Assignment 3: Putting It All Together

At this point, you've started to learn about some of the many ways to select and manipulate objects in 3D user interfaces. Now, it's time to try this yourself, using Unity and Vuforia to write your first augmented reality app. You will construct an augmented environment in which a user will be able to create and edit instances of virtual objects that can be translated, rotated, and scaled; and assemble compound instances that can be manipulated in the same way as regular instances. The goal is to prepare you to understand the techniques and programming tools you'll be using in your final project.

First, make sure that you included Vuforia in the patch release of Unity 2017.3 on your computer, and that the patch release is no older than Patch 2017.3.0p2. If you did not check the Vuforia package box when you installed Unity, you will need to reinstall Unity and click that box in the installer. (Refer back to Step 2 in our IA document on Installing Unity and Vuforia if you need help.) The Vuforia Developer Portal will be a crucial resource throughout this assignment. Begin by reading the Vuforia Getting Started page. You should also watch the following three video tutorials before proceeding with the assignment: ARCamera prefab in Unity, Image Targets in Unity, and Vuforia Play Mode for Unity. You will be using Image Targets (and, optionally, Cylinder Targets) in this assignment; therefore, it will be essential for you to read the Image Targets and Cylinder Targets guides to understand how they work. Your targets should be defined in a Device Database (which we provide, as described below). You should also read our IA document on Developing with Unity and Vuforia. (Please be sure to familiarize yourself with all documentation mentioned in this paragraph!)

All interaction in this assignment should be accomplished based on the relative position and orientation of tracked physical objects (Vuforia Image Targets or Cylinder Targets), tracked parts of the environment (only if your device supports Vuforia Ground Plane), and your Android/iOS device. These can be optionally accompanied by triggers or modifiers specified using one or more button or touchscreen interactions. That is, you should not use the touchscreen for selection or manipulation, except insofar as you use it (a) to specify a vector relative to the camera, (b) as a trigger (e.g., tapping the screen), or (c) as a modifier (e.g., touching the screen to modify some behavior). Note the additional requirement of needing to use at least one toolbar for selection, as described below. Furthermore, triggers or 2D UI components should not be used for specifying the magnitude of a transformation; for example,
you should **not** use a physical or virtual button to scale up/down in discrete steps or a 2D slider to set the scale.

Using the touchscreen as a trigger is *optional* because you may decide to use other means to trigger selection (e.g., using Image Targets). However, you are welcome to use the touchscreen or 2D UI components in any way you would like for debugging purposes only (i.e., *not as part of your final UI*).

**Functionality**

**Objects and Instances**

Your app should allow the user to create, delete, select, and transform any number of instances of at least four different virtual objects. At least two of the objects should be models created either by you (using either Clara.io or VECTARY, which can create .fbx objects containing relatively small numbers of polygons) or by others (properly acknowledged in your documentation). The other objects, if any, can be created directly in your Unity code. While you will specify the initial position, scale, and orientation of each instance, the user will be able to change it.

**Workspace**

Your instances should reside within a *workspace* whose coordinate system is determined either by a printout of a large Image Target (we will call it the *ground target*) that you place in your environment (see the description of Vuforia Targets later in this assignment) or by using Vuforia Ground Plane (*if your mobile device supports it*). When Vuforia recognizes the physical printout of the ground target within the camera image (or identifies an appropriate planar surface using Ground Plane), it determines the position and orientation of the camera relative to the target or surface, and derives the geometric transformation that can be applied to instances that you wish to visually “attach” to that target or surface. In addition, the camera image itself will need to be drawn as the background in the frame buffer to make this a video see-through AR app. (The Vuforia ARCamera prefab will handle this for you. You can read more about this in *Getting Started with Vuforia in Unity*.) Since you are doing this assignment with a device containing an integrated camera and display, your app will be an example of what is sometimes called “magic lens” AR.

**Selection**

To select instances, you are welcome to implement any 3D selection technique discussed in class or the course reading, or a variation on it, provided that your selection technique uses at least one Image Target or Cylinder Target (independent of the ground target). We will call this Image Target or Cylinder Target a *toolbar*, and its position and orientation should be controlled by the user. For example, the user could hold (or wear) a toolbar Image Target mounted on a card or Cylinder Target serving as a wand, whose position and orientation are tracked to accomplish pointing, or to be a virtual hand. Or, if you wore or mounted your device, you might hold (or wear) toolbar targets in or on both hands; for example, to perform two-handed ray
pointing or a two-handed image-plane technique. These are only examples; you’re encouraged to try other alternatives and to speak with us to get feedback on your ideas. In all cases, your 3D UI (just like a good 2D UI) should provide feedback to make it clear when you have selected an instance and what you have selected. You are not required to support selecting more than one instance at a time.

**Translation**

You should make it possible to move the selected instance to any desired location within the workspace (i.e., with arbitrary x, y, and z coordinates) that is visible to the camera while it is tracking, and which can be reliably reached. The user should be able to translate an instance without affecting the instance’s scale or orientation.

**Scaling**

Your user should be able to scale the selected instance isotropically (i.e., uniformly scaling by the same user-specified factor along all three of the instance’s principal axes). You are optionally invited to support anisotropic scaling (in which the user specifies scaling by different factors along one or more axes). The user should be able to scale an instance without affecting the instance’s translation or orientation.

**Rotation**

Your user should be able to rotate the selected instance to any desired orientation within the workspace. (Depending on the approach you implement, you may need multiple interactions to rotate to an arbitrary orientation.) Please make sure that that each instance's orientation is visually obvious, based on some combination of the instance’s geometry and surface properties (e.g., texture). The user should be able to rotate an instance without affecting the instance’s translation or scale.

**Coordinated translation and rotation through seamless transition to a toolbar coordinate system**

Your app should support some way in which an instance can be seamlessly transferred between the workspace coordinate system and a second coordinate system (and then back again). The second coordinate system should be defined by a toolbar Image Target or Cylinder Target. By seamless, we mean that the position and orientation of the instance as seen by the camera should not change during the transfer, except insofar as the camera, toolbar, and either physical ground target or ground plane stage move relative to each other. In other words, if the camera, toolbar and ground target or ground plane stage are stationary relative to each other during the transition, a seamless transition would have absolutely no visible effect until there is relative motion, at which point the instance should be rigidly attached to its current coordinate system. (Note that this should be extremely easy to do in Unity, since the position, scale, and orientation of an object in world coordinates will be unchanged when you move the object from one parent to another.)
Once a transfer has occurred, either from workspace to toolbar, or from toolbar to workspace, only the toolbar or ground target or ground plane stage to which the instance is currently attached should need to be tracked for the instance to be displayed relative to it.

Note that when the instance is transferred back to the workspace, its position and orientation relative to the workspace will have changed unless the workspace and toolbar have remained completely motionless relative to each other during the entire time that the instance has been transferred to the toolbar. Thus, this approach allows your user to simultaneously translate and rotate the selected instance.

(Please reread the last two paragraphs: If your app isn't doing what it says, you should talk to an IA.)

Assembling compound instances in the assembly space

As described thus far, your workspace can be populated by instances of the four or more objects that you support and each instance can be individually manipulated. But what if the user would like to manipulate a collection of instances as if they were a single instance? To make this possible, you should support what we will call an assembly space, in which the user can assemble a set of instances into a compound instance that will be treated as if it were a single instance when it is manipulated in the workspace. That is, a compound instance in the workspace should act as if it were a regular instance insofar as the selection, translation, rotation, scaling, and deletion techniques that you implement in the workspace should treat a compound instance as if it were a single instance. The assembly space should be fixed to a large Image Target (i.e., it should not use Vuforia Ground Plane) and that Image Target should be different from any used for the workspace.

The assembly space should initially be empty. When the assembly space is empty, the user should be able to select an instance that is already in the workspace and indicate that it is to be viewed in the assembly space, causing it to also appear there. That initial instance should appear at the center of the assembly space, at a scale and orientation that you choose to make the assembly process easy for the user.

The user should then be able to create new instances of any of the four objects in the assembly space, and translate, rotate, scale, and delete them individually as desired (using whichever techniques you want) in the assembly space. As each new instance is added to and manipulated in the assembly space, it should also be able to be seen at its appropriate relative scale in the workspace. Note that only the first instance added to an empty assembly space is chosen from the workspace.

While there is anything in the assembly space, you do not need to support direct interaction with the workspace to create, delete, or manipulate objects in the workspace. At any time while there is anything in the assembly space, the user should be able to commit the assembly space or clear the assembly space. If the user commits the assembly space, it should become empty, but
the compound instance being assembled in it should remain in the workspace, where it can be
manipulated as if it were a single monolithic instance. If the user clears the assembly space, it
should become empty, and the workspace should revert to the way it was when the assembly
space was last empty.

If the first instance added to an empty assembly space from the workspace is itself a compound
instance (previously assembled in the assembly space), then the user should be allowed to edit
that compound instance. That is, the user should be able to add new instances of any of the
four objects and translate, rotate, scale, and delete any of the individual instances of which the
compound instance is composed. Clearing or committing the assembly space should work as
previously described. That is, clearing should leave the compound instance the way it was when
the assembly space was last empty, while committing means the compound instance will
typically have different or differently transformed components than it did when it was added to
the empty assembly space. Thus, a compound instance can only be edited in the assembly
space and cannot be made a building block of a higher-level compound instance.

**Vuforia Targets**

We prepared a Vuforia target database and accompanying target images from which you should
pick ones to use—please do not use any other targets in your assignment. The target database
includes:

- Two 8.5" x 11" Image Targets, each of which contains a 9.75" x 6.86" image.
- Four 2" x 6" Image Targets, each of which could be used to create a planar “wand.”
- One Cylinder Target with the same dimensions as a small 250mL drink can (e.g., Red
  Bull): 5.24" tall, 6.58" circumference, 2.06" top/bottom diameter.
- Four Frame Markers, which are not supported in the current version of Vuforia.

Please print any target on 8.5" x 11" (“letter”) stock so that the absolute size of each target will
not be changed. Just to make sure, please choose whatever your print dialogue’s equivalent of
an “Actual Size” option under “Paper Sizing and Handling.”

Any Image Target should be printed on (or attached to) a piece of thick card stock, rather
than thin paper, so that it maintains its planarity during use. Image Targets that are
nonplanar (e.g., curled, creased, wrinkled, or crumpled) will track poorly!

If you use the Cylinder Target, please wrap yours carefully around a can of the correct
size. Simply rolling the printout into a free-standing approximation of a cylinder will
create a target that will track poorly at first and deteriorate further as the paper cylinder
deforms with wear. A Cylinder Target must be formed into a cylinder of the correct
diameter to track properly!

**Documentation**

To help your user understand how your selection and manipulation techniques work, you should
provide two kinds of documentation: a written description and a video demonstration.
Written description
Write a brief document that describes how to use your system, including how to create and delete instances; how to select, position, scale, and rotate instances and the workspace coordinate system; how to seamlessly transfer an instance or the workspace between the ground and a planar toolbar or cylindrical wand; and how to assemble compound instances. This should also present the rationale behind your choice of techniques, discussed in context of what we covered in class and in the readings. Please include screen shots in your description, integrating them into the document. There is no minimum length; however, your document should fully explain how your user interface works and why you designed it that way. Your description should be submitted as a PDF file.

Video demonstration
Create a narrated video demonstration (at most four minutes in length) that shows your system in action. Since each of you will do things differently, your goal here is to make it as easy as possible for us to understand how your system works by seeing it work, to minimize the time that it takes us to learn how to use it ourselves. If you don't clearly show us the required functionality in your video, you can't assume that we will figure out how to do it on our own.

Please capture the video of your system by using another device with a camera, such as your laptop or another phone. (If you instead capture a video directly from the screen of the device running your app, this will put an added load on your device, slowing it down, and will also prevent you from showing both the augmented view through your camera and a third-person view that includes you manipulating the device on which your app is running.)

Please keep it simple: exotic visual and sound effects are neither needed nor desirable!

Share your video with us (but not with others!) as an unlisted YouTube video or using a private link on Vimeo, including its URL in your README file and the CourseWorks Upload comment. Please be sure that the date and time when the video was posted are consistent with when we should consider your assignment to have been submitted!

What to submit
Your submission should include your complete Unity project (but, no executables or Xcode project directories), your written description, and a link to your video demonstration. It is your responsibility to make sure that any file you submit is virus-free. Note, again, that any screen captures should be integrated into your written description, and not included as separate images. Each file should include your name and UNI at the beginning.

Your submission should include:
1. The entire Unity project folder compressed. (Do not include the app executable or the XCode project for iOS.)
2. Your written description PDF file.
3. A README file with the following information:
How to submit

Please compress all files in your submission into a single zip file (don't forget any needed data files), named "YourUNI_Assignment3.zip". Name your video "YourUNI_Assignment3", upload it to an unlisted YouTube video or a private Vimeo video, and include the URL in your submission, as described below. Then, submit through CourseWorks, following these steps:

1. Log into CourseWorks.
2. Select COMS W4172 in My Courses.
3. Select Assignments from the left-hand navigation pane.
4. Select Assignment 3.
5. Click the Submit Assignment button in the top right corner.
6. The Submit Assignments page will load. Choose your zipped project using the browse dialog window that appears after pressing "Choose File."
7. After choosing your project, copy the URL of your unlisted YouTube video or private Vimeo video into the Comments field beneath the File Upload section.
8. Press "Submit."

Now is the time to verify that you submitted your assignment correctly by downloading the zip file you just submitted to a location on your computer outside the directory tree where you did your development. Then, create a new project with which to test it and deploy it to your mobile device. Next, make sure that your uploaded video is playable through the URL you provided. Please don’t wait to do this! We will not accept excuses that an incorrect or damaged file was accidentally uploaded.

Please try to submit well before the deadline since CourseWorks can sometimes become busy and slow. You may use up to the number of late days you have left on this assignment, but remember that there is one more assignment still to come besides the final project.

Hints

Your grade for this project will be determined in significant part by the quality of the user interface that you implement, which will be evaluated using the Nielsen Usability Heuristics.
Please use an occlusion model for your toolbar (e.g., see How to Use an Occlusion Model with a Cylinder Target).

Be sure that after moving an instance to a new location, it will be possible to select it again using your selection technique!

Enabling Extended Tracking for a target can make it possible to work in a much larger volume. But, be sure you understand the constraints on which Extended Tracking relies when using it, including the kinds of targets that are supported and the need not to move that target.

Depending upon the targets you are using in this assignment, the angle with which they are viewed, and the lighting, the camera position and orientation computed by Vuforia can change noticeably from one frame to another, resulting in “jitter.” Consequently, the greater the distance between a vertex of an instance “attached” to an Image Target or Cylinder Target and the Image Target or Cylinder Target itself, the greater will be the variation in that vertex's position from one frame to another. For example, if you are implementing pointing with a visible “ray” emanating from an Image Target, this will be more evident in the jittering of the distant tip of the ray than in the base of the ray. In general, your scene will be more stable if (a) there are more completely unobscured, clearly viewed features that are seen simultaneously by your camera, (b) the vertices in your scene are closer to the targets that define their positions, (c) an Image Target or Cylinder Target is closer to the camera, and (d) an Image Target or Cylinder Target is not viewed from an extreme glancing angle.

Whatever you do, let Vuforia determine the position and orientation of any targets. That is, you should not require the physical camera to be at a fixed position and orientation relative to any target.

Tracking with optical targets is sensitive to lighting and the quality and rigidity of the printed target. A dirty, wrinkled, crumpled, folded, or curled target will significantly decrease the quality with which you can track. (Yes, we said this before!)

Avoid camera poses that view targets from oblique glancing angles that can cause the ink in the targets to appear shiny. Ensure that your Image Targets are perfectly flat. (You will get the best results if you attach them to cardboard or other firm material.) Please see Optimizing Target Detection and Tracking Stability, Attributes of an Ideal Image Target, Attributes of an Ideal Cylinder Target, How to Optimize the Physical Properties of Image Targets and Natural Features and Image Ratings guide.

You will get the best results working in a brightly lit environment.

If you’re having trouble implementing your desired interactions with targets, consider temporarily implementing “backup” debug interactions that use the touchscreen or 2D GUI control panels to verify the non-optical-tracking aspects of your code (e.g., your scene graph or transformations).
This will help isolate problems with general program logic, scene graph design, and 3D transformations that you might erroneously attribute to optical tracking.