SYSTEMATIC TEST CASE GENERATION FOR AUGMENTED REALITY APPLICATIONS

Richard LaFranchi

MCS Student, Department of Computer Science, Colorado State University, Fort Collins, CO
rlafranc@rams.colostate.edu Colorado

Introduction

- What is Augmented Reality? - Software that Imposes Virtual Objects into the Physical World
- Very little research exists in testing as it applies to Augmented Reality (AR)
- **Research Question** - Can existing techniques for generating tests in 2D GUI applications be effective in AR applications?
- We've developed an approach for test generation based on existing research in testing 2D GUIs (Graphical User Interfaces)

Approach

Comprised of the following algorithms for generating tests:

- Depth-first Traversal Algorithm to Extract Structure of Virtual Objects (aka GUI Ripping) Algorithms 1 and 2 based on [2]
- Build an Event-flow Graph [1]
- Path Finding Algorithm to Generate Test Cases as shown in Algorithms 3 and 4 based on [1, 3]

Ripping Algorithm

Algorithm 1: Our modified version of the GUI Ripping algorithm

```
Input  : The AR Forest
Output: No output
1 Function DFS-AR (forest)
      /* topLevelNodes are the currently present virtual objects in the AR Window   */
2     for node in forest.topLevelNodes do
3         DFS-AR-Recursive(node)
```

Algorithm 2: Our modified version of the recursive GUI Ripping algorithm

```
Input  : A Virtual Object or Node in the AR Environment
Output: No output, structure of the forest is updated at each level
1 Function DFS-AR-Recursive (node)
2     if isExecutable(node) then
          /* Virtual objects may support one or more different types of events      */
3         for eventType in eventTypesFor(node) do
              /* Execute the event on the node                                      */
4             execute(node, eventType);
              /* invokedNodes() is a method that finds any new virtual objects in the AR
                 Window that appear                                                 */
5             children= invokedNodes();
              /* Update the structure so that invoked nodes are added as children to the
                 current node                                                       */
6             node.addChildren(children);
7             for child in children do
8                 DFS-AR-Recursive(child);
```

Event-flow Graph

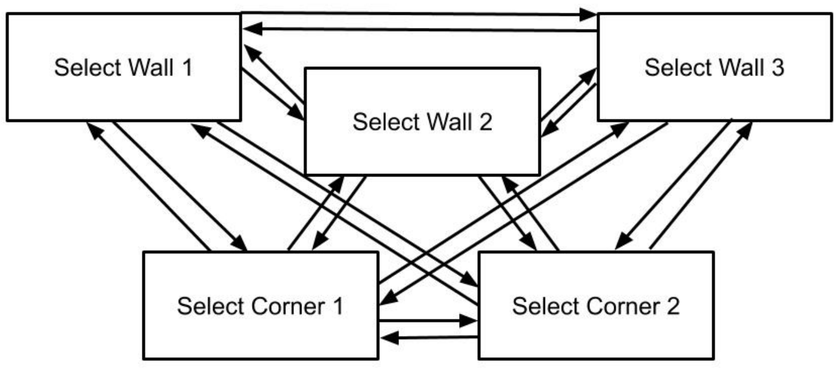

Figure 1: Example Event-flow Graph

Path Finding and Test Generation

Algorithm 3: Simple path finding Algorithm that finds all paths between a source event and a destination event in an event-flow graph

```
Input  : The event-flow graph, a source event, a destination event, a path list holding the sequence of
         events, a paths list holding a list of paths
Output: No output, the paths list holds the list of tests generated
1 Function getAllPaths (graph, source, destination, path, paths)
2     source.visited = true;
3     path.append(source);
4     if source = destination then
5         paths.append(path);
6     else
7         for neighbor in graph.neighbors(source) do
8             if neighbor.visited = false then
9                 getAllPaths(graph, neighbor, destination, path, paths);
10    path.pop();
11    source.visited = false;
```

Algorithm 4: Algorithm that loops through the top-level virtual objects and uses getAllPaths to generate the tests

```
Input  : The event-flow graph
Output: No output, paths list can be used to export data to a desired format
1 Function generateTestCases (graph)
      /* Last event in the graph is the destination                                 */
2     destination = graph.vertices[graph.vertices.length-1];
3     paths = [];
4     for node in graph.topLevelNodes do
5         path = [];
6         getAllPaths(graph, node, destination, path, paths);
      /* export paths to desired format                                             */
7     exportData(paths);
```

Case Study

- Approach was evaluated by developing a prototype of an iOS AR application using XCode, Swift programming language, and ARKit.
- Application acts as a physical measuring tool for measuring the walls of physical structures as shown in Figure 2
- A subset of 30 tests were executed manually to attempt to discover faults

Figure 2: AR Application of Virtual Walls Imposed on Physical Structure

Results

Building	Building A			Building B	
Prefix	2 Walls	3 Walls	4 Walls	5 Walls	**Total**
Tests Gen.	2	18	2880	1,814,400	1,817,300
Feasible	0	8	272	42,560	42,840
Executed	2	8	10	10	30
Faults	0	2	10	10	22
Meas. F.	0	2	3	0	5
Repr. F.	0	0	6	10	16
System F.	0	0	1	0	1

Table 1: Summary of Tests Generated

Conclusion

Our approach for test generation was determined to be effective, however more research in this area is needed in being able to automate execution and the end to end testing process.

References

[1] A. Memon. An event-flow model of gui-based applications for testing. *Softw. Test., Verif. Reliab.*, 17:137–157, 09 2007.

[2] A. Memon, I. Banerjee, and A. Nagarajan. Gui ripping: reverse engineering of graphical user interfaces for testing. In *10th Working Conference on Reverse Engineering, 2003. WCRE 2003. Proceedings.*, pages 260–269, Nov 2003.

[3] A. M. Memon et al. *Comprehensive Framework for Testing Graphical User Interfaces*. University of Pittsburgh Pittsburgh, 2001.