F97 6.837 Asst 3A – Object Modeling

This week's lectures address object modeling. Geometric primitives (polygons, spheres, polyhedra, etc.) can be generated procedurally and/or combined to form quite complex modeled objects. This assignment challenges you not to craft just one such object, but to **design a family of objects** indexed by any number of **parameters** which you define. Parameters might control complexity (for example of fine detail or global scale), inclusion (for example of certain sub-objects), creative or articulated aspects (for example, facial expressions, joint angles), or any other properties of which you wish to allow variation and specification.

This assignment challenges you to **write a program which outputs a different Inventor file for each set of command-line arguments it is given**. For example, your program might be called `myuid_road`, and be invoked with `-paved` or `-dirt`, with `-curvy` or `-straight`, or with `-flat` or `-bumpy` or `-steep`, or with `-miles 1` or `-miles 5`, or with `-billboards` or `-guardrails` (or both). Depending on the command-line, your program should output an Inventor object which is a member of the "object family" you have designed. This object is distinguished from others by the inclusion or exclusion of certain components (e.g., guardrails); by its complexity or size (e.g., miles of road); by its surface appearance (e.g., paved, dirt, etc.); and by its geometric configuration (e.g., flat, hilly, bumpy, etc.).

Modeling Conventions: Unless the command line specifies otherwise, your object should **occupy a cube of side one centered at the origin**. The object should be constructed so that its "Up Vector" is the positive z-axis, $\hat{\mathbf{z}} = (0, 0, 1)$. Face normals should be "outward" pointing. Note that you can model your object in whatever coordinates you choose; however, afterward you should enclose it inside a suitable transformation so that it respects the convention above.

Produce objects of reasonable complexity. Your object should render smoothly – say, at ten Hz or more – on an Athena SGI machine. This is a polite practice, as it will prevent instancers of your object from choking their renderers with too many polygons.

The program executable should be your athena user ID `uid`, suffixed by a reasonably short name describing the object – e.g., `jslevene_rocket` or `gupta_phone` or `lweisman_TA`. This convention guarantees everyone credit for their work, describes the object which will be emitted, and guarantees uniqueness (for next week and beyond, when all 100-plus executables will reside in a common class directory).

Your program, when run with the argument `-help`, **must produce an extended help message (i.e., documentation of its purpose and operation)**. If run with no arguments, it should produce a default object – a member of your object family, of course. Your program must support at least three kinds of parameters on the command line, which "index" your object family. To make life easier for users of your objects, you may wish to include sizing, orientation, or placement directives on the command line as well. (Such directives would of course override the default requirement stated above.)

The Point:
The goal of this assignment is to elicit **100-plus programs** (one from each class

member), each of which generates interesting, creative parametrized objects. Thus, your creative power will be multiplied a hundredfold and distributed to your classmates! On Tuesday Sept. 30, we will have an in-class exhibition (with color prints) of some of your objects. A week from today, Assignment 3B will ask each of you to instance each others' objects, then situate the resulting geometry in a full scene of your own making (due Oct. 3).

Development:

`uid_object.c` (Example program)

`arg_parse.[ch]` (Courtesy Paul Heckbert at CMU)

`Makefile`

Viewing Tools as described in lecture:

`/usr/sbin/ivview` (simple Inventor viewer)

`/usr/demos/Inventor/gview` (scene graph viewer, editor)

`/usr/sbin/SceneViewer` (sophisticated Inventor viewer, editor)

To turn in (details TBA):

Your C or C++ source file `uid_object.c` (please do not code in, or hand in, any other source files);

Your **stripped** executable file `uid_object` (remember, we will invoke `uid_object -help` to read your documentation); and

At least four examples of valid Inventor files produced by invocations of your program, named `uid_object.N.iv` where `N` is `1,2,3,4....` Each file should have an embedded comment echoing the command line which generated it (as shown in `uid_object.c`). Strive to make your objects interestingly different, to show off the "expressiveness" of your parameter space.

Of course, you should substitute your uid and the name of your object wherever `uid` and `object` are used above!

Assignment 3A is due 5pm Friday, September 26, 1997.