Interactive Photo Displays
Source documentation

This document aims to give a look at the implementation of Interactive Photo Displays. The application is developed on the Unity game engine with C# as the programming language. This document is aimed at developers that are familiar with the basics of Unity development.

Prerequisites for development

The application has been developed with Unity version 5.4.1f1 on Windows 10. Newer Unity versions might also work but correct functionality can't be guaranteed. Unity can be downloaded for free from Unity's website.
The application requires a Microsoft Kinect v2 sensor for user input. For development the Kinect SDK must be installed. It can be downloaded for free from Microsoft's website.

Overview

Interactive Photo Displays is an application that displays user created sequences of sceneries, which consist of layered images with interactive elements. These scenery sequences are defined in a separate configuration file, which also contains other display properties. Configurations and sceneries are created and stored as text files in JSON format. The application uses user's position and hand gestures, tracked by a Kinect sensor, for controlling interactive elements.

Program flow

When the application starts, an instance of the ConfigurationLoader reads configurations from the JSON file that is given as a command line parameter. This creates a singleton instance of the Configurations class that any other instance can then refer to as needed.
An instance of the SceneryQueue class then starts to display a queue of sceneries defined in the configuration file.
Each time the queue advances and a new scenery is started, an instance of the Scenery class is created and any previous instance is destroyed. The Scenery instance reads the JSON file that was given in the configuration file and according to that loads the scenery. In the loading process the Scenery reads through all objects in the JSON file and creates corresponding scenery elements (SceneryImage, SceneryPopUp or SceneryText).
The BodyTracker class uses Kinect input to track user movements. The currently active Scenery instance uses the tracking information to move its objects such as images according to user movements.

Classes

This chapter describes the most essential classes and how they work together.

Configurations

Session specific settings are handled with the Configurations class. The configurations are loaded once at the start of the application.

Scenery objects

Scenery

The Scenery component represents a single unique scenery that contains and controls every element in the scenery, such as images and text.
A scenery is created by instantiating a copy of a prefab that contains the Scenery component and then calling the Scenery's LoadScenery method. The Scenery then reads the data from the given scenery JSON file and creates every scenery element specified in the file.
After the scenery is loaded, it starts to constantly read user input from BodyTracker. According to the input it controls the movements of its elements (specifically MovableSceneryObjects).

SceneryObject

All scenery related components that require data from the scenery files implement the SceneryObject interface. Implementing the interface means implementing methods for loading from (SetData) and saving to (GetData) a serializable class that inherits SceneryObjectData.
To summarize, a SceneryObject component should implement SetData and GetData and for those methods have its own SceneryObjectData inheritor to describe all the data that the component can load or save in the external text files.

Scenery elements

A Scenery consists of several distinct types of SceneryObjects, which in this document are referred to as scenery elements. These types of elements are the image, the text and the pop-up element. For each element there is a prefab of which a copy is created when the scenery is loaded. Each of these element prefabs has their distinct SceneryObject implementer components, which are described below.

SceneryImage

The SceneryImage component adds a visible image to the scenery element. This image is loaded from the path specified in the scenery JSON file. This path is a URL pointing to a local file or a file on the Internet. SceneryImage can alternatively load an OGV video file. The image is given to and then rendered by the SpriteRenderer component of the element prefab. The image can be rendered partially transparent with a transparency color and threshold.

SceneryText

The SceneryText component adds a visible text to the scenery element. The text is loaded from the scenery JSON file and can have basic text formatting.
The text and other data is given to and rendered by a Text component in the element prefab.

SceneryPopUp

The SceneryPopUp component adds a text container that can be opened and closed with user input.
The text in the pop-up functions similarly to SceneryText.
The container's background's color, transparency and size can be adjusted. The height of the container automatically adapts to the given width and the length of the text.
By default the pop-up is hidden behind a small icon on the screen and is displayed when a user's hand grabs the icon. The pop-up can then be hidden by grabbing it again. It can also be set to hide itself after a set time.
Alternatively the pop-up can be set to be always displayed without user activation.

Additional SceneryObject components

MovableSceneryObject

The MovableSceneryObject component allows Scenery to move the scenery element such as an image in order to create an impression of depth and movement.
A MovableSceneryObject has a starting position, scale and rotation. The Scenery constantly changes all its contained MovableSceneryObjects' positions and scales depending on the movements of the user and theirs starting states (position and scale).

AnimatedSceneryObject

The AnimatedSceneryObject component enables a waving animation for the image or text element. The animation has horizontal and vertical speed and magnitude properties. These properties are sent to the shader (Animated/Sprite or Animated/Text) used in the element's material.

User input

The program uses Microsoft's Kinect v2 motion sensor as user input. Interfacing with the Kinect is done with Microsoft's Kinect library for Unity.

BodyTracker

The BodyTracker component uses the Kinect library to read relevant user input data. It keeps track of each user detected by the Kinect and provides a simple interface for detecting users' movements and mapping user position data to more usable formats. The methods provided by the BodyTracker are then used by other parts of the program for acting upon user input.

Open Sound Control

The Open Sound Control (OSC) protocol is used to communicate with other applications. The UnityOSC library (https://github.com/jorgegarcia/UnityOSC) is used for creating connections and communication.

OSCMessenger

The OSCMessenger component uses the UnityOSC library to create a client for sending data and a server for receiving data. It waits for Configurations to be loaded so that the connections can be made on ports and IP chosen by the user.

Messages sent by OSCMessenger:

- When a new user appears in view of the Kinect, a message containing the user's ID (which is the tracking ID of the body given by Kinect) as a string is sent to address '/user/new'
- When a user leaves, a similar message with the user's ID as a string is sent to '/user/lost'
- All users' positions are sent in messages containing the user's ID, its x position, its y position and its z position one after the other as strings to '/user/location'. These location messages are sent with a frequency set by the user in the Configurations.
- When the scenery is changed, a message containing the scenery's index as an integer (starting from one) in the current queue is sent to '/scene/changed'
- When a pop-up element is activated and appears, a message containing the element's text content's first 64 characters are sent to '/information/shown'
- When a pop-up element disappears, a message containing the element's text content's first 64 characters are sent to '/information/hidden'

Messages listened to by OSCMessenger:

- When a message to ‘/scene/change’ is received, a new scenery is loaded from the current scenery queue at the index that is given in the message as an integer.