Intermediate Report for Augmented Reality

Erin Jamroz, Selah-Mae Ross, Matthew Burke, Thomas Freeman, David Greene

October 24, 2012

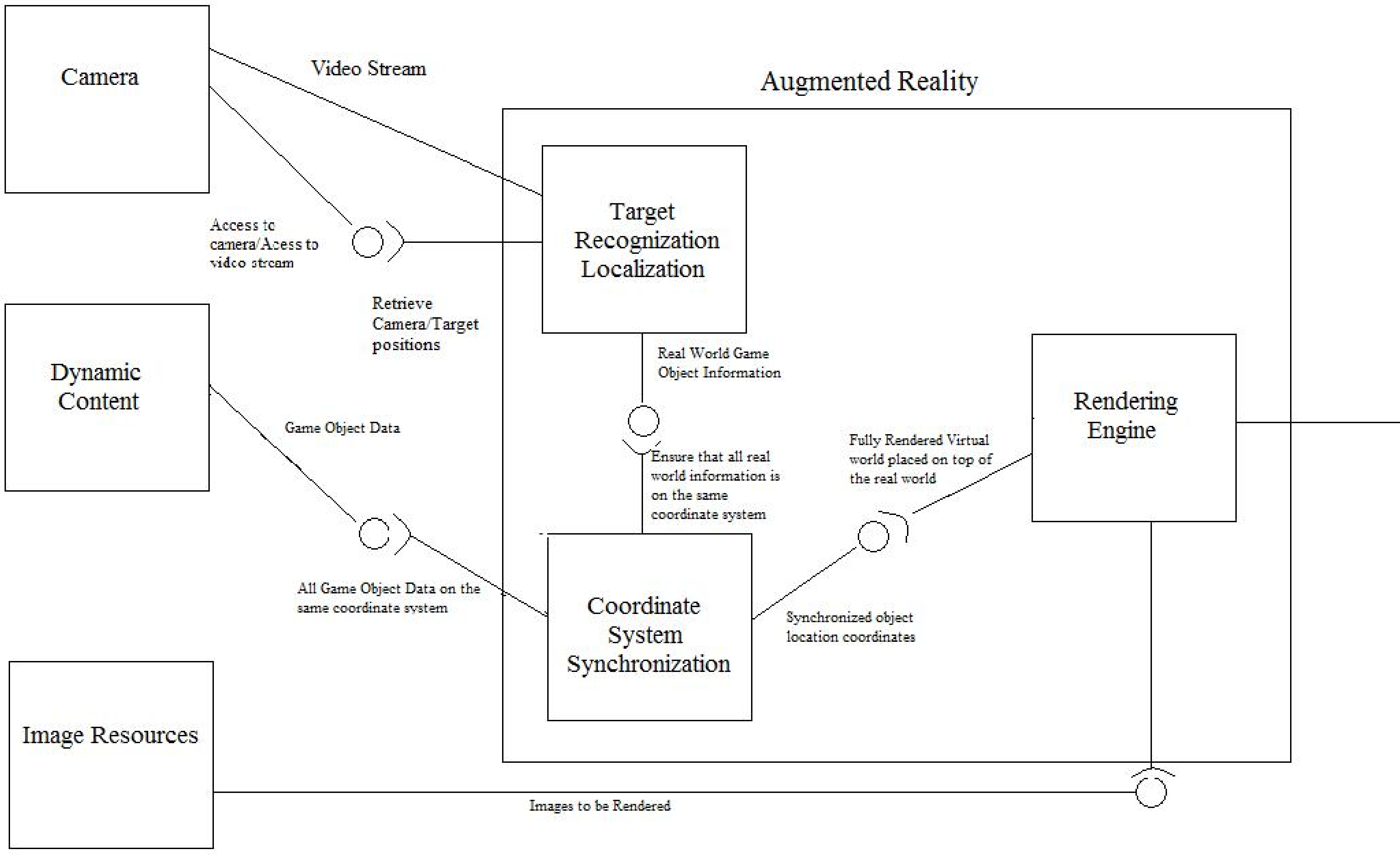
Abstract

Our company, *Vi-Char,* is developing a unique game which implements augmented reality, motion capture, and interactions with input into the game environment by individuals who are not involved in the controlling of the character. The game itself will be a platformer in which the character of the game, a robot with a jetpack, is controlled by three players through the motion capture via the Microsoft Kinect. The world will be rendered as a 3D scene for Android phones users using augmented reality, and the users of the phones will be the ones introducing new obstacles into the scene in order to attempt to trip the playable character up.

Of the five separate divisions working on the creation of this game, this Intermediate Report is the documentation for the Augmented Reality Development Team. The Augmented Reality team has been tasked with the process of taking live video footage as fed by the Android camera and given by the Mobile Development Team, and superimposing a 3D scene on top of the real world. The AR team will use models provided by the Game Engine Development Team and render them at coordinates that have been determined based on the phone's positioning in relation to a pre-set target on in the game's static world. Though the setting of the world never changes, the scene being rendered on the Android phone will always be changing depending on the user's position and what they can see.

The contents of this Intermediate Report will cover several aspects of *Vi-Char's* game. To start, a brief mention of the design strategy planned in order to complete the augmented reality development portion of the project will be produced along with an updated version of a design diagram. Following the design strategy is an explanation of procedures that must be followed in order to install the augmented reality application that we are developing for the Android phone. Next will be the implementation details for the AR team, followed by the procedures that were completed in order to progress to our current status. After an explanation of the procedures, there will be a section in which our previous plans and thoughts, difficulties, and changes that the Augmented Reality team have reflected on. In relation to the changes that the AR team have determined need to take place, the next section will include a modified timeline that will include objectives that we completed on time, late, and when we believe we will be able to finish our next determinable goals. The final part of the Intermediate Report will be a glossary in order to help the reader ease through foreign terminology that may present itself in this report.

Design Summary


Camera
Video Stream
Augmented Reality
Target Recognization Localization
Access to camera/Acess to video stream
Retrieve Camera/Target positions
Dynamic Content
Real World Game Object Information
Rendering Engine
Game Object Data
Ensure that all real world information is on the same coordinate system
Fully Rendered Virtual world placed on top of the real world
All Game Object Data on the same coordinate system
Coordinate System Synchronization
Synchronized object location coordinates
Image Resources
Images to be Rendered

In order for the Augmented Reality Development Team to complete their determined portion of the *Vi-Char* interactive game, there are several inputs that we require in order to complete the design previously determined. As follows the design diagram above, the AR team requires a live video stream in order to recognize the location of targets and the camera in relation to the real world as shown in the module “Target Recognition Localization”. This stream is provided by the camera, and in turn, the Mobile team. After the camera and target positions are determined, that information is then passed on to be merged with data provided by the webserver and phone as done in the “Coordinate System Synchronization” module. Next, the information that was processed in the previous module is passed along and merges with data provided by the Resource data bank. When these merge within the “Rendering Engine” module, the result is a 3D world superimposed on the real world. This scene is then sent to the phone’s screen, where it is displayed for the Android phone user.

Install Documentation

The following instructions will allow the user to download this project and run the application on a phone. They will not enable the user to modify the code. If you would like this functionality, see the installation notes below. These instructions assume that the user has a current versions of the Eclipse IDE, Git, and the Android SDK installed.

If you need any of these, you can get them here:

- Eclipse
- Git
- Android SDK - Be sure to follow the installation instructions thoroughly. To run this application, you will need the Android 2.2 (API 8) package installed. You can download it though the Android SDK manager. It is also recommended that you install the Eclipse ADT plugin. You will find instructions in the Android SDK installation instructions.

Once you have all of the above installed, download the Augmented Reality repository from https://github.com/UPS-CS240-F12/augmented-reality.git and follow the steps below:

1. Find the ImageTargets folder under augmented-reality/samples/ and import it as existing Android code. You should have a project named com.qualcomm.QCARSamples.ImageTargets.ImageTargets.
2. You will likely need to add a new class path variable to the Java build path. Open then Eclipse Preferences pane. Got to Java->Build Path->Classpath Variables. Create a new classpath variable named 'QCAR_SDK_ROOT' and point its path at the root of the augmented reality repository (by default it is augmented-reality/).
3. At this point you may have a build error. If all you want to do is install the application on a phone and run it, this is easily resolved. If you want to modify any of the JNI resources or code, then you will need to install the android native development kit. See notes below. Otherwise, select the project, then open the project properties under the Project->Properties menu. In the properties window, select builders, and then remove the offending builder from the list (most likely an unrecognized external builder).
4. Clean the project.

5. Connect an Android (API 8 or newer) device with USB debugging enabled (Settings->Applications->Development).
6. Run as an Android application on the connected device.

Installation Notes

1. We have only been able to run this application directly on an Android device. We are not currently able to run it in the emulator.
2. If you want to be able to modify the JNI resources (i.e. C++ code), you will need to install and setup the Android NDK. This should allow you to edit, and re-build the project.
3. If you want the full monty, follow the Vuforia SDK installation instructions. They are long but thorough. Pay attention and don't skip steps.
4. Once running, you'll probably want some Image Targets to point the phone at. They are located in the media/ folder: augmented-reality/samples/ImageTargets/media . Both 'stones' and 'chips' will work immediately. The app won't track 'tarmac' until the user opens a menu and swi

Implementation Details

SEE APPENDIX (A)

For our design we chose to take one of the sample projects already provided by Qualcomm. After extensive testing we concluded that ImageTargets.cpp provided the best basis for our projects goals. ImageTargets provides the functionality to display a single object on a single target. We have taken ImageTargets and added the additional functionality that we required for the game to function as specified by the class. With the groundwork already provided to us we are able to focus on implementations. Currently the only format for storing object data is in the form of a C++ .h file. The augmented reality package works best with .h files.

In our current implementation we do not have not required much fault tolerance. However, once integration with other groups occurs we will require a higher level of fault tolerance. Some things we plan to handle are when we fail to receive an update on a particular object and when we lose connection to the server all together. When we lose track of a particular objects updates we will wait a specified time before terminating the object (no longer render or tack it). If all contact with the server is lost we unfortunately can not run autonomously so we will have to pause the game until connection is reestablished.

As previously mentioned we store our object data in the form of .h files. Every game object that needs to be rendered needs to be stored this way. As long as we receive all the game objects before installation on the mobile devices we can convert the objects ourselves from various other formats (.obj, .3ds, etc). Every game session we will need to store coordinate data in the form of a 4*4 matrix. This coordinate data is only temporary for each game session so therefore only needs to be stored for each game and can be wiped between games.

Procedures Followed

At this still relatively early stage of implementation the division in group tasks has been fairly weak. Most of us have dealt with the same relatively small section of code, though some individuals have take more specific focuses. Matthew Burke has done much to integrate smoothly with the Vuforia side of things, and has brought us recognition of multiple AR targets and objects displayed simultaneously. Erin Jamroz has been working on creating a ‘draw’ method, which would ultimately process all the data coming from the game engine team. Thomas Freeman has worked on converting the blender textures to a format that we can import and use successfully in Vuforia. David Greene has worked with tweaking aspects of the drawing, mainly interfacing with OpenGL ES. Selah-Mae Ross will work on retrieving easily conveyable information from Vuforia’s position matrices to relay target locations and other such information back to the game engine.

Our team has had to interface with a very large number of tools on the project, and probably the most on the entire team. Our team, like all others, has relied on github for source control. As for code, we work in both Java as well as C++ in order to interface correctly with Android and OpenGL ES respectively. We have had to use the Android SDK, Vuforia SDK, OpenGL ES, and Blender in order to accomplish our task. We currently use Eclipse for our coding environment, as it provides excellent tools for Android development.

Our basic testing procedures involve careful management of our repository, and the use of Android debugging tools. maintaining local copies of the git repository, creating individual branches for each specific task being performed, and not pushing new versions to the main server until we are certain the modifications has not broken any aspect of the software. Eclipse in conjunction with the Android SDK provides a good environment to develop in, as it is quite quick and easy to compile code in eclipse and then run it directly on the Android device via USB. This allows us to test thoroughly and rapidly, allowing for testing to happen as changes occur, rather than waiting for an entire build to finish being developed.

Our coding and naming conventions are not entirely well defined, as none of our parties has written large amounts of code. Most of our code as of yet is refactored Vuforia sample code, which itself has exceedly poor style. We attempt to name variables clearly, with unambiguous names that provide some comprehensible short description of what is contained. We also attempt to comment our modifications, as well as the functionality we have discovered from the almost completely uncommented sample code. Our conventions are not thoroughly established however, as almost all of our coding as of yet is simply tweaks and conversions of existing code, and almost all of this will change when the final development and integration cycle begins.

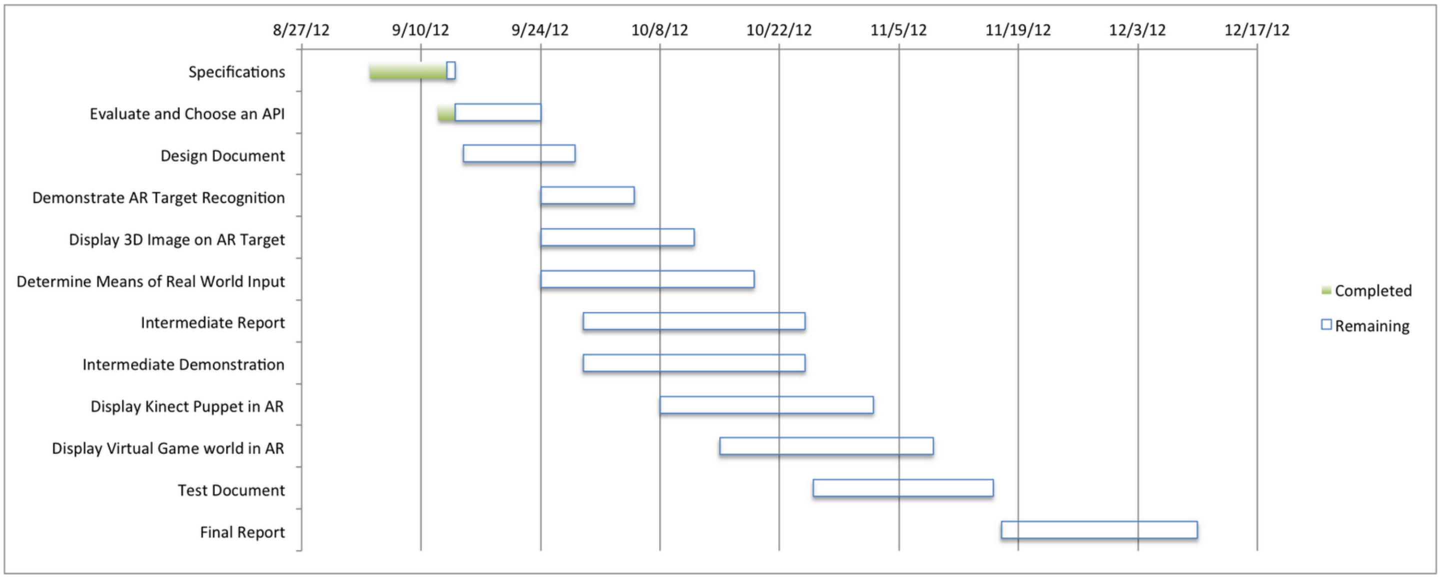
Reflection

Our group has faced a great many challenges throughout the life of this project ranging from tricky installation to integrating with poorly documented code. From the very start, getting Vuforia software to compile and run on your machine is challenging task. First, a solid understanding of eclipse would be a nice prerequisite because you must know which settings to modify and how. Second, there are multiple packages that must be installed, which require you to modify environmental path variables form the command line. This task is quite challenging if you are unfamiliar with the command line in the first place. Looking back, it would have been more efficient to have all of the members of the group sit down and install all the software together. All the bugs could have been worked out in conjunction and everyone would have a system to develop and test on. While the installation had its hassles, the most challenging aspect of this project has been using/integrating with existing systems and architectures. At no point has our group been able to develop from the ground up because we must work our design into the existing Vuforia architecture. We are constantly modifying existing Vuforia commands/code to suit our needs. To add to this difficulty, their code has little to no comments, making it very difficult to actually find relevant sections of code.

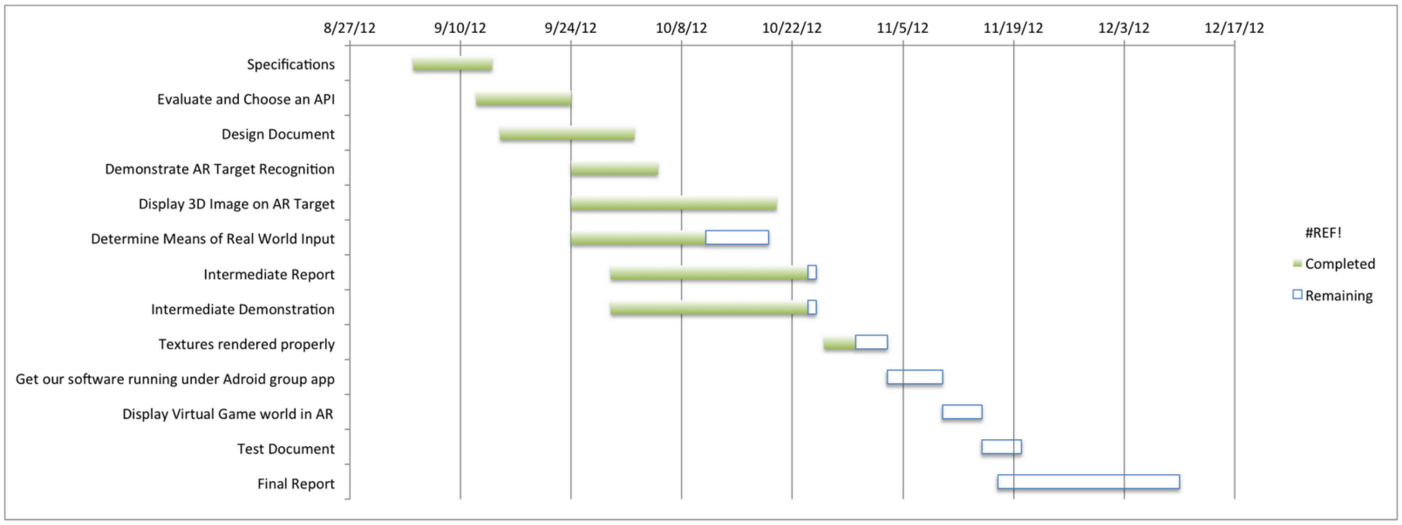
One of the strategies for working with the code that has worked well is breaking up the group into smaller task oriented teams. As opposed to all of us working with all of the code to accomplish a large task, we have worked in two-person teams on portions of the code to achieve smaller tasks. We then reconvene and discuss what we have figured out, so that everyone can understand the sections of code that other groups were working on.

We have done a fairly good job of sticking to our original timeline. We have had all relevant documents finished on time. As for our coding deadlines, we are behind on determining if real-world input will be possible. We also have not gotten actual game engine models to render properly on the phone yet, which is our largest shortcoming. Aside from these two goals not being met, we have gotten 3d data of our own to render on the phone and have an understanding of how to manipulate that data after it has been rendered. Our next major goals are to get game data to render properly, have our software run underneath the android group’s application layer, and be able to display dynamic content. Below is our original timeline, which has been updated (See Timeline section for details).

Old Time Line



Timeline



We have done a fairly good job of sticking to our original timeline. We have had all relevant documents finished on time. As for our coding deadlines, we are behind on determining if real-world input will be possible. We also have not gotten actual game engine models to render properly on the phone yet, which is our largest shortcoming. Aside from these two goals not being met, we have gotten 3d data of our own to render on the phone and have an understanding of how to manipulate that data after it has been rendered. Our next major goals are to get game data to render properly, have our software run underneath the android group's application layer, and be able to display dynamic content.

Glossary

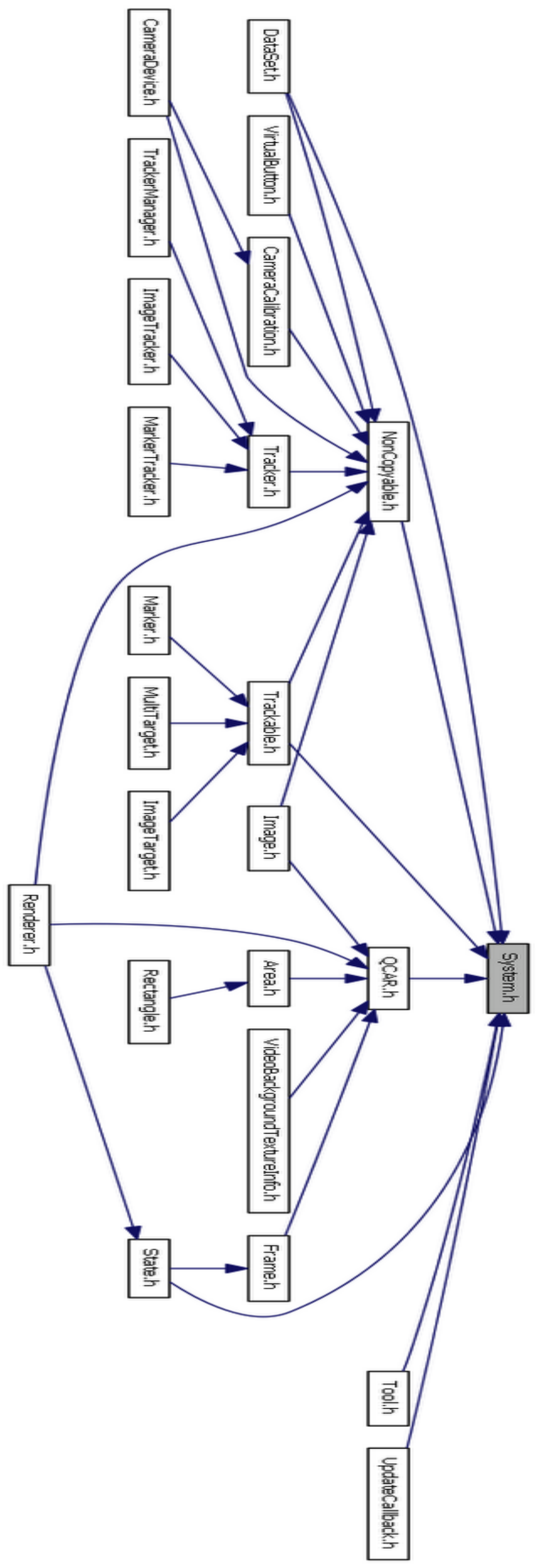
Android: Mobile operating system for smart phones and tablets, developed and owned by google.
Blender: Free open source software for 3d modeling available for all major operating systems.
Image Resource Data Bank: This will be a collection of image resource files of all objects to be displayed in the game. It will be stored on the phone.
OpenGL ES: Open graphics language for use on embedded graphics, used commonly for Android as well as iPhone app development.
SDK: Source Development Kit: An assortment of tools for use by software developers so that they can develop for certain platforms and interface with certain systems.
VuforiaTM: is an augmented reality software development kit for mobile devices. It is produced by Qualcomm and facilitates the development of AR applications.

VuforiaTM Terminology:

- **Image Target/AR Target:** Images that the Vuforia SDK can detect and track. Any image that has certain characteristics and has been processed by the Vuforia Target Management System may be used as an Image Target. Image Targets have robust tracking and are resilient even when partially occluded.
- **Frame Marker:** these are simplified AR targets compared to Image Targets. The Vuforia SDK identifies Frame Markers through a unique id coded into a binary pattern of black and white squares around their edge. The entire
- **Multi-Target:** A collection of Image Targets arranged on a three-dimensional box. Since the relative locations of the targets is known, the SDK can anticipate their locations and include augmentations even before new targets are acquired.
- **Virtual Button:** Virtual buttons can be included in Image Targets. These buttons are "pressed" when they are occluded, at which point, some corresponding action can be executed.
- **Target Management System:** A web based system that allows developers to upload custom images to be used as Image Targets.

Appendix

A.


DataSet.h
VirtualButton.h
CameraCalibration.h
CameraDevice.h
TrackerManager.h
ImageTracker.h
MarkerTracker.h
Tracker.h
NonCopyable.h
Marker.h
MultiTarget.h
ImageTarget.h
Trackable.h
Image.h
Renderer.h
Rectangle.h
Area.h
QCAR.h
System.h
VideoBackgroundTextureInfo.h
State.h
Frame.h
Tool.h
UpdateCallback.h

References

"Android Developer Tools", Android, last accessed 10/23/2012 http://developer.android.com/tools/index.html

"Blender Home Page", Blender, last accessed 10/23/2012, http://www.blender.org/

"OpenGL ES Develop Overview" Khronos, last accessed 10/23/2012, http://www.khronos.org/opengles/

"Vuforia Developer Guide," Qualcomm, Version 2.1.18, last accessed 10/1/2012, https://ar.qualcomm.at/developer_guide

"Vuforia Augmented Reality SDK," Wikipedia, last updated 9/8/2012, last accessed 10/1/2012, http://en.wikipedia.org/wiki/Vuforia_Augmented_Reality_SDK