Computer Science 112
The Art & Science of Computer Graphics
The College of Saint Rose
Spring 2013

Topic Notes: Advanced Mead Functionality

The `loft` Function

We next consider a generalization of the `extrude` function, called a `loft`. A loft operation is basically a multi-step extrude, where we specify a series of cross-sections of the model, and the loft connects them up pairwise with extrudes.

Mead Example: `LoftedSpring`

Here, we construct a spiral spring using a loft. To loft a material across n cross sections we would specify:

```
(loft xSect1 xSect2 xSect3...xSectn)
```

where each `xSect` is a polygon in three-dimensional space.

The return is the lofted object (a `Mesh`) that covers the cross sections. The ends are capped with the polygons as described by `xSect1` and `xSectn`. We can use the resulting mesh just like we use meshes construted by other means.

However, it is often more convenient to define a list of cross sections, such as:

```
(define frame (list xSect1 xSect2 xSect3...xSectn))
```

To use this list as the set of parameters to `loft`, we essentially need to remove all of the items from that list and place them as parameters. Fortunately, this is a common operation and can be done using Scheme's `apply` function:

```
(apply loft frame)
```

The result of the `apply` is to take the list specified as its second parameter (`frame`) and use those as parameters to a call to its first parameter (`loft`).

For this specific example, we will build up our list of cross sections using a recursive function reminiscent of `multiAdd`. Here, the function `multigon` take a polygon and transforms it (initially with `initXform` and then successively with `delXform`) into a list of `n` copies that, ideally, form the frame of the object we wish to construct.

Note that we are building and returning a list. In the base case (zero polygons remaining to be added to the frame) we simply return an empty list. The ultimate list is constructed, one polygon at a time, using the `cons` function. This takes a polygon (which becomes the `car`, *i.e.*, `first` of our list) and a recursively constructed remainder of the frame (which becomes the `cdr`, *i.e.*, `rest` of our list).

Now, we need to constructing a cross section for our spring. Our cross section is a flat rectangle, with the hope that the spring looks something like a slinky.

```
(define springXsect
  (2to3d '((-10 -1) (10 -1) (10 1) (-10 1))))
```

We then use `multigon` to build the frame (`springFrame`) that we loft to form one "round" of the spring (`springLoop`), then add multiple rounds (using our original `multiAdd`) to build up the full spring (`ourSpring`). Notice that each round of the spring rises 36 units, so each successive round must be translated up by that amount.

This example goes a step further by doing a simple animation of the spring in motion. (Can you see the flaw in the approach taken?)

Atmospherics

POVray, and hence Mead, has the capability to add atmospheric effects: fog.

In Mead, atmospherics are controlled by messages sent to the `Image` class.

There are three messages you can send to your `image` to control fog:

- `(fog)` – turns on fog
- `(fogAttenuation n)` – sets the density of the fog, the default is 100 (which is pretty dense), lower numbers are more dense
- `(fogColor rgb)` – sets the color of the fog to the given RGB color, the default is `white`

Note: image quality must be set to 10 or higher, or fog will not be rendered.

This is demonstrated in a simple example:

Mead Example: `Fog`

One other item of interest in this example is the use of a `multiAdd` within a `multiAdd`, taking advantage of the fact that the inner `multiAdd` returns a `Group` that we can use as the items to be added by the outer `multiAdd`.