For this assignment you will implement a basic ray tracer. Use the base-code provided on the class web site as a starting point. Percentages indicate the weight of each problem for grading. The assignment includes mandatory problems and a hacker’s bonus. You will score 100% if you solve all mandatory problems. The hacker’s bonus gives you additional points that will be included to determine your overall grade for the class.

Your solution that you hand in needs to include your source code with documentation and rendered images. The code documentation consists of comments in the source code and a readme file that describes the classes you modified. Test your implementation using the sphere, teapot, and bunny scenes from the class web site. Include a rendered image that demonstrates the solution of each problem. Please name the images accordingly. Send all files in a compressed archive to turnin@graphics.ucsd.edu before the deadline.

We will compile a gallery of the best images from each assignment and show it on the class web site. From each assignment, we will choose an outstanding image whose author will have the chance to present it in class.

1 Generating Camera Rays, 15%

Write code to generate camera rays using a camera specification based on a look-at vector, an up-vector, a vertical field of view, and an image aspect ratio. Your code should be able to handle arbitrary image resolutions.

2 Shapes and Ray-Surface Intersection

2.1 Spheres, 10%

Extend the sphere class by implementing ray-sphere intersection. Write code to compute the surface normal at the intersection point.
2.2 Triangles, 15%

Extend the *triangle* class by implementing ray-triangle intersection. Write code to compute the surface normal at the intersection point.

2.3 Instancing, 15%

Write a specialization of the *object* class to support shape instancing. Your code should allow the transformation of instances by transforming rays. It should correctly compute the corresponding transformation of normal vectors. Test your implementation by rendering an ellipsoid as a transformed sphere.

3 Shading, 15%

Implement a material that contains diffuse, specular, and ambient terms. The material parameters should include coefficients that modulate diffuse, specular and ambient reflectance, and the Phong exponent. The material should be able to deal with multiple point lights.

4 Shadows, 15%

Implement shadow rays to render shadows.

5 Reflection, 15%

Extend your material model to include specular reflection. You will need to cast rays recursively to follow the reflected light path.

6 Hacker’s Bonus: Refraction, 15%

Add specular refraction to your material. Include Schlick’s approximation for Fresnel reflection.