COMS W4172: 3D User Interfaces—Spring 2014

Prof. Steven Feiner
Date out: February 10, 2014
Date due: February 25, 2014

Assignment 1: Unity in Space

Introduction

This assignment will be the first Unity project you develop on your own, after having completed Assignment 0 & Assignment 0.5. You'll be designing a virtual 3D solar system with which the user can interact using your mobile device's touchscreen. Your solar system should include a sun, planets, a satellite, and a spaceship.

Since we do not want you to have to create your own 3D models, you are free to download models from the Unity Asset Store, or any other source, providing you have permission and cite each source properly in your documentation. Alternatively, you can load a model (.fbx file, .obj file, or other supported formats) by dragging its file into the **Project** View in the Unity Editor Window. Any associated textures should be added to a "Textures" folder placed next to the loaded model in the **Project** View. You can also import an entire directory at once. (There are plenty of free models available, so you should *not* have to purchase any.)

Sun and Planets

Your solar system should have a sun and at least three planets. Each planet should be a descendant of the sun in the scene graph hierarchy (but, see the hint about hierarchy below).

Each planet should rotate about its own axis (the direction in which the axis points is up to you) *and* orbit the sun. Please note that your trajectories do *not* need to reflect the physics of a Keplerian solar system! In particular, you are welcome to make your orbits circular and of arbitrary angular velocity. (We'll talk more about rotation on Tuesday, February 11.)

Satellite(s)

Your solar system should have at least one satellite (natural or artificial) orbiting one of the planets. Each satellite (note that you only need one for the entire assignment) should rotate about its own axis (the direction in which the axis points is, again, up to you). Each satellite should be a descendant of its planet in the scene graph hierarchy (but, see the hint about hierarchy below).

Spaceship

The spaceship should be relatively small when compared to the planets, but should not be difficult to see (i.e., the ratio of its size to the sizes of the planets should be unrealistically large). The user should be able to control the spaceship's direction, orientation, and speed, as discussed below.

Lights
You will need at least one light in the scene graph in order to view the scene. (Note that the free version of Unity supports casting shadows from directional light sources only, so don't worry about shadows.)

Selection
The user should be able to select any object in the solar system (sun, planets, satellite(s) and spaceship) using your mobile device's touchscreen. (Direct selection using the touchscreen can be achieved using the Ray object. See the Physics.Raycast function and the ray casting tutorial.)

Once an object has been selected, you should change it in some way to indicate this visually. For example, the object could change color to a color that marks it as selected. (To put that another way, even though all the objects are different colors, when one is selected, you could make it turn to a specific color or do something else that indicates that it is the selected object.) Please think about which approach would be most effective.

Initially, no object should be selected. At most one object should be in the selected state at a time. You should also support some way to deselect the currently selected object without selecting another object. You are welcome to use an on-screen button to enable selection, to avoid accidental selection as the user navigates the scene.

Control Panels
Referring to the Unity GUI and GUILayout class documentation and the GUI Scripting Guide, create a partially transparent control panel for each object. These panels should be placed in a position of your own choosing. Each panel should be visible *only* while the object to which it belongs is selected. Thus, none of the object control panels should be visible when the application is initialized. The panels will contain controls for the parameters of the actions you assign to your objects (see next section for details).

There should also be one camera control panel, which should be visible at all times.

Object Actions

Sun
When the sun is selected, all planets and satellites should initially stop moving, pausing at their current pose. Once the sun is deselected, all planets and satellites should resume rotating around their axes and orbiting.

Planets and satellite(s)
When a planet or satellite is selected, you should give the user the ability (e.g., by interacting with the object's panel) to stop the object (including any descendants, such as a satellite) from rotating around its axis and orbiting, pausing at its current pose, and to start it rotating and

orbiting again. However, when the object is deselected, any paused actions should start. For example, when a planet is paused, the planet should stop rotating and orbiting, and any satellite of the planet should stop rotating and orbiting.

The user should also be able to set the speed and direction of the object's orbit around its ancestor.

A selected planet or satellite should be able to perform the following action controlled by its panel: The selected object (but *not* its descendants) must be able to rotate about each of a set of *three* orthogonal axes, but should only rotate about one of those axes at a time. (The axes can be those of any coordinate system you desire whose origin is at the center of the object.) Using the object's control panel, the user should be able to choose which one of the three axes the rotation should be around and the rotational speed (from zero to some preset value) and direction (counterclockwise or clockwise). Choosing a different axis should cause the object to begin rotating about that axis with the current rotational speed and direction, *starting from the object's orientation at the time at which the new axis is selected*. (That is, the object should not reset its orientation when a new axis is selected.)

Once the planet or satellite is deselected, it (and any descendants) should resume rotating around its original axis of rotation and resume orbiting, taking into account its orientation at the time at which it was deselected.

Spaceship
The spaceship should be able to move freely about the environment. The user should be able to control the spaceship using a GUI menu. If the user sets the speed of the spaceship to some non-zero value, it should begin moving along its forward axis. The user should be able to set the spaceship to any orientation.

If the spaceship "collides" with a planet, you should print something to the screen and reposition the spaceship in a "safe" location. (To accomplish this, please refer to the Unity documentation on trigger collisions and the video tutorial).

Camera
Your user should be able to place the camera in one of three modes:

Default mode: The camera should be initially located and oriented such that your entire solar system is visible. In this mode, the camera should only be able to move in the *XZ* plane, bounded by min-max *XZ* coordinates of your choosing. It should be controllable using a virtual analog stick or a directional pad. Note that Unity comes with a Virtual Joystick class (written in JavaScript), which you are welcome to use. To use it, import it using: (Menu bar) **Assets→Import Package→Standard Assets (Mobile).** Then, after it is imported, (Project View) **Assets→Standard Assets (Mobile)→Prefabs→Single Joystick.**

The camera should be able to rotate as well. The user should be able to control the camera's

pitch and yaw. The pitch should be limited to −80° to +80°, the yaw should be unconstrained, and the camera should not roll. When the user returns the camera to this mode, it should be placed at the position and orientation that it last had in the mode.

Planetary mode: The user should be able to "attach" the camera to any planet, such that the object is visible at the center of the camera's view at a preset distance of your choosing from the planet. When the camera is attached to a planet, the camera's position and orientation no longer need to be directly controlled by the user, as they were in Default mode. In addition, the camera should not be affected by the planet's rotation about its axis.

Spaceship mode: The user should be able to "attach" the camera to the spaceship, such that it is rigidly mounted to the spaceship, translating and rotating with it. When the camera is attached to the spaceship, the camera's position and orientation no longer need to be directly controlled by the user, as they were in Default mode.

Note that objects that are outside the bounds established by the camera's near and far clipping planes will *not* be visible.

Hints

Before starting this assignment, please note that there is an extensive collection of Unity Tutorials. We strongly suggest that you view the Roll-a-Ball tutorial. Additionally, you may want to look through the Unity reference manual, to get a better feel for the Editor.

Please see the Unity reference page on Input for a comprehensive overview of the functionality of both the desktop and mobile input classes. Please read through the document to familiarize yourself with the features and limitations of both classes.

To do this assignment well, you should think carefully about how you structure your scene graph hierarchy. Which nodes should you use and how should they be arranged in the hierarchy relative to each other? (The relationship between a parent and their children is important, while the order in which siblings are listed does not matter.) How should the transformations that you apply to your objects be composed to achieve the required effects? Begin with just a sun and one planet to experimentally verify that your approach works, so you can modify it early on if necessary.

Regarding hierarchy, understanding rotation is crucial here. Note that when an object is rotated, its descendants will also rotate. Therefore, if you want object *B* to act as if it were a descendant of object *A*, but not be affected by *A*'s rotation (e.g., to have a planet rotate at a different rate than its satellite), the easiest way to do this is to create Empty GameObject *A'*, make *A* and *B* both children of *A'*, where *A* is centered at *A'* and *B* is offset from *A',* and then rotate *A* and B individually. If you do this, transforming *A'* will transform all its descendants, but *A* and *B* can each have its own independent rotation. For example, if you use this approach to handle a planet and its child satellite, *A'* would be the parent of the satellite *B*, while *A* would be the planet you rotate.

Regarding model files: Some of the model files you find may be of an inappropriate scale relative to the other objects you're using. Therefore, be prepared to apply a scale transform to one or more of your models to bring them up or down to a reasonable size. In addition, note that some models may contain too many polygons for your mobile device to render your scene at a reasonable frame rate.

Regarding ray casting: When using the Physics.Raycast function, you will be returned a RaycastHit object. The RaycastHit object contains a reference to a Collider. The Collider, contains a reference to the GameObject to which it is attached. You can use that reference to determine what object was collided with, using the ray cast through the screen.

Regarding the Unity Scripting manual: The manual has been written to support three programming languages: C#, JavaScript, and Boo. *The programming language used for examples in the reference manual defaults to JavaScript, but can be switched through a drop-down menu available at the top of every page.*

Regarding rotation: In Unity, the "Rotate" *function* in the "Transform" class specifies a *relative* orientation change that will be composed with the current orientation. In contrast, the Rotation, LocalEulerAngles, and EulerAngles *variables* will set the *absolute* orientation (i.e., will override the current orientation). The Rotate function needs to be given the amount to rotate in the *X*, *Y*, and *Z* axes either as three separate floats or as a Vector3. It will apply a rotation to the object about the *Z* axis, *X* axis, and *Y* axis (in that specific order).

Regarding the Unity Virtual Joystick: Unfortunately, this asset is not well documented; but it is a useful feature. Please see the Developing with Unity and Vuforia guide on the TA wiki for instructions on how to use the Virtual Joystick asset. (The Virtual Joystick has a script written in JavaScript; however, you do not need to program with JavaScript in order to use the Joystick.) You are also welcome to create a Directional Pad instead, using standard GUI assets, or use any C# written Joystick equivalent that you write yourself or find on the internet (though you must provide credit).

Regarding Play Mode: There are two sets of importable Standard Assets: desktop and mobile. The mobile assets may use particular features in the mobile APIs that do not work on the desktop. An example of this is the Virtual Joystick above. While it works immediately on your mobile device, it will not work in Play Mode on your desktop. So while debugging, you will want to include secondary controls that are guaranteed to work on in Play Mode, such as Input.getMouseButton(). Since these controls may affect performance slightly, you will want to disable them when not debugging, or if you feel comfortable with your app's performance on your device, you can choose to support both modes.

A second note on Play Mode: Play Mode is a very useful way of debugging your app, but it is not fully indicative of how it will run on your phone. *Make sure you test fully on your device whenever you introduce a new feature.*

A note on textures: When you load in a model and associated texture in two separate load steps, the texture might not connect to model automatically. In order to connect the texture to the model, drag the texture onto the Texture box in the Material component of your model in the **Inspector** View.

What to submit

Your submission should include:

- The entire Unity project folder compressed.
    - Do not include the app executable (or the XCode project for iOS).
- A readme with:
    - Your name & UNI
    - Date of submission
    - Computer Platform
    - Mobile Platform & OS version & Device name
    - Project title
    - Project directory overview
    - Special instructions, if any, for deploying app
    - Instructions for using app
    - Missing features
    - Bugs in your code and Unity
    - Asset sources
- A *brief* video demonstrating your application's features.

How to submit

Please compress all files in your submission into a single zip file, remembering to include any needed data files. Submission will be done through CourseWorks using the following the steps:

1. Log into CourseWorks.
2. Select Drop Box from the left hand navigation pane.
3. Expand the COMSW4172_001_2014_1 Drop Box folder. You should see a folder with your name. If you do not, please email the TA.
4. Select Upload Files in the drop-down list to the right of your name.
5. The Upload Files page will load. Choose your zipped project using the browse dialog window that appears after pressing "Choose File."
6. After choosing your project, enter the display name. It should follow this convention: "YourUNI_Assignment1."
7. Press "Upload Files Now."

Please try to submit before the deadline, since CourseWorks can sometimes become busy and slow.

Remember, you can only use a *single* late day on this assignment, so start early! And, have fun!