Software Specification of the GORT Environment for 3D Modeling

Thoren McDole, Haipin Cua, Chang Huang, Leon Kania, Sergiu Dascalu, Fred Harris

Department of Computer Science
University of Nevada, Reno, NV 89557
Email: {dascalus, fredh}@cs.unr.edu

Abstract

This paper presents the UML specification of the GORT (**G**L **O**bject **R**endering and **T**ransformation) environment for 3D modeling. GORT is an efficient, easy to use CAD-like 3D modeling tool that supports both the creation of scenes using graphical objects and the generation of code that represents the scenes. Planned as open source software, GORT will facilitate the visual design of 3D objects and their environments and will provide the basis for several advanced extensions. GORT's requirements, both functional and non-functional are presented in the paper and excerpts from the UML specification of the GORT software are included. Directions of further research and development are also described.

Keywords: CASE tool, software specifications, UML, GL, 3D modeling.

1 Introduction

GORT (**G**L **O**bject **R**endering and **T**ransformation) is an environment designed to enable programmers to virtually model graphical objects and render graphical scenes created to source code. GORT software includes a comprehensive toolkit that allows the creation of a multitude of objects through the use of a variety of powerful features. The environment will provide users with both the ability to modify graphical objects on varying levels of complexity and the capability of customizing materials and textures of objects. GORT will also provide powerful methods for the creation of complex objects by applying various modeling operations to simple graphical primitives. The planned set of GORT operations include facilities such as object intersection, object addition, and object subtraction.

What sets GORT apart from other existing modeling programs, for instance 3D Studio Max [1] or ProE [2], is that it is designed around the idea of automatic code generation.

In particular, the ability to create source code for scene representation will greatly assist graphics programmers with their understanding and use of a new graphical API. Time consuming tasks such as comprehending subtle nuances involved in setting up normals, viewing volumes, selecting materials, adjusting lighting, and specifying camera angles will be significantly simplified by the ability of observing the visual effects of changes to code in near real time. In addition, GORT generated code could be used in a wide variety of graphics applications in which the developer desires to spend as short as possible time in the consuming tasks of object creation and manipulation [3].

In order to build GORT, a software development process based on a simplified version of the UP (the Unified Process [4]) and using as supporting notation UML (the Unified Modeling Language) [5] has been used. In particular, we have relied on the approach and notation presented by Arlow and Neustadt in [6]. The first version of GORT, currently in its implementation phase, is expected to be ready by mid-May. The inclusion of a number of practical extensions is planned for this year and work on the environment as well on the application of GORT to a variety of case studies is envisaged to continue beyond the timeframe of year 2003. In this paper we present the specification of the GORT software tool, a specification that encompasses both the set of initial requirements for the environment and the principal elements of the software model. We have found that by applying a rigorous, systematic, yet efficient software engineering approach many of the uncertain elements of the tool have been clarified in a timely manner.

This paper, in its remaining part, is structured as follows: Section 2 provides more details about GORT, Section 3 presents a selection of the more important (both functional and non-functional) requirements for GORT, Section 4 presents the tool's software model in terms of both behavior (use cases and scenarios) and structure (class diagram), Section 5 briefly surveys several directions of

further development, and Section 6 concludes the paper with a summary of our work and a note on the current status of the GORT project.

2 GORT: A Brief Overview

GORT is a virtual 3D modeling environment dedicated to assisting software developers in creating advanced graphical interfaces. There are two main areas of applicability on which we have focused GORT's development. First, it can be used as a practical aid in the education of programmers learning the OpenGL API [7] by allowing them to visually connect the abstract aspects of the API with their visual counterparts. Second, it provides an efficient tool for graphical interface designers, a tool that allows both efficient composition of visual interfaces and generation of 3D user interface code.

GORT has features that are set to readily assist programmers in creating a wide variety of models, from basic to complex. These models, utilizing GORT's ability to output code, can be used to build more complex models or sceneries and can be included as a part of another application. This facility provides a means for a programmer to quickly and easily develop applications capable of modeling complex physical systems. It also allows GORT to be useful in a variety of fields, including chemistry, physics, engineering, robotics, animation, and computer game development.

Modeling simple to complex models is possible through GORT's advanced drawing and editing features. Drawing features include the ability to create simple primitives such as spheres, cones, cylinders, pyramids, boxes, and planes. Editing features include the ability to take the above mentioned primitives and extend them through matrix transformations and object intersection, union, and subtraction.

GORT's GUI consists primarily of four *view ports*: top, front, right, and perspective, that provide the user views from various angles. These, however, are only the default views. A user can customize these views by using other preset angles or by adding cameras inside the scene. GORT has a unique feature in which the user can view the models in the perspective view much like being inside the scene and being able to "walk" around the objects.

One of GORT's main functions is to assist programmers in the understanding of the OpenGL API. It allows the learner to connect abstract ideas such as lights, cameras, and viewing volumes with visual objects such as a light bulb, a camera object, and a box. The learner will be able to view the code generated from a scene and understand the association between graphical elements and the code that produced them.

The ability to generate compileable code from a scene is also a major function of GORT. The native file type for GORT projects is highly extensible. GORT parses this file and produces C and OpenGL code but additional types of output modules may also be developed that could produce C++ or Java code as well as use other graphics API such as DirectX [8].

Through systematic UP and UML-based development of its model, GORT has significant in-built flexibility. Its operational capabilities can be extended with new functions and features without affecting the architecture of its software. This makes possible the incorporation of future add-ons as well as links to external tools. Several extensions to GORT that we are currently considering are discussed in Section 5 of the paper.

3 Requirements

Before starting the specification of GORT's software model, we have defined a series of functional and non-functional requirements that should be satisfied by the first working version of GORT. In the following, using the concise, practical style suggested in [6], the most important requirements for GORT are presented. Due to space limitations, we present here only a simplified version of these requirements.

3.1 Functional Requirements

GORT's most important functional requirements are listed below. Nevertheless, we believe they could provide a fairly complete picture of the tool's capabilities in its current, initial version. For traceability during the software development process each functional requirement has a number and is denoted using the format `<R#>`.

`R1` The software shall provide an interface for basic file management for native GORT file types.

`R1.1` Basic file management shall include "new," "open," "save," and "save as".

`R1.2` The software shall provide a CRC (cyclic redundant check) mechanism for validating file types.

R2 The software shall provide an interface for basic edit capabilities of primitives.
  R2.1 Basic edit capabilities shall include "cut," "copy," "paste," "select," "inverse select," and "delete."
R3 The software shall allow advanced edit capabilities of primitives.
  R3.1 Advanced edit capabilities shall include addition and subtraction of two objects.
  R3.2 Advanced edit capabilities shall also include an option for finding the intersection of two objects.
  R3.3 Advanced edit capabilities shall also include an option for "grouping" and "ungrouping" primitives.
R4 The software shall allow transformations of primitives.
  R4.1 Transformations include "translate," "rotate," "scale," and "skew."
  R4.2 The software shall provide a "picking mechanism."
R5 The software shall keep a history list of the user's action for "undo" and "redo."
R6 The software shall allow "view ports" control.
  R6.1 "View ports" control shall include "zoom in," "zoom out," "pan," and "dragging."
R7 The software shall provide definitions for six primitives.
  R7.1 The six primitives shall be box, sphere, cone, cylinder, pyramid, and plane.
R8 The software shall provide a drawing mechanism.
  R8.1 The software shall allow any of the primitives indicated in R7.1 to be drawn in any of the view ports.
  R8.2 The software shall provide a "rubber banding" mechanism.
R9 The software shall provide a material editor and a mechanism that allows materials to be applied to objects.
R10 The software shall provide an interface to allow the user to manipulate and change primitive default properties.
R11 The software shall provide an "export file" interface and mechanism that converts native GORT file types into C and OpenGL code.

3.2 Non-Functional Requirements

Non-functional requirements for GORT include primarily constraints on the environment's implementation. Following is the list of these requirements, denoted using the format <T#>:

T1 The software shall be written in C and C++.
T2 The software shall utilize GTK+ 2.0 API with GL extensions, OpenGL API, and the Standard Template Library (STL).
T3 The software shall utilize an XML parser.
T4 The native GORT file type shall be in XML format.
T5 The software shall have a graphical user interface.
  T5.1 The graphical user interface shall be divided into five sections: Menu Bar, Tabbed Toolbar, Property Box, View ports Section, and Status Bar.
  T5.2 The Menu Bar shall have the following options: "file," "edit," "modify," "draw," "options," "render," and "help".
  T5.3 The Tabbed Toolbar shall have the following tabs: "objects," "view," "lights," "cameras," "render," and "export".
  T5.4 The View Ports Sections shall be divided into four sections: Front, Left, Top, and Perspective View.
  T5.5 The Property Box shall have forms appear based on the selected object.
  T5.6 The Status Bar shall display warning, actions, and other messages.
T6 The primitives shall have two definitions: parametric and implicit.
T7 The software shall provide an interface to display future expansions for GORT.

4 UML Specification

Based on the above lists of requirements, the UML-model of GORT has been built. In the following we present excerpts from both the behavioral part (use cases and scenarios) and the structural part of GORT (part of the system's larger class diagram).

4.1 Use Cases and Scenarios

Using the UML-notation, the UP-based approach described, and the guidelines presented in [6] we have drawn the use case diagram of GORT (Fig.1), have detailed all use cases (examples are given in Fig. 2 and Fig. 3), and specified a comprehensive set of scenarios (examples are given in Fig. 4 and Fig. 5). Based on these use cases and scenarios, other UML diagrams that describe system behavior and structure have been drawn, including the class diagram which is presented (partially) in Fig. 6.

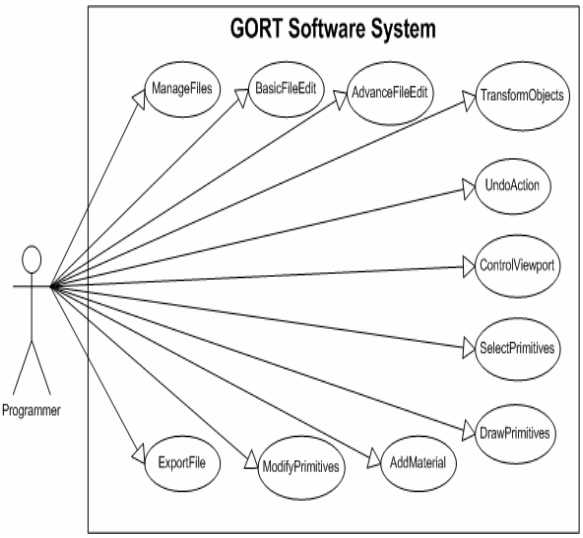

Figure 1 GORT: Use Case Diagram

Use case: ManageFiles
ID: UC1
Actors: Programmer
Flow of Events: 1. The use case starts when the user selects one of the options from the file menu. 2. If user selects "new": 2.1. If the software just started no events will take place. 2.2. If there is a current file open, the software will ask if user wishes to save current opened file, discard it, or to cancel. 3. If user selects "open": 3.1. If there is a current file open, the software will ask if the user wishes to save current opened file, discard it, or to cancel. 3.2. If there is no current opened file, the file selection form pops up. 4. If the user selects "save": 4.1. If the file is being saved for the first time, the file save form will pop up and ask the user for a file name. 4.2. If the file is not being saved for the first time, then the software saves the file without asking for a file name. 5. If the user selects "save as": 5.1. The file save form will pop up and ask the user for a file name.
Alternative Flow 1: 1. The user may exit the program at any given time.
Alternative Flow 2: 1. The user may decide to go initiate this use case by either canceling or moving to another use case.

Figure 2 GORT: The ManageFiles Use Case

Use case: AdvanceEditObject
ID: UC3
Actors: Programmer
Flow of Events: 1. The use case starts when the user selects an advance edit function from the edit menu. 2. If the user selects "union": 2.1. If there are exactly two objects selected then the first object selected inserts the other object into its union list. 2.2. If there are more than two objects or exactly one object selected an error message is issued. 3. If the user selects "subtract": 3.1. If there are exactly two objects selected then the first object selected inserts the other object into its subtract list. 3.2. If there are more than two objects or exactly one object selected an error message is issued. 4. If the user selects "intersect": 4.1. If there are exactly two objects selected then the first object selected inserts the other object into its intersect list. 4.2. If there are more than two objects or exactly one object selected an error message is issued. 5. If the user selects "group": 5.1. If there are more than one selected objects then the first selected object inserts all the others object into its group list. 6. If the user selects "ungroup": 6.1. If there is exactly one object selected then the selected object looks into its group list. 6.1.1. If there are objects in the selected objects group list, the objects in the list gets moved to the GORT draw object list.
Alternative Flow 1: 1. The user may exit the program at any given time.
Alternative Flow 2: 1. The user may decide to re-initiate this use case by either canceling or moving to another use case.

Figure 3 GORT: The AdvanceEditObject Use Case

Scenarios for Use Case: Transform Objects
ID: UC12
Actors: User
Primary Scenario: 1. The use case begins when user selects a transform method from menu "Modify" or clicks any one of transform item form tool bar. 2. The user selects a transform method. 3. The system enters the transform state. 4. The user selects an object for transform. 5. The system displays this object's properties. 6. The user either enters a new data or drags the selected object to a new position or derection. 7. The system displays the modified new object.
Secondary Scenarios: InvalidNewData OutofBounds UndoTransform

Figure 4 GORT: Primary and Secondary Scenarios for the TransformObjects Use Case

Scenarios for Use Case: DrawPrimitive
ID: UC16
Actors: User
Primary Scenario: 1. The use case begins when user selected an object from create menu or clicks object tab from the toolbar. 2. The system displays the default objects 3. The user selects an object. 4. The system starts recording the mouse's movement. 5. The user clicks and holds mouse button at one point and releases the button at another point. 6. The system records the two points and calculates the parameters base on selected object. 7. The system displays this object and its properties. 8. The user modifies the object's properties. 9. The system displays the modified object.
Secondary Scenarios: InvalidMouseClick DataModificaton ViewportModification

Figure 5 GORT: Primary and Secondary Scenarios for the DrawPrimitive Use Case

4.2 Class Diagram

Due to its dimensions, we present in Fig. 6 only a part of GORT's class diagram (analysis level). Also due to its dimensions we have placed it at the end of the paper. Complete details on attributes and, especially, on methods are fleshed out in a larger class diagram (design level), whose dimensions preclude its inclusion in this paper. Nevertheless, we believe the excerpt from the class diagram presented in Fig. 6 provides a good indication on the complexity of GORT's software.

5 Future Extensions

There are numerous possible extensions for this tool that supports the practical and rapid creation of 3D interfaces. At this point in time, we consider extended GORT's capabilities with the following:

- Export to other formats, for instance formats supported by 3D Studio Max, Corel, GIF, jpeg, bitmap;
- Larger collection of objects: geo-sphere, capsule, disc, torus, spirals, N-gons, curves, 2D objects, etc.;
- Advanced representation of primitives including Bezier, NURBS and other spline curves;
- Enhanced facilities for light specification and camera manipulation;
- Animation, for instance in robotics and movies (cartoons);
- Environmental effects: fog, combustions, reflections, etc.;
- Advanced rendering techniques, e.g., ray tracing;
- Multi-threading, for both simultaneous rendering on multiple machines and simultaneous editing by multiple artists.

6 Conclusions

We have presented in this paper a software tool, denoted GORT, intended to facilitate the complex work of 3D user interface designers. What distinguishes GORT is its extensibility, which allows the addition of various facilities and options, and its capability for code generation, which provides an efficient instrument for 3D graphics program representation that can be easily incorporated in a large variety of software products. In order to ensure a sound, rigurous development of the GORT software, a

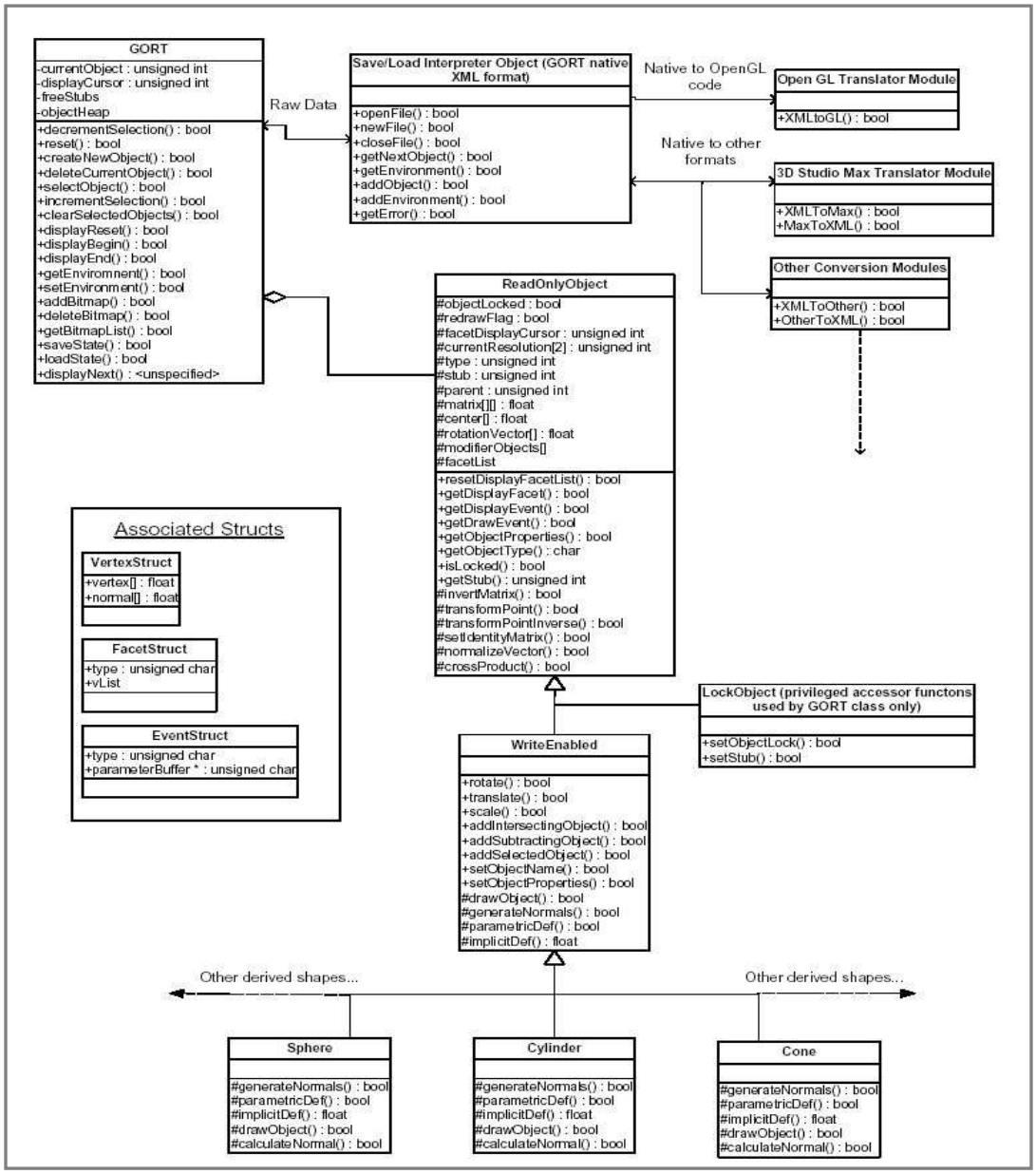

Figure 6 GORT: The Class Diagram (partial)

systematic construction process has been followed, leading to the creation of a software model whose main components at the specification level have been presented in this paper.

A number of possible extensions have also been described in the paper in support of our believe that GORT will prove to be both a very practical tool for the development of graphics-intensive software and a very promising basis for future research.

At this point in time, GORT's UML analysis model has been completed and the tool's implementation is undergoing. The first working version of this environment for 3D modeling is expected to be available in about two months. On a longer time frame, advanced functionality along the lines presented in Section 5 will also be incorporated in GORT.

References

[1] OpenGL Architecture Review Board, *OpenGL Programming Guide*, Third Edition, Addison-Wesley, 2002.

[2] PTC, Home of Pro/Engineer, available at: http://www.ptc.com, accessed March 14, 2003.

[3] Myer, B.A., and Rossen, M.B., "Survey on User Interface Programming," *Proceedings of the Conference on Human Factors in Computer Systems*, May 1992, Monterey, CA, pp. 195-202.
[4] Jacobson, I., Booch, G., and Rumbaugh, J., The Unified Software Development Process, Addison-Wesley, 1999.
[5] OMG's UML Resource Page, available at: http://www.omg.org/uml/, accessed March 14, 2003.
[6] Arlow, J. and Neustadt, I., *UML and the Unified Process: Practical Object-Oriented Analysis & Design*, Addison-Wesley, 2002.
[7] OpenGL – High Performance, available at: http://www.opengl.org, accessed March 14, 2003.
[8] Microsoft DirectX, available at: http://www.microsoft.com/windows/directx/ accessed March 14, 2003.