pARser: Implementation of a XML Parser to Load Virtual Objects onto a Mobile Augmented Reality Interface

Nanette Wu
Massachusetts Institute of Technology
Department of Computer Science & Engineering
nanette@mit.edu

Supervised by Prof. Pan Hui, Dr. Farshid Hassani Bijarbooneh, Wenxiao Zhang
HKUST-DT System and Media Lab

ABSTRACT

This project serves as a file-loading extension to an existing cloud-computing augmented reality library. By utilizing the Persister class parser from the Simple framework, this Android application acts as a scene loader that deserializes information describing virtual objects from an XML file, which contains descriptive attributes such as positioning and lighting. These objects can either be primitives, as defined by the Rajawali package, or user-defined models (.obj or .mtl). After processing the inputted information, objects generated on a mobile augmented reality (MAR) interface are superimposed on a camera preview layer. Ultimately, the library extension can simplify and expedite the process of creating MAR applications and displays.

Key Words: Augmented reality, file parser, mobile device, Android.

1. INTRODUCTION

Augmented reality, or altering the perception of the real world with computer-generated information, has grown in popularity on mobile operating systems[1]. As shown by the virality of applications like Snapchat and Pokémon Go, this concept has proven to attract wide audiences with its novelty. However, creating custom mobile augmented reality (MAR) interfaces remains difficult without a thorough understanding of computer graphics and open source libraries.

This issue engendered an idea to create a "black box" design to utilize these existing tools: take some user input describing a scene, manipulate the scene in a "black box", and load the scene on a mobile device. To a greater extent, is it possible to design a user-friendly tool to load and manage multiple objects on a virtual scene?

2. BACKGROUND

This project is extended from CloudRidAR, "a mobile cloud computing technology" that uses "in-cloud big data processing algorithms" to "offload intensive computational operations to the cloud and co-located devices."[2] Thus, without overwhelming one's mobile device, a wide spectrum of data can be accessed by a user. However, once the information was collected from the server, it became apparent that data implementation and organization were significant issues.

The first attempt at a "black box" solution to manage and load objects was made through the previous work of an intern, who developed an Android-based application called *ARSceneLoader*. By experimenting with an XML file parser, she was able to extract data from an XML file and later process it with her own renderer class. Using the Rajawali[3] package—an Android OpenGL engine—objects could be generated onto an augmented reality layer with descriptive characteristics, such as color, texture, and positioning. Although her application did work properly, inputting data to the XML file was rather inefficient, and some features (i.e. lighting, parent-child relationships) were faulty at runtime. Hence, while her design created a solid foundation for the basis of the app, the execution of this idea needed improvement.

3. CHOOSING A XML PARSER

A file parser is a tool that can access and analyze the contents of a file without having to manipulate its original state. One of its primary functions is to be capable of deserialization, or file reading. Once the file is opened and deciphered, the extracted information can be properly formatted, with further processing, to function as a data library for the user's desired purpose (i.e. information catalog, team roster). If a user changes this information during runtime, the parser should in turn be able to handle serialization, or file writing; the altered data would be reformatted into its initial extracted form, and later be written into a new file or over its original file.

After looking into the Android and Java documentations, it appears that file parsers are fully-functional tools that already exist in open source libraries and built-in operating system packages. Thus, since the original task of determining how to integrate parsing had been resolved elsewhere, the challenge shifted to finding the optimal parser for a mobile augmented reality environment.

An integral decision was to pick the file type to be parsed, which was narrowed to either JSON or XML. While JSON is more lightweight and allows for faster data transmission, it lacks the flexibility that XML has with its markup language structure. Without inherent semantics, XML proved to be a better candidate than JSON—which uses JavaScript object notation—by giving more freedom to the user in the design of their data library. Thus, XML, with its convenient outlining configuration, was chosen as the ideal file type to complement this application.

The next step was to research existing XML parsers that would seamlessly fit the task at hand. The Android developers recommended using either *XMLPullParser* or *ExpatPullParser*, which were included in the Android API. However, since the code was dense and raised concerns about runtime efficiency, the decision was made to use the parser from *ARSceneLoader*. This parser seemed to better fit the application's needs: it used the *Persister* class from the *Simple* XML framework, which implemented the *Serializer* interface that allowed objects to be persisted and loaded from external sources. Moreover, the information read from the file could directly be stored into Java objects, such as ArrayLists and Strings, which affirmed this parser as the optimal choice.

4. OBJECT ORGANIZATION AND MANAGEMENT

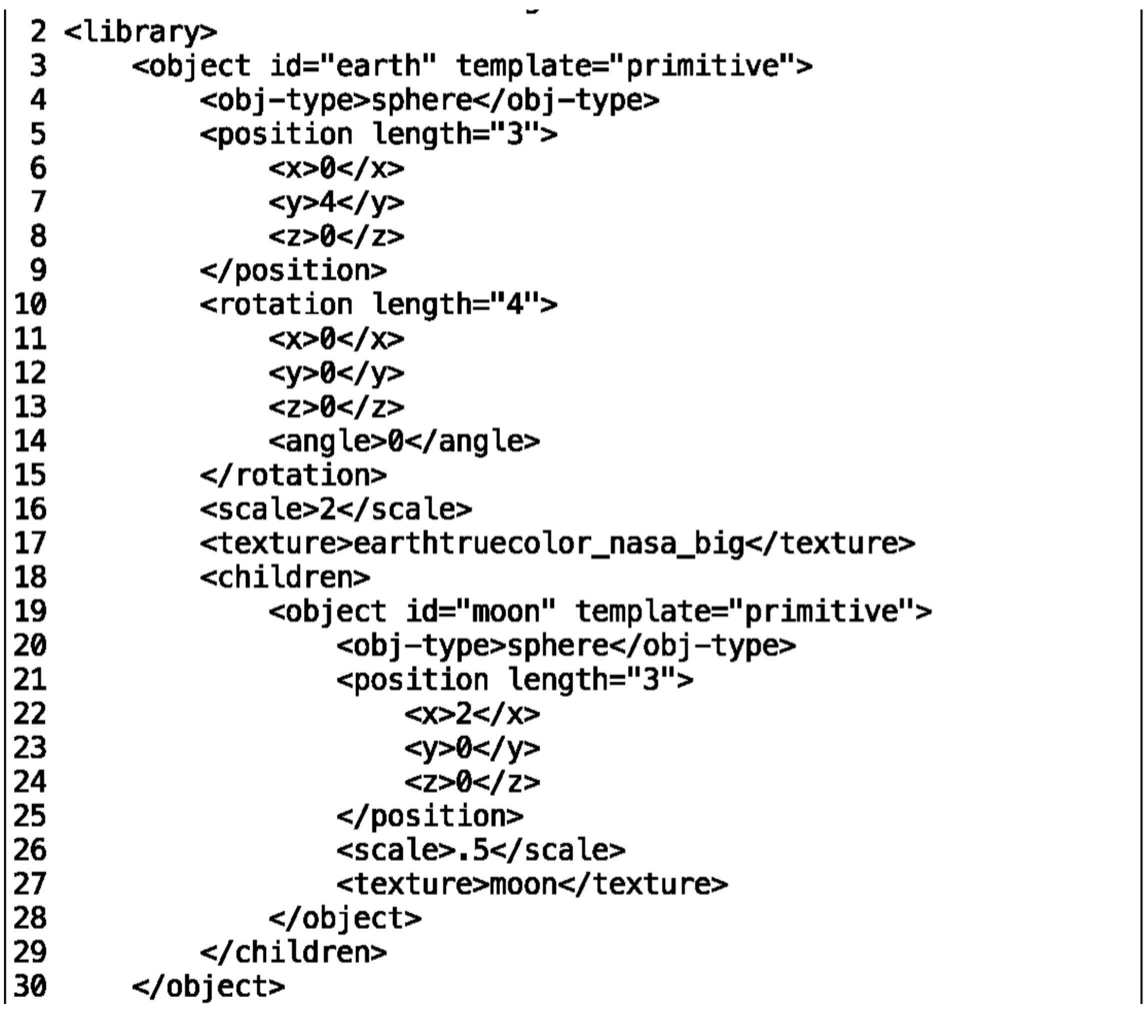
In pARser, the two types of objects to be loaded were *ParsedObject*, a physical graphical object (Figure 1) and *LightObject*, an invisible lighting object (Figure 2). The attributes of these objects were described in separate classes, as described below:

4.1 PARSEDOBJECT ATTRIBUTES

Type	**Var. Name**	**XML Tag**	**Description**
String	id	id	String identifier of object.
String	template	template	Primitive or User-Defined Object
String	objType	obj-type	Primitives only; Rajawali object type.
double[]	position	position	[x, y, z]; Cartesian coordinates.
double[]	rotation	rotation	[x, y, z, angle]; Angle w/vector <x,y,z>
float	scale	scale	Object sizing.
String	texture	texture	Loaded from file in res/drawable.
float[]	color	color	RGB values w/alpha (translucency).
String	filePath	file-path	User-Defined Objects: leads to res/raw.
List<>	children	children	List of ParsedObject children.

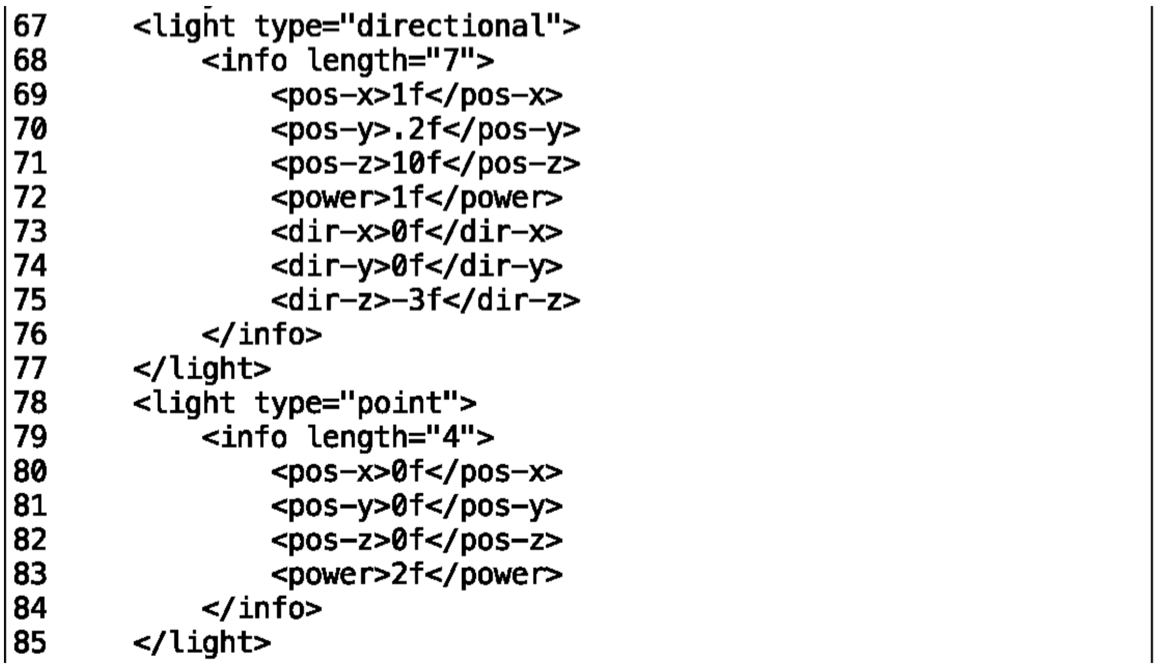
4.2 LIGHTOBJECT ATTRIBUTES

Type	**Var. Name**	**XML Tag**	**Description**
String	type	type	DirectionalLight or PointLight; see Rajawali package for detailed info.
float[]	lightingInfo	info	Indices 0, 1, 2: Position (x,y,z) Index 3: Power (strength of light) Indices 4, 5, 6: Ray Direction (x,y,z)



```xml
<library>
    <object id="earth" template="primitive">
        <obj-type>sphere</obj-type>
        <position length="3">
            <x>0</x>
            <y>4</y>
            <z>0</z>
        </position>
        <rotation length="4">
            <x>0</x>
            <y>0</y>
            <z>0</z>
            <angle>0</angle>
        </rotation>
        <scale>2</scale>
        <texture>earthtruecolor_nasa_big</texture>
        <children>
            <object id="moon" template="primitive">
                <obj-type>sphere</obj-type>
                <position length="3">
                    <x>2</x>
                    <y>0</y>
                    <z>0</z>
                </position>
                <scale>.5</scale>
                <texture>moon</texture>
            </object>
        </children>
    </object>
```

Figure 1. XML description of a primitive object with a child.



```xml
    <light type="directional">
        <info length="7">
            <pos-x>1f</pos-x>
            <pos-y>.2f</pos-y>
            <pos-z>10f</pos-z>
            <power>1f</power>
            <dir-x>0f</dir-x>
            <dir-y>0f</dir-y>
            <dir-z>-3f</dir-z>
        </info>
    </light>
    <light type="point">
        <info length="4">
            <pos-x>0f</pos-x>
            <pos-y>0f</pos-y>
            <pos-z>0f</pos-z>
            <power>2f</power>
        </info>
    </light>
```

Figure 2. XML description of two types of light: a light ray (directional) and light source (point).

5. ANDROID APPLICATION DEVELOPMENT

With an XML parser and a design for object organization in place, the next step was to implement it in the Android framework. The Rajawali package (version 1.1.970) was compiled to use OpenGL in Android, and the Simple framework (version 2.7.+) was also compiled to utilize the XML parser. Using Android Studio, the application was broken down into several classes:

`MainActivity` – Creates a modifiable AR interface and links the renderer/touch listener.

`MainRenderer` – Handles visual settings; creating objects, camera positioning, lighting.

`LoadedScene` – Implements the XML parser and binds objects to the scene.

`ParsedInfo` – Stores information loaded from XML file.

`ParsedObject` – Contains features describing a graphical object to be loaded on a scene.

`LightObject` – Contains features that describes a light object to be loaded on a scene.

`XMLParser` – Implements the *Persister* class to read and write to descriptive XML files.

As an extension from the CloudAR library, pARser also used generalized classes that already existed in the CloudAR package[3]:

`ARContent` – Objects that should be loaded to an *ARScene*.

`ARScene` – Interface that represents the parent class of an augmented reality scene.

Furthermore, to generate a camera preview, a *Camera2* fragment from the Android documentation was also included. Although *ARSceneLoader* originally included a working camera code fragment, the old *Camera* API was already deprecated and needed to be replaced to keep up with the latest Android SDK version. The classes imported from the *Camera2* fragment include:

`AutoFitTextureView` – Fits the camera onto an Android screen.

`Camera2BasicFragment` – Handles *Camera2* API from Android.

`CameraActivity` – Adds the camera to the container being used.

6. TESTING AND ENHANCEMENTS

After the parsing tool became fully functional, the final step was to create a test scene displaying the objects that overlaid a camera preview. An XML file was created that tested the major elements that were created: Primitive objects, UserDefined objects, DirectionalLight, and PointLight (Figure 3). Two primitive objects were added; one with texture (earth) and a child (moon), and one with a custom color (light blue cube). A custom object created on Blender was also added (linked rings and monkey). A directional light was set to make the objects appear more three-dimensional with a shadow, and a point light was set underneath the Earth, which makes the white South Pole appear more illuminated. To better display the parent-child relationship between the Earth and the moon, the objects also rotate counter-clockwise about the y-axis <0, y, 0>.

Finally, to make the scene interactive, a drag-and-drop feature was added to move around the elements in the scene. The logic behind this was taken from the *TouchAndDragFragment* included in the Rajawali package, and involves an extensive understanding of computer graphics.

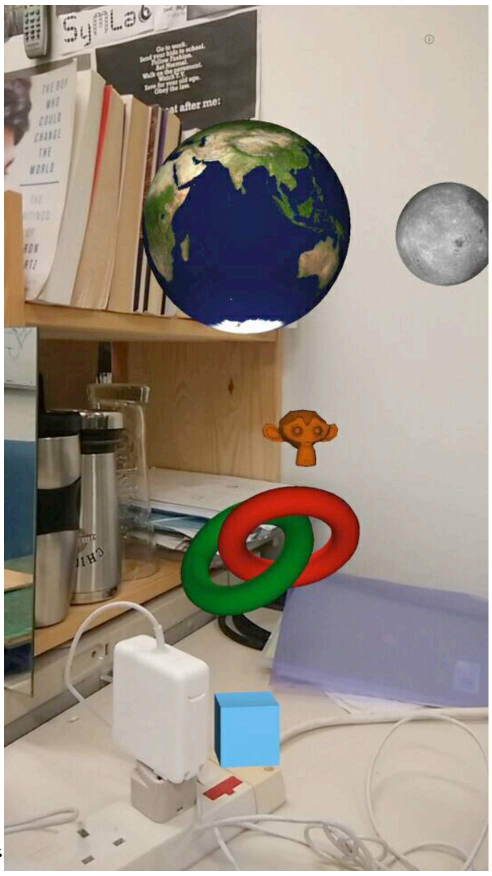

‘

Figure 3. Example overlay of primitive and user-defined objects.

7. RESULTS

The final application successfully incorporates an XML parser in an Android application for MAR. Information extracted from an external file is processed to generate an augmented reality scene overlaying a camera preview. There is no limit on the number of objects that can be loaded onto the scene; however, screen size and CPU capacity should be considered when doing so. Each object can have as many descriptive characteristics as desired, bearing in mind said constraints. Once the scene is fully processed, the only file that should be further manipulated is the XML file, which describes the scene in the "assets" folder, to change the scene appearance.

The tasks accomplished by this application are cyclic; each step taken is a result of a previous step and pushes forward successive actions (Figure 4). The process begins with a descriptive external XML file. Once the file is deserialized by the built-in parser, the contents are stored in a ParsedInfo class, which has two lists that track: 1. Objects to be rendered and 2. lighting objects. These lists are further dissected and transformed into Rajawali objects, which are stored in a staging layer via the LoadedScene class. Once properly declared and initialized, these objects are bound to a MAR scene with the help of a renderer class. If desired, the finished product can be serialized into an XML file and stored for later use, thus starting from the beginning again and creating a cycle.

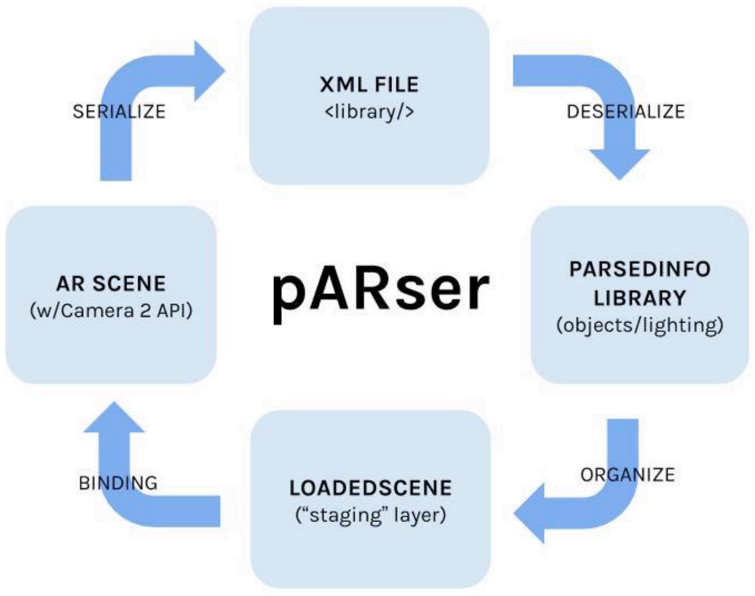


Figure 4. Summary of concept from XML file reading to AR scene loading.

8. DISCUSSION/CONCLUSION

pARser accomplishes the goal of creating an augmented reality scene manager: the implementation of an XML parser takes care of object parsing, while the functionalities of a well-designed Android program manage multiple objects on a virtual scene. Although pARser does what it is expected to do, there are still several improvements that can be made; the application is far from being a finished product. The Rajawali package is difficult to understand for those who do not have prior knowledge of OpenGL. As a result, the implementation of the descriptive object attributes is not user-friendly and requires extensive research to do so. A potential improvement to simplify the usage of Rajawali is to have a more intuitive interface to manipulate the settings of the scene. For example, there could be a graphical user interface (GUI) for the user to adjust intensities of light, color, and transparency with sliding scales. If the challenge of optimizing screen space on a mobile device could be overcome, the GUI would further encapsulate the idea of making this a simple tool to manage objects on a scene. A slightly more complex direction to take pARser could consider adding a built-in library for scene interaction, such as a drag-and-drop function. If this were successfully implemented, this application would be a valuable tool for any user to generate augmented reality scenes on their mobile devices.

REFERENCES

1. What is Augmented Reality (AR)? - Definition from Techopedia. (n.d.). Retrieved August 04, 2017, from https://www.techopedia.com/definition/4776/augmented-reality-ar

2. HKUST, Computer Science and Engineering. (2016, September 20). HKUST Builds Frameworks to Boost Development of Mobile Applications on Augmented Reality [Press release]. Retrieved August 4, 2017, from http://www.ust.hk/about-hkust/media-relations/press-releases/hkust-builds-frameworks-boost-development-mobile-applications-augmented-reality-2/

3. Ippel, Dennis. Rajawali Package. (2013). GitHub repository. Retrieved August 04, 2017, from https://github.com/Rajawali/Rajawali

4. Huang, Z., Li, W., Hui, P., & Peylo, C. (2014, June). CloudRidAR: A cloud-based architecture for mobile augmented reality. In Proceedings of the 2014 workshop on Mobile augmented reality and robotic technology-based systems (pp. 29-34). ACM.