# Introduction to Computer Graphics

# Assignment Four

October 14, 2003

U-grad due date: Tuesday, December 9, 2003, 11:59 PM
Grad Student due date: Tuesday, November 25, 2003, 11:59 PM
(U-grads doing project should shoot for Grad student due date)

Credit: Ugrad 20 points (Relative, and roughly absolute weighting)
Credit: Ugrads doing projects 10 points (Relative, and roughly absolute weighting)
Credit: Grad 8 points (Relative, and roughly absolute weighting)

NOTE: The GRAD and UGRAD/HONOR'S VERSION IS DIFFERENT. The GRAD/ HONOR'S version is **EASIER**, to allow extra time for project work.

In this assignment we will improve "pp-world" even more. The same rules regarding input and output and program exit from the previous three assignments apply.

There are two parts to this assignment. First, we will add a few extra features. Then we will add ray tracing directives. In general, think of the default ("regular") display as a indication to the user about what to expect when the slow ray tracing step is finally taken. Note **extra deliverables** at the end of the assignment.

**Non-project people ONLY**
Add a sphere command. Parameters are x,y,z, and radius. Implement an approximation of a sphere as a collection of polygons. You can use any collection of polygons that resembles a sphere. Try to use a simple strategy! The main requirement is that a sphere must be clearly distinguishable from a stretched/rotated box. You can give your spheres default colors.

Scaling of a sphere with x,X,y,Y,z,Z should scale the sphere uniformly (don't create an ellipsoid, unless you want to do so for extra credit, in which case, go for it, but let us know that you are doing so in the README. Of course, a complete job would require an "ellipsoid" command as well, but this would be a secondary extra).
**End part for non-project people**

The light command

light <x> <y> <z> <r> <g> <b>

now gives the location as well as the direction of the light. In regular mode, you can assume that it is a point source. In both modes, ignore how far it is (unless you wish to play with it for fun and possible bonus points). In regular mode, compute its direction relative to a polygon based on the center of polygon. It would be best to use the center of mass of the polygon (why?), but the average of the vertices is acceptable. In ray-trace mode, you will have to give a bit of thought about how to handle the point source nature of the light. (If you like, you can make it a sphere of user settable size. If this is the case be sure to explain the merits and limitations of your enhancements in the README file). For minor extra credit, implement additional lights.

The "ambient" command from assignment 3 remains as is, and affects both regular and ray traced output.

Add a command:

specular <v> <n>

<v> is an integer from 0 to 100 which expresses a percentage of the light to be added via the Phong model, and <n> is the exponent in the Phong model. The command applies to every surface of a preceding box, or the preceding sphere. In regular mode, apply the specularity to the entire surface, based on the center of the surface. Arguably it would be best to use the center of mass of the polygon, but the average of the vertices is acceptable.

Add a command:

mirror <v>

<v> is an integer from 0 to 100 which expresses a percentage of the light to be reflected. The command applies to every surface of a preceding box, or the preceding sphere. In regular mode, add a bunch of white to the surface (proportional to <v>) to indicate to the user that this surface is a mirror.

The user interactions specified in assignment 3 should be in place and made to work where applicable. **Non-project people:** Make sure the they now work with spheres, including the ability to add them. Rotations of spheres are a no-op, but stretching should work.

**Non-project people only**

You should add the ability to make things specular and have mirror reflections using picking and user input. If you like, you can just have a few simple mirror and specular options such as not-a-mirror, poor-mirror, and good-mirror, but you should provide at least two levels of specularity strength, and two levels of specular sharpness (controlled through the exponent). If you choose to provide minimal capability, you need to make sure that the choices given provide noticeable and varied effects. Since I am purposely leaving this part of the interface up to your imagination, be sure to put the user incantations in the README file.

**End non-project**

Implement “course-render” and a “render” menu items. These render the world using ray tracing. For “course-render” you can use any shortcut you can think of to make it faster. This will help you debug your code and your users debug their scenes. For example, you could shoot one ray for every block of 4 or 9 pixels, and you could severely restrict the ray-tracing depth. “render” shoots at least one ray per pixel. If “render” works within a few seconds, then “course-render” can be considered optional.

(Semi-optional). Implement “save” which saves the current view as a tiff (or other image format) file. I assume that this is relatively easy to arrange, and will help produce the image deliverables described below. However, if this is not easy, then screen shots are acceptable. Linux users who want to make screen shots may want to call up the man page for the ImageMagick “import” command.

**Extra credit**

If you would like to improve on the program, be sure to explain what you did in the README file, and it will be considered for extra credit.

**Deliverables**

You must electronically submit a README containing any relevant information, but at a minimum, your name; an executable (called a4); and a src directory containing source files and a Makefile which can be used to build the executable. Also, submit at least one input script files and a resulting image files (i.e., tiff or jpeg, or something similar). For example, submit 1.input, 1.tiff. These images should show off your program, and at a minimum, show diffuse reflection, specularities, shadows, and mirror reflection.

The program must compile and run on one of the graphics machines (gr01, ... , gr10). Put in the README file the machine which you have verified this on.

Note that the graphics machines can be booted into Windows by people in the lab, so that it is possible that if you are working remotely that you will need to try more than one. We encourage students to use the higher numbered machines for Windows (7 through 10), but this cannot be enforced.

The turnin name is cs433_hw4.