The Development of a 3D System for C-Snap

Shigeru Imai (imais@rpi.edu)
August 5, 2014

This document describes a brief design overview of a 3D system in C-Snap, the status of the project and possible enhancements. This project has been done over the summer of 2014. Readers of this document are expected to be knowledgeable about the background of the project and the basic software architecture of C-Snap.

1 Code Repository

The latest code is available at: `https://github.com/imais/Snap--Build-Your-Own-Blocks`. Note that the code described in this document is on the `real-3d` branch.

2 Design of the 3D System

2.1 Required Libraries

We use *three.js* (`http://threejs.org/`) to render 3D graphics. The following libraries have to be loaded in `snap.html` when running the 3D system in C-Snap.

- `three.js`: The main three.js library.
- `helvetiker_regular.typeface.js`: A font library required to render texts. Distributed as part of three.js.

2.2 System Architecture

To render 3D objects on the stage, we have added a new HTML5 canvas called `canvas3D` to the existing three layers of canvases for the Background Costume, Pen & Stamp, and 2D Objects as shown in Figure 1.

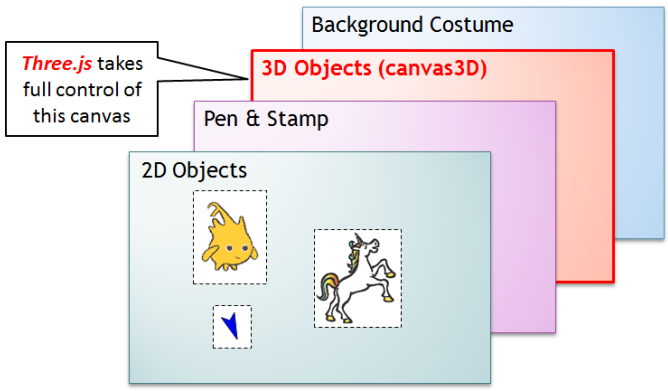

Figure 1. Layered canvases on the stage

Figure 2 is the system architecture of the 3D system in C-Snap. `SpriteMorph` and `StageMorph` manage 3D related instances as follows:

- `StageMorph` manages instances associated with the 3D space such as `scene, camera, light` and `renderer`. Note that `renderer` is configured to render 3D objects on `canvas3D`. For the initialization and rendering processes, refer to `init3D()` and `drawOn()` of `StageMorph` respectively.
- `SpriteMorph` manages instances associated with a 3D object such as `object, geometry`, and `map` (*i.e.*, texture). It receives commands from codelets and applies 3D geometrical transformations (*i.e.*, translation, scale, rotation) to the 3D object. It never renders to `canvas3D` directly.

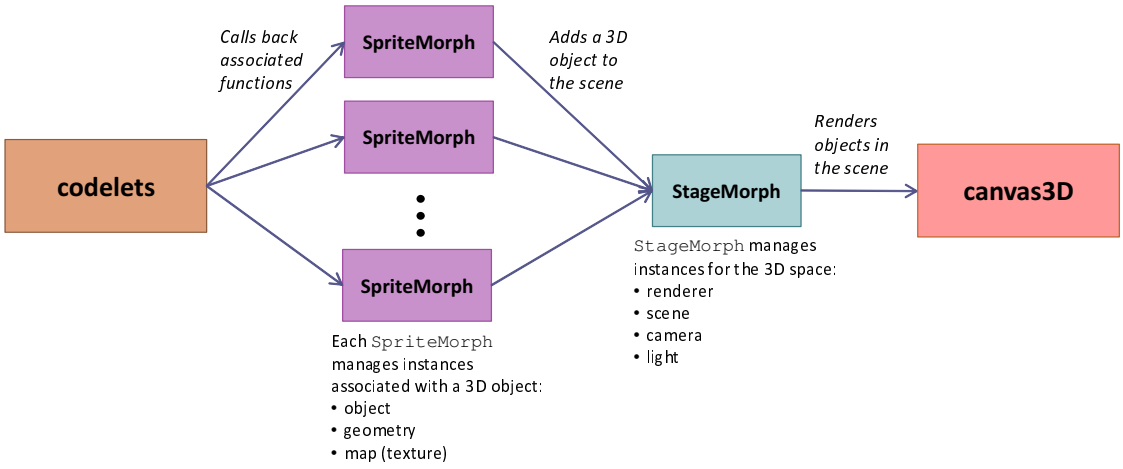

Figure 2. System architecture of the 3D system in C-Snap

2.3 Basic Rendering Sequence

`StageMorph` and `SpriteMorph` interact with each other to draw objects on the stage as shown in Figure 3.

3 Implementation Notes

- In `SpriteMorph`, `geometry` (3D Costume) and `map` (Texture) are loaded from a HTTP server. To avoid accessing the server repeatedly, these instances are cached.
- To dynamically change codelets (*i.e.*, blocks) depending on if the selected sprite is 2D or 3D, `blocksCache3D` is defined in addition to existing `blocksCache`.

  Also, when updating the codelets, `selectSprite()` from `IDE_Morph` is called; however, there may be a simpler way.
- `Costume` has flags called `is3D` and `is3dSwitcable`, which change their values as shown in Table 1. While `is3D` is used to identify if the costume is 3D or not, `is3dSwitchable` is used to identify if the costume is able to toggle between 2D and 3D. If the sprite does not wear any costume (*i.e.*, Turtle), it is regarded as if both flags are false.
- To differentiate "stamped" 3D shapes created by 3D pen from a sprite with a 3D costume, two lists `shownObjects` and `hiddenObjects` are defined in `StageMorph`.

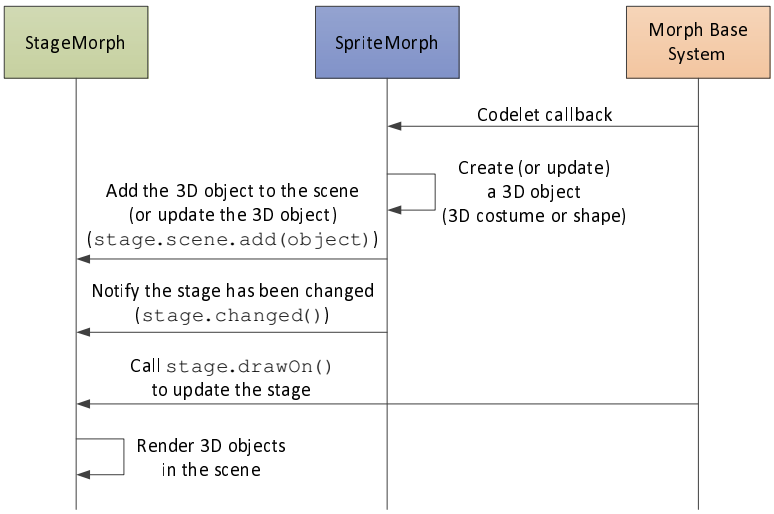

Figure 3. Sequence diagram of the rendering process

Table 1. Flags defined in `Costume`

Flag	State			
	No Costume	2D Costume (2D)	2D Costume (3D)	3D Costume
`is3D`	N/A	false	true	true
`is3dSwitchable`	N/A	true	true	false

While `shownObjects` is a list of rendered objects on the stage, `hiddenObjects` is a list of created, but not yet rendered, objects. The objects in `hiddenObjects` move to `shownObjects` when the "show pen" codelet is executed.

When the "clear" codelet is executed, `StageMorph` clears the objects in `shownObjects` from the scene and then clears both lists.

- The camera position is always configured to look at the origin $(0, 0, 0)$. It means that the center of the stage is always at the origin regardless of the camera position.

4 Status/Issues

Functions not implemented on time:

- Drag & drop
- Nesting (connecting multiple objects)
- Thumbnail display

Known issues:

- The "turn camera N degrees around X-Axis" codelet is not properly working.

- Once a texture is loaded for a 3D costume, the user can still select other textures, but cannot go back to the original 3D costume without a texture.
- When rendering many 3D objects, processing speed becomes an issue. For example, a knitting application in Figure 4 renders a number of small 3D knots. This application consumes almost 100% CPU even after the rendering and the browser does not respond anymore (it seems three.js code is running even after the codelets execution).

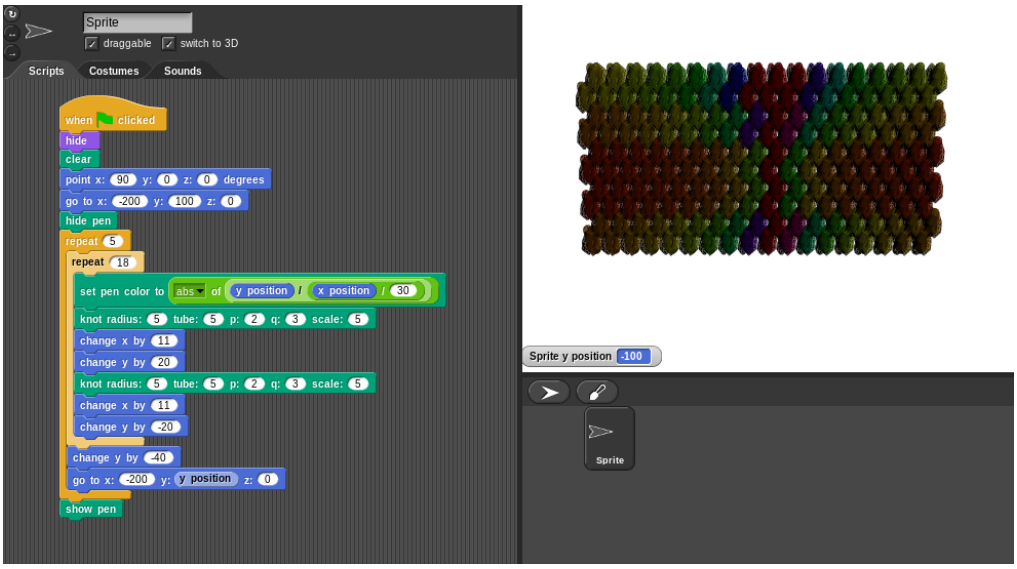

Figure 4. A knitting application

- When saving projects, 3D related states are not saved.
- When a 2D object switches to 3D, it should rotate around the center of the object, but currently it does not work that way.

5 Possible Enhancements

- Adding a user interface to (de)select textures.
- Interactive camera position change with mouse movements (just as implemented in the original *VirtualWigwam* application).
- Using GPU to speed up the rendering process (*i.e.*, using `WebGLRenderer` instead of `CanvasRenderer`).
- Drawing 3D lines with the 3D pen.
- Making the initial costume (Turtle) 3D-switchable. It is not very intuitive that we have to wear 3D costumes every time we want to use 3D features.
- Adding trigonometry functions (*i.e.*, $\sin, \cos, \tan$) to move 3D objects in a circle or a sphere.
- 3D versions of the "move" and "glide" codelets currently available in 2D.
- 3D physics simulation (very difficult!).