Lumière: An Infrastructure for Producing 3D Applications in Smalltalk

Fernando Olivero, Michele Lanza, Romain Robbes
REVEAL @ Faculty of Informatics, University of Lugano
{fernando.olivero,michele.lanza,romain.robbes}@usi.ch

***Abstract*—With the goal of developing 3D applications using Smalltalk, we decided to build a lightweight 3D framework named *Lumière* from scratch, because after conducting a brief survey of the available frameworks, we found many of them to be either outdated, abandoned, undocumented or too heavyweight.**

In this paper we present the design and implementation of *Lumière*, an object-oriented framework based on a stage metaphor to display interactive 3D micro-worlds.

I. Introduction

The Smalltalk language and its many dialects include several frameworks for developing 3D applications. Some of them are simple wrappers of low level rendering libraries (such as OpenGL or DirectX), while others are complete frameworks for producing 3D graphics. However, many of them are outdated, unmaintained, undocumented or heavyweight [2].

Therefore we built *Lumière*, the missing 3D framework in Smalltalk. We implemented it using Pharo and OpenGL, with the objective of producing 3D graphics with a simple, modern, lightweight and efficient framework. *Lumière* is an object-oriented framework that provides a layer of abstraction over graphical primitives and low-level rendering. The low-level rendering of the framework is done by OpenGL, an industry standard library for doing high performance graphics. We chose to use OpenGL for efficiency, maintainability, and portability reasons [2]. *Lumière* provides the infrastructure for building 3D applications using intuitive metaphors and high-level concepts such as shapes, lights, cameras and stages. Using *Lumière* a programmer can create complex 3D scenes that we call *micro-worlds*, using performant 3D graphics fully integrated with the underlying environment (Figure 1).

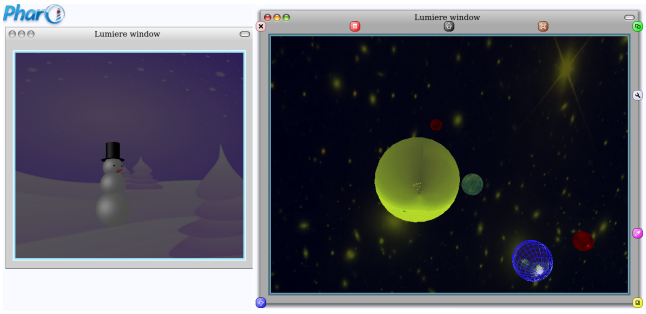

Fig. 1. Lumiere micro-worlds

In the following sections we describe the design and implementation of *Lumière*.

II. The Design of *Lumière*

When designing *Lumière* we chose to build the framework around an intuitive metaphor, promoting ease of usage and understanding, because of the immediate mental model that metaphors provide. We settled on a stage metaphor, where rendering takes place by cameras taking pictures of micro-worlds lit by the lights of the stage. Similar concepts were already present in other Smalltalk frameworks such as Balloon3D and Croquet [3], and also in foreign languages such as Open Scene Graph[1]. In the following we detail each aspect of the design and provide UML diagrams of the exposed APIs.

A. Rendering a Scene

A stage provides a context for taking pictures of a micro-world, therefore a stage contains a micro-world, cameras and lights, and other environmental properties such as ambient lights and fog (see Figure 2).

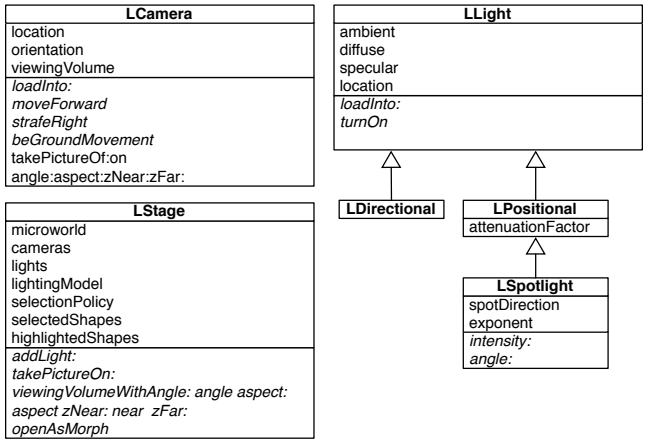

Fig. 2. UML class diagram of Lumiere stages, lights, and cameras

The lights are responsible for illuminating the scenes. There are several types of lights in *Lumière*, similar to the OpenGL lighting model. Each light can be positioned anywhere on stage and contributes to the final appearance of the shapes in a scene.

Cameras take pictures of the microworlds, rendering them on a canvas. They dictate several properties of the final drawing such as the distance, orientation and angle of sight from which the picture is taken. As such, they determine which shapes of the micro-world are visible in the rendered image.

[1] http://www.openscenegraph.org/projects/osg

B. Modeling a micro-world

A *Lumière* micro-world is a 3D world, programmatically modeled and populated by a diverse variety of visual objects called shapes. Shapes are the building blocks of *Lumière* scenes, they have visual properties such as scale, color, materials and textures. A shape can be a primitive or a composite shape that groups several lower-level shapes. *Lumière* provides special-purpose composite shapes with a predefined layout to simplify the positioning of the shapes composing it (see Figure 3).

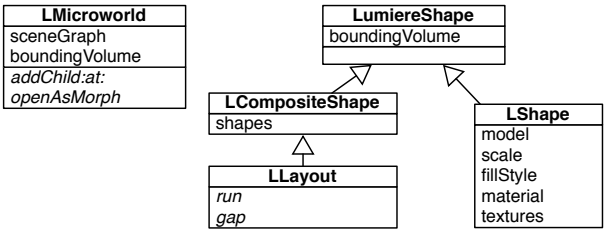

Fig. 3. Lumiere micro-world and shapes

The visual properties, orientation, and location of *Lumière* shapes are described programatically when designing a *Lumière* micro-world. The spatial relationships between shapes are modeled in a micro-world using a scene graph implementation, which is a hierarchical representation of the position of the shapes that populate a given micro-world. A scene graph is a directed graph composed by different types of nodes. There are nodes for loading translations, rotations, scales into a canvas. Other nodes when loaded into a canvas generate a particular figure to be drawn (see Figure 4).

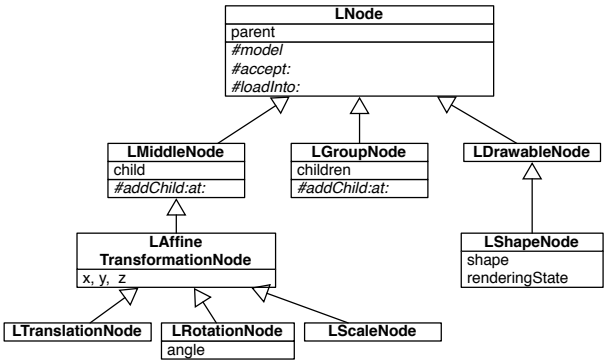

Fig. 4. Lumiere scene graph nodes

When a *Lumière* node is loaded into a canvas, it modifies the state of the latter according to the its type. For example when a scale node is loaded into a canvas it produces a modification in the size of every shape rendered afterwards.

III. The Implementation of *Lumière*

Lumière uses several underlying frameworks, from the base graphics renderer, OpenGL to the Pharo UI framework, Morphic. The architecture of *Lumière* is depicted in Figure 5.

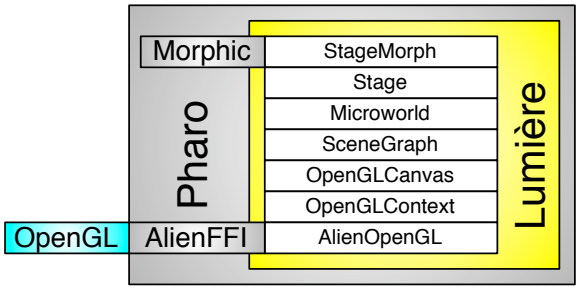

Fig. 5. The architecture of *Lumière*

Lumière uses OpenGL through AlienFFI, a foreign library wrapper framework. It uses a canvas framework to facilitate its interactions with the OpenGL renderer, and is integrated with the Pharo environment trough the Morphic, Omnibrowser and Glamour frameworks.

A. AlienOpenGL

OpenGL is written in C to maximize performance and hence *Lumière* needs to interface with it. We implemented **AlienOpenGL**[2], a framework that uses AlienFFI to access the underlying functionality offered by OpenGL. AlienOpenGL allows one to invoke library functions by sending messages to a singleton instance of ALienOpenGLLibrary.

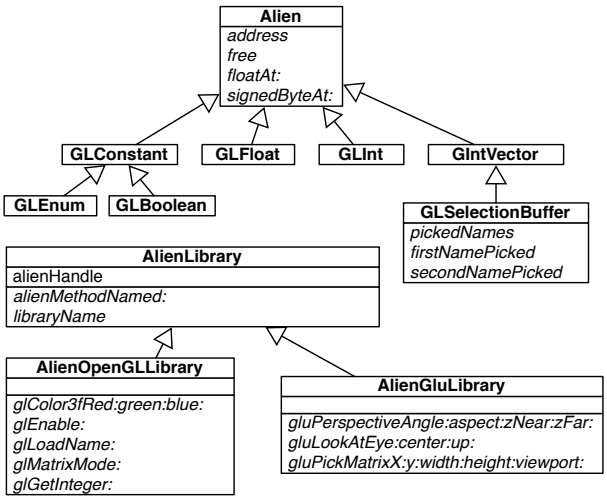

Fig. 6. AlienOpenGL API

See Figure 6 for a UML class diagram of the API of AlienOpenGL.

This framework also provides OpenGL data types reification, access to the Glu library (an extension to the base OpenGL library), and facilities for creating and manipulating an OpenGL drawable surface, managed both by the operating system and AlienOpenGL, where the rendering takes place.

[2]AlienOpenGL is available at http://www.squeaksource.com/AlienOpenGL

B. Canvas and context

When taking pictures of a micro-world with a camera, *Lumière* traverses the scene and loads the primitive shapes onto a canvas, which is an abstraction of a 3D surface, where primitive figures can be rendered. A canvas has a context, an object that reifies an OpenGL context (a particular state of the base rendering system), and acts as an adapter between *Lumière* code and the basic OpenGL protocol of the AlienOpenGL framework. The canvas context allows us to provide some optimizations by maintaining a cache of the state changes, and forwarding only real state changes to the low level AlienOpenGL library.

A canvas forwards the load primitives requests to the appropriate class of geometry object of *Lumière*, for example the message `#loadSphereScaled:` is forwarded to an instance of `LSphere`. This enables to implement different vertex loading mechanisms without modifying any of the canvas load methods, decoupling the request of loading a primitive from the actual low-level implementation. In Figure 7 we display the canvas, context and geometry class diagram.

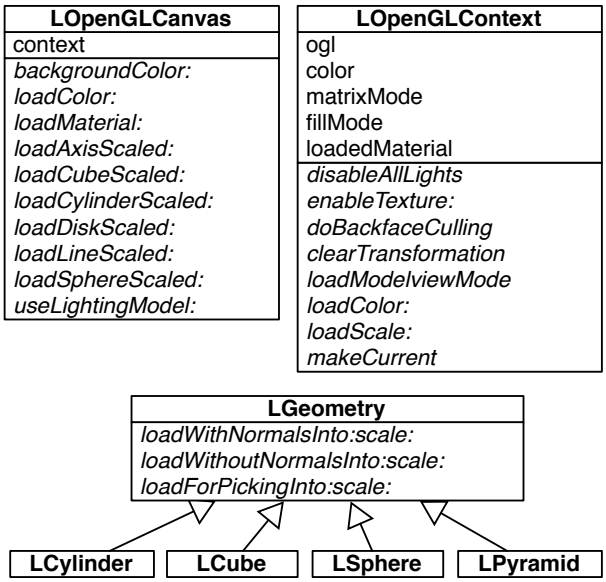

Fig. 7. Canvas, context and geometry

C. Scene graph

The scene graph dictates the final appearance, orientation, and location of the shapes that populate a micro-world. Each node in the path from the root node to a drawable node, contributes to modify the mentioned properties. For example, inserting a scale node after the root node of a micro-world that contains only one drawable node that renders a cube, affects the final size of the cube being rendered.

A scene graph is a convenient structure for accessing all the shapes in a micro-world when performing several tasks. *Lumière* uses the Visitor design pattern [1] to streamline this process. *Lumière* includes three different visitors (see Figure 8) for traversing scene graphs:

1) `LRenderingVisitor`: A rendering visitor traverses the scene and loads the visible nodes into a canvas to display images on the screen.
2) `LScenePickingVisitor`: Picking is the process trough which OpenGL determines which 3D object is under the mouse cursor. OpenGL renders the scene in a hidden buffer for this, so the picking visitor renders a lower detail version of the drawables for optimization purposes.
3) `LBVHCullingVisitor`: Culling is the process of determining which parts of the scene are visible of not. The culling visitor traverses the scene performing intersection tests, pruning the nodes of the shapes that are outside a particular viewing volume.

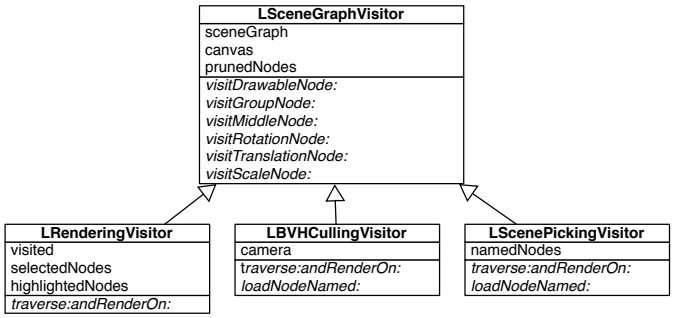

Fig. 8. Lumiere scene visitors

D. Underlying environment integration

Lumière is fully integrated in the Pharo smalltalk environment. A stage can be opened in a window, integrated in browsers and respond to mouse and keyboard events.

Integration with Morphic. *Lumière* stages are integrated into Morphic, the standard UI framework of Pharo. Lumiere shapes, micro-worlds and stages answer the message `#openAsMorph`, opening an instance of a `StageMorph`. A Stage Morph holds an AlienOpenGL draw-able for performing the low level rendering, displaying pictures of the micro-worlds taken by the cameras of the stages.

Integration with Omnibrowser. Stage morphs can be inserted into Omnibrowser, the standard window environment of Pharo. A stage morph answers the messages `#addWindow` and `#removeWindow`, for adding or removing the window decorating it.

Integration with Glamour. *Lumière* stages can also be integrated into Glamour, a new scriptable browser framework for Pharo. Any stage can be rendered as a *Lumière* presentation inserted into a glamour browser (see Figure 9).

Responding to user events. The shapes of micro-worlds displayed by stage morphs can react to user input, from the keyboard and mouse. *Lumière* provides modifiable stage interaction policies to control the highlighting, selection, clicking and keyboard event handling. For example some stage interaction policies specify single or multiple selection, floating over shapes awareness and different camera keyboards movements.

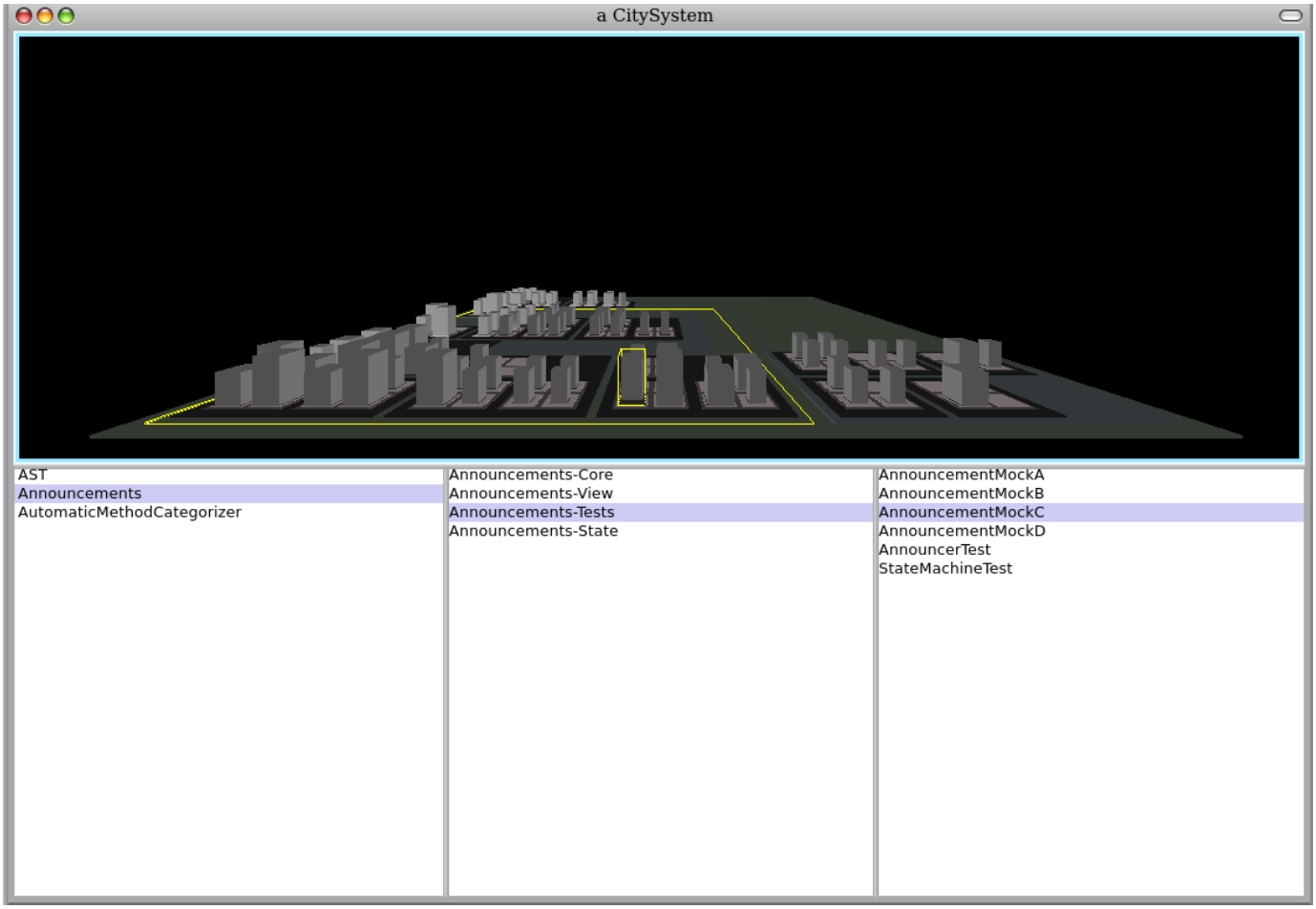

Fig. 9. Glamour integration

In Figure 10 we present to the right a stage morph with a selected shape, and to the lef a stage morph with a highlighted shape integrated into Omnibrowser.

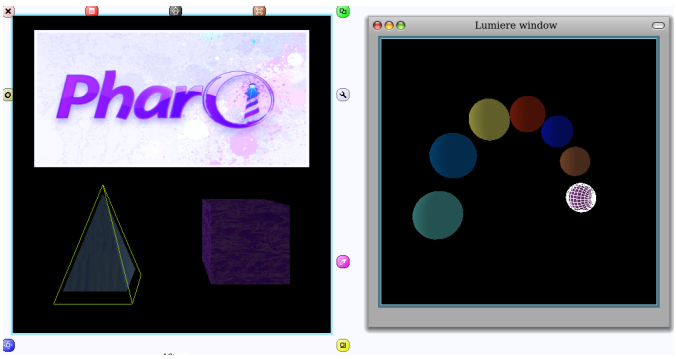

Fig. 10. Pharo integration

IV. Conclusion

In this paper we presented the design and implementation of *Lumière*, our novel 3D framework implemented in Smalltalk. *Lumière* uses a stage metaphor to render micro-worlds – graphs of 3D shapes generated programmatically– in OpenGL. *Lumière* is implemented using Pharo and is fully integrated with the underlying environment. The 3D scenes generated by *Lumière* can be integrated in Pharo's windows and browsers, and *Lumière* provides support for customizable keyboard and mouse interactions.

References

[1] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley, Reading, Mass., 1995.

[2] F. Olivero, M. Lanza, and R. Robbes. Lumiére: A novel framework for rendering 3d graphics in smalltalk. pages xxx – xxx. ACM Press, 2009.

[3] D. Smith, A. Kay, A. Raab, and D. Reed. Croquet - a collaboration system architecture, Jan. 2003.