Abstract

This proposal is for a Blender addon that will facilitate model creation and editing while seeing those edits directly in PySoy. PySoy will run within Blender’s embedded Python interpreter and gain access to Blender’s Bmesh data structures while changes are made within Blender itself. Those changes will instantly be reflected in PySoy and displayed in a separate render window in the Blender GUI.

1 Intro

The implementation will be accomplished via a Blender addon tool written in Python. The addon will access the Blender Python API to accomplish this task. There should be no changes to the Blender C / C++ code base, as this tool needs to be self-contained and not require changes to Blender’s master. However, if that is needed, then changes to the C++ side will be worked out with the Blender developers. I currently attend Blender’s developer IRC meetings, and can ask about the appropriate requirements either there or in the bf-committers mailing list.

2 Project Details

The Blender addon will access Bmesh and related data structures in Blender and copy them to the PySoy internal data structures representing its models. This tool will provide a render window in Blender itself. An “update” button will be present in a panel dedicated to the tool. Information about data translation (i.e. vertex count, edge count) will be present in the PySoy window, thus allowing modelers to see if there are any discrepancies between the Blender copy and the PySoy translated copy. The addon will not support armatures for the summer work. Textures and materials will be supported.

Bpy.data contains an easy way to copy model geometry data from Blender. Additionally, bpy.context module is useful. Only read-only methods will be used in the addon.

I’ve done some previous prototyping with the Blender Python API:
Data reflection from Blender to PySoy should first involve creating a Blender object from the plugin itself via `bpy.data.objects.new`. From there, we can gather the verts and faces from `object.data.vertices` and `object.data.faces` (read-only functions) or via getting a handle to the mesh via `bpy.data.meshes[meshname]`.

Creating a panel and buttons in Blender is straight-forward and can be seen in the link to my Python code above.

Once the Blender plugin has copied the current mesh data, calls to the PySoy API can be made. For each update, the simplest approach is to remove the previous mesh and instantiate a new one with the new data. We create a new PySoy body and fill in the vertices and specify faces on the PySoy side.

It will be a bit more involved to run PySoy within Blender’s interface (and showing the updated model). However, it should be possible utilizing OpenGL from the Blender Python API. The Blender API allows drawing in its 3D view via the `bgl` module. We first need to reflect all PySoy rendering output to a buffer object. The frame can then be reflected to Blender via `bgl.glDrawPixels` / `bgl.Buffer`.

### 3 Beginning Milestones

1. Ramp up on both Blender Python API and PySoy internal data structures, i.e. build a simple application that uses mesh data from a .blend file and instantiates a PySoy body object with the mesh data.
2. Get PySoy successfully running in Blender’s embedded Python.

### 4 Midterm Milestones

1. Properly access Bmesh and model data after edits are made to model.
2. Provide “update” button in PySoy addon panel that will reload assets in PySoy at that time.
3. Render PySoy to a Blender 3D view.

### 5 Final Milestones

1. Get model data working in PySoy after data copy.
2. Iterate on GUI design, code, performance, etc.

### 6 Bio

I am currently a PhD candidate at the University of California at Santa Cruz. I have extensive professional programming experience in computer graphics. I have been a teaching assistant for two courses (over that last four years) that both teach Blender.
I have worked at two game companies and on other open source projects professionally. See my resume here: http://people.ucsc.edu/~behollis/hollister_resume_9.14.pdf