Computer Graphics
HW3: Ray tracing.

In this assignment you will write a basic ray tracer. A substantial amount of starter code is provided in a template program called "ray" which takes care of many of the low-level details of parsing, data structures, transformations, etc., as well as containing high-level functions outlining the ray tracer. Your job is to fill in missing code in key functions to complete the implementation. Every place in main.cpp that requires additional code or some modification is marked with the comment FILL IN CODE. You may make support functions and types as necessary, but try to make all changes in ray.cpp rather than udray.cpp or udray.h. In your README any changes made elsewhere must be noted. You are also free to implement or use any exisiting 3D vector class and implement a C++ version of the ray tracer. You will receive 10% extra credit if you implement your own C++ version of the ray tracer (but your code should use the exact structure of the template we provided).

Part I: Due May 2, 2017

1. Implement a new rendering style to visualize the depth t of objects in the scene. Two input depth values specify the range of depth values which should be mapped to shades of gray in the visualization. Depth values outside this range are simply clamped.

2. Complete DIFFUSE section of shade_ray_diffuse(); Add sphere intersection testing in intersect_ray_sphere(). This function should parameterize the Intersection object returned (with material information, normal direction, etc.) like intersect_ray_glm_object() does

3. Scene complexity and creativity
As the last point implies, you must not only implement the ray tracer but also create a scene that shows off its functionality. Do this by creating a custom.scene file (see below for format) to arrange objects, place lights, and choose a camera angle. Submit the image that you produce as a .ppm or your favorite image format, along with your scene description file. The scene you create should demonstrate as much functionality as you implement, including diffuse shading, shadows, and reflections (to a depth of at least 2 reflections) on multiple objects and spheres. All submitted images should be rendered at a resolution of at least 500 x 500, although a much smaller image size is recommended while you are debugging.

Part II: Due May 9, 2017

4. Complete shade_ray_local(), which adds specular and shadow effects. This may call shade_ray_diffuse();

5. Complete reflection component of shade_ray_recursive() • Add support for refraction in shade_ray_recursive(). You may assume that there is no nesting of transparent objects (this means that spheres should be regarded as solid) and that the medium outside the objects is air

Part III: Due May 16, 2017

6. Implement a bounding volume hierarchy to accelerate ray tracing.

.scene file format

There are several kinds of commands contained in a .scene file type, each appearing on a separate line. Whole lines can be commented out by starting them with '#'. Here are the command types:

1. Camera position: **camera x y z dx dy dz upx upy upz.** Always the first command in the file; arguments work like **gluLookAt()**.

2. Clip planes: **clip left right bottom top zNear zFar**. Always the second command; arguments work like glFrustum().

3. Image dimensions: **image width height**. Third line in file. By reducing the image size, you can debug your code without waiting too long for it to run.

4. Light(s): **light x y z amb_r amb_g amb_b diff_r diff_g diff_b spec_r spec_g spec_b**. One light per line.

5. .obj object(s): **obj path_to_file x y z sx sy sz rx ry rz amb_r amb_g amb_b diff_r diff_g diff_b spec_r spec_g spec_b shine IOR reflectivity transparency**. One object per line. (x, y, z) positions the object and (sx, sy, sz) scales it. Rotations (rx, ry, rz) are read but not currently implemented. Subsequent values parametrize material properties (overriding any that may be in the .obj file itself).

6. Sphere(s): **sphere x y z radius amb_r amb_g amb_b diff_r diff_g diff_b spec_r spec_g spec_b shine IOR reflectivity transparency**. One per line.

Here is an example .scene file (included with udray code above). It makes the following images with different shading options (these are set at end of trace_ray():