COURSE DESCRIPTION

Dept., Number  CSC 133  Course Title  Object-Oriented Computer Graphics Programming
Semester hours  3  Course Coordinator  John Clevenger
URL (if any):  http://gaia.ecs.csus.edu/~clevengr/

Catalog Description

Introduction to computer graphics and to advanced topics in object-oriented (OO) programming. The OO paradigm is used throughout, utilizing computer graphics as the vehicle for solidifying basic OO concepts, studying the implementation of event-driven systems, and for developing a thorough understanding of advanced OO concepts such as inheritance and polymorphism. Topics include fundamental concepts of object-oriented programming, software design patterns, graphic devices, line and surface drawing, simple 2D and 3D representation, and use of User Interface components. Prerequisites: At least a C- grade in both CSC 130 and CSC 131, and full CSC, CPE, or MATH/CSC major status.

Textbook

David Flanagan, Java Foundation Classes in a Nutshell, O'Reilly, 1999.

References


Course Goals

After completing this course, students should be able to:

1. Explain the concept of a class hierarchy in object-oriented programming.
2. Give examples of base and derived class construction in an object-oriented programming language.
3. Explain what is meant by 'method overriding' and 'method overloading’, and give examples showing the differences.
4. Describe the similarities and differences between public, private, and protected access.
5. Explain and give examples of the differences between inheritance of interface and implementation.
6. Explain and give examples of multiple inheritance.
7. Explain the concept and effect of static class members.
8. Explain what a virtual method is, and show how to make use of it.
9. Describe and show the use of dynamic binding and polymorphism.
10. Explain and give examples of extended type assignment compatibility, runtime instantiation of objects, and the use run-time type information in an object-oriented program.
11. Demonstrate understanding of critical issues in memory management of objects, including constructors, heap management, destructors, and garbage collection.
12. Give examples of basic graphics-oriented programming objects.
13. List and describe characteristics of raster CRT devices.
14. Explain the concept of RGB color space and its use in a graphics program.
15. Explain the concept of and uses for linear transformations (scaling, rotation, translation), and their matrix representation and manipulation.
16. Explain the concepts of 2D world space, windows, viewports, and display space; and be able to manipulate these concepts in an object-oriented program.
17. Design and implement a simplified object-oriented graphics definition and display environment, including manipulation of world-to-display space transformations and application of these transformations to simple wire-frame objects.
18. Implement simple 2D modeling hierarchies.
19. Explain the mathematics underlying simple 2D Bezier curves, and implement graphics routines to manipulate such curves.
20. Use common object-oriented-based event handling techniques.
21. List, describe, and use common user-interface component objects.
22. List, explain, and implement common software design patterns such as observer/observable, mediator, visitor, and factory.

**Prerequisites by Topic**

*Thorough understanding of:*
- Programming in a high-level language, such as C++, Java, Eiffel, Ada or Scheme, as evidenced by at least three semesters programming experience.
- Implementation of linear lists including stacks and queues, and binary trees.

*Basic understanding of:*
- "Object-based" programming, including class definitions, object instantiation, method invocation, and public vs. private fields.
- Use of recursion in a program.
- Pre-calculus math including trigonometric functions, Cartesian coordinates, points, lines and planes in space, coordinate transformations, conics, algebraic relations and functions, polynomial equations, inequalities, and matrix operations.
Major Topics Covered in the Course

1. Review of Java Basics: compilation and execution, classes, files and methods, references, garbage collection, arrays, I/O, vectors, wrapper classes, strings, parameter passing, exceptions, similarities and differences from C++.
2. OO Concepts: class associations, UML class diagrams, composition and aggregation; abstraction and encapsulation; accessors (selectors and mutators); invariants; simple inheritance, overloading vs. overriding.
3. Graphical User Interfaces: frameworks, AWT and Swing, containers and components (frames, panels, buttons, labels, combo-boxes), window closing, layout managers.
4. Event-driven Programming: design patterns, Java Events, Listeners, and Interfaces; Action, Window, Item and Keyboard Listeners; menus.
7. Inheritance: extension vs. specialization vs. specification; abstract classes and methods.
8. Polymorphism: runtime typing, late binding, Java vs. C++; upcasting and downcasting; Multiple Inheritance - specification vs. implementation, Java Interfaces.
10. Graphics Operations: local, world & screen coordinate systems; windows, viewports and the viewing transformation, zoom and pan operations; object selection; screen-to-window (inverse) mapping, clipping.
11. Design Patterns: creational vs. structural vs. behavioral; Observer, Iterator, Template, Composite, Decorator, Factory, Adapter, Strategy.

Outcomes

Thorough understanding of:
- The concept of class hierarchy in object-oriented programming, including relevant characteristics such as abstraction, encapsulation and information hiding, types of inheritance, method overloading and overriding, dynamic binding and polymorphism.
- The use of common GUI component objects in the design and implementation of graphical user interfaces.
- Implementation of common event-driven GUI programming techniques.
- Implementation of basic graphics-oriented programming concepts such as object selection and display of graphical objects.

Basic understanding of:
- Type assignment compatibility, runtime instantiation of objects, and the use run-time type information in an object-oriented program.
- Issues in memory management of objects, including constructors, heap management, destructors, and garbage collection.
• Characteristics of raster CRT devices and the RGB color model, and how they relate to programming graphically-based interactive systems.
• 2D linear transformations (scaling, rotation, translation), including their matrix representation and manipulation.
• 2D world space, windows, viewports, and display space and the transformations used to map between them.
• Simple 2D modeling hierarchies and the use of transformations to represent their components.
• 2D Bezier curves, including implementation of routines to draw such curves.
• AWT and Swing as examples of the concept of an application framework.
• Implementation of Java listener interfaces as examples of observer/observable design patterns.

Exposure to:
• A variety of common software design patterns such as, mediator, visitor, strategy and factory
• Simple 2D screen-based animation techniques.

Laboratory Projects

1. Implement a simple class-based program in Java (1 week).
2. Add a simple non-interactive GUI to the program (2 weeks).
3. Add event-handling for buttons, menus, and window operations to the GUI (2 weeks).
4. Extend the GUI to include interactive mouse drawing and keyboard operations (2 weeks).
5. Extend the program to support hierarchical graphical object creation and manipulation (2 weeks).
6. Extend the program to include interactive object selection and highlighting, and simple screen-based 2D animation (2 weeks).
7. Extend the program to include world coordinate system representation, mapping from world to device space, and pan/zoom operations (2 weeks).

Estimated Curriculum Category Content (Semester hours)

<table>
<thead>
<tr>
<th>Area</th>
<th>Core</th>
<th>Advanced</th>
<th>Area</th>
<th>Core</th>
<th>Advanced</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithms</td>
<td>0.5</td>
<td></td>
<td>Data Structures</td>
<td>0.5</td>
<td></td>
</tr>
<tr>
<td>Software Design</td>
<td>0.5</td>
<td>0.5</td>
<td>Prog. Languages</td>
<td>0.5</td>
<td>0.25</td>
</tr>
<tr>
<td>Comp. Arch.</td>
<td>0.25</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Oral and Written Communications

No significant component.
Social and Ethical Issues

No significant component.

Theoretical Content

No significant component.

Problem Analysis

For each laboratory assignment students are required to analyze the requirement specification of the assignment and recognize ambiguities in the specification. They are required to identify various possible alternative resolutions to requirement ambiguities, propose a set of unambiguous requirements, and identify various alternative design and implementation options to meet the requirements. They are then required to document their design and implementation choices, and to provide a complete working implementation of their design.

Solution Design

For each laboratory assignment students are required to analyze the requirement specification of the assignment and recognize ambiguities in the specification. They are required to identify various possible alternative resolutions to requirement ambiguities, propose a set of unambiguous requirements, and identify various alternative design and implementation options to meet the requirements. They are then required to document their design and implementation choices, and to provide a complete working implementation of their design.