Part B: Explanation of the Penguin

Objects and The Scene Tree

The penguin is rendered by traversing a tree of scene nodes representing the polygons used to draw the penguin. Each polygon has a transform relative to its parent, with some additional time-dependent transformation function to animate it, as well as an easing function to smooth out that animation.

Main Control Loop (`animate`, `display`)

The main loop repeatedly renders the scene at 40fps, calling `animate` to update the animation parameters, and then `display` to draw the scene. `animate` performs some timing code to track elapsed time, updates the UI, and calls `update` on the root element of the scene (see below). `display` clears the screen for a new render, then calls `render` on the root element in the scene.

Actions on an Object (`update`, and `render`)

update

Animation is entirely time-dependent, so `update` just increments the internal time of the scene node by the specified amount, and repeats that process for each child, each child's child, and so on.

render

Each object is rendered during the render call of its parent, so it can inherit the appropriate transformations.

First, a new transformation matrix is pushed onto the stack, and the base transform is applied. This brings the working space into the reference frame of the object. The animation easing function is then applied to the current time, and used to transform the object's local coordinates according to the current time within the animation. The polygon is then drawn in this reference frame, the process repeated for each child, and the transform popped from the stack.

In order to allow children to be rendered behind their parent, the opengl depth buffer is enabled in the initialization of the program. Some children are inserted into the scene tree muliple times

Controls

Each part of the penguin in the scene tree has rotation, translation and scale controls.

Justification

While this is somewhat more complex than simply calling a series of nested rendering functions for each element of the penguin, I argue that this added complexity is more than made up for by the reduction of duplicate code, the ability to automatically create user interface elements for objects, and the simple declarative way of defining what objects will end up on the screen.