coordinates
2. Projection transformations
3. Viewport transformations
4. The Modelview matrix
5. Rotation, scaling and translation
J. Complex Motion

1. 4x4 matrix stacks
2. Pushing and popping matrices
3. Data structures for hierarchical motion
4. Designing class inheritance for graphical data structures
K. Color
1. RGBA color values
2. Alpha blending
3. glBlendFunc() and its parameters
4. Enabling and disabling blending
L. Lighting
1. Camera Position
2. Diffuse lighting
3. Specular lighting
4. Ambient lighting
5. Spots
6. Materials
M. OOP in Graphics Programming
1. Defining material and scene-object classes
2. Defining classes for categories of scene objects
3. Expressing hierarchy through object-members
4. Expressing variations of scene objects through inheritance
N. Texture Mapping
1. Using GL_TEXTURE
2. Loading textures
3. Enabling and disabling textures
4. Interaction of textures and lighting models
O. Shaders
1. The graphics pipeline
2. Vertex shaders
3. Graphics shaders
4. Stock shaders
5. Geometry shaders
6. Fragment shaders
7. Uniform buffer objects
P. Fonts
1. Third party font libraries
2. Configuring and using fonts

Lab Content

A. Configuring an OpenGL Development Platform
1. Download the OpenGL libraries if they are not built-in to your platform's IDE
2. Download any support GUI toolkits required for a GUI-based OpenGL program
3. Write and debug a a modular 2-D shape program that lives within your platform's native UI using a module init() and draw() paradigm
B. Exploring Graphics Primitives
1. Write a program that demonstrates at least six of the graphics primitives managed by glBegin() and glEnd()
2. Make the program interactive, using a simple menu with drop-downs, sliders, buttons, etc.
C. Creating a Simple 3-D Scene
1. Use the glVertex() and glNormal() to produce some simple objects (cube, polyhedron, etc.) that have fixed color or material-based surfaces
2. Experiment with camera position to get differing views
3. Use your platform's UI model to create an interactive program
4. Use glSwapbuffers() to make motion appear smooth and free of flicker artifacts
D. Using Vertex Arrays and Frame Buffer Objects
1. Build object data using vertex arrays and/or FBOs
2. Demonstrate the data structure by a program that allows the user to manipulate the objects within the scene
E. Controlling Camera Position and Complex Object Motion
1. Create classes that reflect hierarchical objects with multiple joints and appendages
2. Use the Modelview stack to control the various parts of the object, independently, without losing the integrity of the object as a whole
3. Let the user control motion and also demonstrate various speeds
4. Separate camera and object motion in your program, so that the user can control each one, separately
F. Controlling Color and Lighting
1. Use OOP to create material classes and assign materials to the players in your scene
2. Assign different characteristics to your scene's lights and ambient illumination
3. Let the user control the lighting characteristics while the objects are in motion
G. Incorporating Texture Mapping
1. Load two or more textures into memory, and assign them to some of your objects
2. Display the scene with animation to view the textures at all angles as they are rotated
3. Incorporate transparency and alpha blending
H. Using Shaders in Your Program
1. Define shaders in your program
2. Make use of at least two of the following: fragment shaders, vertex shaders, geometry shaders, or uniform buffer objects
I. Adding Fonts to Your Display
1. Download and install some third party font utility for OpenGL
2. Create a scene which incorporates fonts
3. Let some fonts be 2-D labels for the window, and others be 3-D objects that are lit by the scene and move as the camera moves

Special Facilities and/or Equipment

A. Access to a computer laboratory with language compilers, and OpenGL API plugins.
B. A website or course management system with an assignment posting component (through which all lab assignments are to be submitted) and a forum component (where students can discuss course material and receive help from the instructor). This applies to all sections, including on-campus (i.e., face-to-face) offerings.
C. When taught via Foothill Global Access on the Internet, the college will provide a fully functional and maintained course management system through which the instructor and students can interact.
D. When taught via Foothill Global Access on the Internet, students must have currently existing email accounts and ongoing access to computers with internet capabilities.

Method(s) of Evaluation

Tests and quizzes
Written laboratory assignments which include source code, sample runs and documentation
Final examination

Method(s) of Instruction

Lectures which include motivation for syntax and use of OpenGL API
Online labs (for all sections, including those meeting face-to-face/on-campus), consisting of:
1. A programming assignment webpage located on a college-hosted course management system or other department-approved internet

environment. Here, the students will review the specification of each programming assignment and submit their completed lab work
2. A discussion webpage located on a college-hosted course management system or other department-approved internet environment. Here, students can request assistance from the instructor and interact publicly with other class members
Detailed review of programming assignments which includes model solutions and specific comments on the student submissions
In-person or online discussion which engages students and instructor in an ongoing dialog pertaining to all aspects of designing, implementing and analyzing programs
When course is taught fully online:
1. Instructor-authored lecture materials, handouts, syllabus, assignments, tests, and other relevant course material will be delivered through a college-hosted course management system or other department-approved internet environment
2. Additional instructional guidelines for this course are listed in the attached addendum of CS department online practices

Representative Text(s) and Other Materials

Wright, Richard, Jr., et al.. OpenGL SuperBible, 5th ed.. 2011.

Shreiner, Dave. OpenGL Programming Guide, 7th ed.. 2009.

Sellers, Graham, et al.. OpenGL Super Bible, 7th ed.. 2015.

These texts are still current and considered to be well suited for instruction in this course.

Types and/or Examples of Required Reading, Writing, and Outside of Class Assignments

A. Reading
1. Textbook assigned reading averaging 30 pages per week.
2. Reading the supplied handouts and modules averaging 10 pages per week.
3. Reading online resources as directed by instructor though links pertinent to programming.
4. Reading library and reference material directed by instructor through course handouts.
B. Writing
1. Writing technical prose documentation that supports and describes the programs that are submitted for grades.

Discipline(s)

Computer Science