Master thesis oponnent's review – Bc. Petr Smrček

Ing. Michal Lukáč – michal.lukac@fel.cvut.cz – Department of Computer Graphics and Interaction, CTU FEE

The candidate's task in this thesis was to analyze and compare various scripting lanugages usable for scripting of computer graphics applications. The candidate has selected the languages Lua, JavaScript (proposed in the task formulation), C# and in addition formulated his own language, NativeScript. The crux of the comparison were supposed to have been the interoperability overhead and script execution times.

The thesis opens with a solid introduction into the concept and features of scripting languages, which provides a useful foundation referred to later in the text. The third chapter provides a more in-depth look into the scripting languages selected for testing.

I must pause at this junction and point out that the text sorely lacks an intermediate chapter where an enumeration of possible choices would have been presented. As is, we are left without the knowledge of what other scripting languages there are, whether or not they are used in graphics applications nor why they weren't selected.

Conversely, the candidate did not justify or explain his choice of tested languages. For instance, the choice of the C#/Mono stack is quite controversial, since off the top of my head I am not aware of a graphics application where this would be used for scripting. In contrast, Python is used extensively as a scripting tool in Blender, a prominent open-source 3D modelling tool, yet is not mentioned in the thesis even once. This is a quite serious oversight on the part of the candidate.

That said, the description and analysis of the languages which were selected is commendable. It is reasonably in-depth and focuses on features relevant more to the programmer responsible for embedding the language, which is after all the aim of this work.

Another controversial step the author took was deciding to implement his own scripting language. While this may be useful in certain scenarios, doing so falls squarely out of the scope of the original task formulation, and the justifications for having done so are weak when the extent of the implemented language is considered. I can imagine how doing so might have been useful if the goal was to compare the LLVM virtual machine against other virtual machines, or possibly as a baseline for measurements; although in the latter case it would have been more useful to simply measure the function call overhead of a native code implementation, since that would have been more relevant for the decision which parts of the application should be made scriptable.

I find no fault with the testing setup. There is a lack of pure computational performance test, but since such tests are heavily task-dependent, I consider the point simulation test a suitable substitute.

Chapter 6 is an outstanding example of dilligent testing and analysis of results. Not only does it present the measurement results, but the author conducted further testing to try and determine the causes of the various results, which is something seldom seen in such work. In section 6.3 a baseline comparison to a C++ implementation is added, which I would have appreciated also in previous tests, but that is merely a small remark. Section 6.4 concisely presents recommendation for various languages based on the testing results.

In conclusion, the candidate has **fulfilled** the task he had been charged with. While the work is not free of questionable decisions and serious flaws, it is an outstanding example of a diligent experimental comparison of the various systems.

In consideration of the above, I **recommend** the thesis for defence and grade it **B – very good**.

I propose the following topics for the discussion:

1. Speaking briefly, how does Python compare feature-wise to the selected languages? Can you also estimate its performance in the testing scenarios?
2. Which languages that are or could be used for scripting graphics applications did you **not** select for testing and why?
3. Figure 6.5 shows that, counterintuitively, the „optimized“ case of the point simulation test is actually slower than the „half-delegated“ case for both Lua and C#. Can you show whether this correlates with the number of transitions between application code and script, or some other measure? Which feature, common to the two but not appearing in JavaScript/V8 would you say causes this?

In Prague, 26th January 2016 Ing. Michal Lukáč