6.837 Intro to Computer Graphics
Assignment 4: Shadows, Reflection & Refraction

This week, we add global illumination to our Ray Caster. Because we cast secondary rays to account for shadows, reflection and refraction, we can now call it a Ray Tracer. You will encapsulate the high-level computation in a `RayTracer` class that will be responsible for sending rays and recursively computing colors along them.

To compute cast shadows, you will send rays from the visible point to each light source. If an intersection is reported, the visible point is in shadow and the contribution from that light source is ignored. Note that shadow rays must be sent to all light sources. To add reflection and refraction effects, you need to send secondary rays in the mirror and transmitted directions, as explained in lecture. The computation is recursive to account for multiple reflections and or refractions.

For debugging we provide you with code to visualize the ray tree created while computing the color of a single pixel.

Tasks

- Expand your Material classes to store the reflective & transparent multipliers and the index of refraction, which are necessary for recursive ray tracing. The prototype for the `PhongMaterial` constructor should be:

  ```
  PhongMaterial::PhongMaterial(const Vec3f &diffuseColor,
                               const Vec3f &specularColor,
                               float exponent,
                               const Vec3f &reflectiveColor,
                               const Vec3f &transparentColor,
                               float indexOfRefraction);
  ```

  You should also implement appropriate accessor functions for these fields.

- Create a new class `RayTracer` that computes the radiance (color) along a ray. Update your main function to use this class for the rays through each pixel. This class encapsulates the computation of radiance (color) along rays. It stores a pointer to the `SceneParser` for access to the geometry and light sources. Your constructor should have these arguments (and maybe others, depending on how you handle command line arguments):

  ```
  RayTracer(SceneParser *s, int max_bounces, float cutoff_weight, bool shadows, ...);
  ```

  The main method of this class is `traceRay` that, given a ray, computes the color seen from the origin along the direction. This computation is recursive for reflected or transparent materials. We therefore need a stopping criterion to prevent infinite recursion. `traceRay` takes as additional parameters the current number of bounces (recursion depth) and a ray weight that indicates the percent contribution of this ray to the final pixel color. The corresponding maximum recursion depth and the cutoff ray weight are fields of `RayTracer`, which are passed as command line arguments to the program. Note that weight is a scalar that corresponds to the magnitude of the color vector.

  ```
  Vec3f traceRay(Ray &ray, float tmin, int bounces, float weight,
                 float indexOfRefraction, Hit &hit) const;
  ```

  To refract rays through transparent objects, `traceRay` is also passed the `indexOfRefraction` (see below), and returns the closest intersection in `hit`, which is used to create the depth & normal visualizations. You can test your code at this point with examples from previous assignments.

- Add support for the new command line arguments: `-shadows`, which indicates that shadow rays are to be cast, and `-bounces` and `-weight`, which control the depth of recursion in your ray tracer.

- Implement cast shadows by sending rays toward each light source to test whether the line segment joining the intersection point and the light source intersects an object. If there is an intersection, then discard the contribution of that light source. Recall that you must displace the ray origin slightly away from the surface, or equivalently set `tmin` as *epsilon*. Note that in this naive version, semi-transparent objects still cast opaque shadows. Implement something better for extra credit.

- Implement mirror reflections for reflective materials (`getReflectiveColor()` > (0,0,0)) by sending a ray from the current intersection point in the mirror direction. For this, we suggest you write a function:

  ```
  Vec3f mirrorDirection(const Vec3f &normal, const Vec3f &incoming);
  ```

  Trace the secondary ray with a recursive call to `traceRay` using modified values for the recursion depth and ray weight. The ray weight is simply multiplied by the magnitude of the reflected color. Make sure that `traceRay` checks the appropriate stopping conditions. Add the reflected contribution to the color computed for the current ray. Don't forget to take into account the reflection coefficient of the material.

- Implement transparency effects by sending rays recursively in the refracted direction. If the material is transparent (`getTransparentColor()` > (0,0,0)), trace a new ray in the transmitted direction. We suggest you implement a function `transmittedDirection` that given an incident vector, a normal and the indices of refraction, returns the transmitted direction.

  ```
  bool transmittedDirection(const Vec3f &normal, const Vec3f &incoming,
     float index_i, float index_t, Vec3f &transmitted);
  ```

  We make the simplifying assumption that our transparent objects exist in a vacuum, with no intersecting or nested refracting materials. This allows us to determine the incident and transmitted index of refraction simply by looking at the dot product between the normal and the incoming ray. You may assume that the camera is always placed outside of transparent objects.

  However, be careful about the direction of the vectors and the ratio of refraction indices. Because we now consider transparent objects, we might hit the surface of a primitive from either side, depending on whether we were inside or outside the object. The `-shade_back` command line option should be used when there are transparent objects in the scene.

  The dot product of the normal and ray direction is negative when we are outside the object, and positive when we are inside. You will use this to detect whether the new index of refraction is 1 or the index of the hit object. Also, the index of refraction for the material surrounding the ray origin is passed as an argument to `traceRay`.

- We provide a `RayTree` visualization tool to help you debug your recursive ray implementation. This tool is activated by pressing the '`t`' key within the OpenGL previsualization. Insert the following commands within your `Raytracer::traceRay()` method as appropriate:

  ```
  void RayTree::SetMainSegment(const Ray &ray, float tstart, float tstop);
  void RayTree::AddShadowSegment(const Ray &ray, float tstart, float tstop);
  void RayTree::AddReflectedSegment(const Ray &ray, float tstart, float tstop);
  void RayTree::AddTransmittedSegment(const Ray &ray, float tstart, float tstop);
  ```

  The main ray is visualized in gray, shadow rays green, reflected rays red and transmitted rays blue. You will not create an instance of `RayTree`, but rather just call the static member functions. The calls will look something like:

```
RayTree::SetMainSegment(myRay,0,t);
```

Most of the time these function calls do nothing. But when the `RayTree` is activated, the segments of the ray tree will be recorded. The `glCanvas::initialize` method now requires a third argument:

```
void initialize(SceneParser *_scene, void (*_renderFunction)(void), void (*_traceRayFunction)(float, float));
```

You'll need to provide a function that takes in two floating point numbers, the x and y coordinates in screenspace (0,0) -> (1,1) of the current mouse position, and traces the ray through the scene. Your call to `traceRay` from within `traceRayFunction` should look very similar to your call to `traceRay` from within your main rendering loop. Note: all the activation/deactivation of the `RayTree` is taken care of in `GLCanvas::keyboard()`.

- rayTree.h
- rayTree.C
- glCanvas.h
- glCanvas.C

- Incorporate the updated code for point light sources and verify that your shading and shadowing implementation correctly renders these lights. The `PointLight` source method `getIllumination()` method handles the distance attenuation term needed for Phong shading. The physically correct attenuation term, $1/d^2$, is not always used for raytracing because it results in images with high contrast that are difficult to properly display on low dynamic range devices. Often 1/d or constant attenuation is used instead (see examples below).
  - light.h
  - light.C

Hints

- You do not need to declare all methods in a class virtual, only the ones which subclasses will override.
- Print as much information as you need for debugging. When you get weird results, don't hesitate to use simple cases, and do the calculations manually to verify your results. Perhaps instead of casting all the rays needed to create an image, just cast a single ray (and its corresponding ray tree).
- Modify the test scenes to reduce complexity for debugging: remove objects, remove light sources, change the parameters of the materials so that you can view the contributions of the different components, etc.
- Comment your code, we take this into account when grading.

Ideas for Extra Credit

- Render semi-transparent shadows where the attenuation depends on the distance traveled in the transparent object; nested refracting materials; shadow ray acceleration using early termination and without returning the normal and material; Fresnel reflection term; other BRDF models such as Cook-Torrance or Ward; anisotropic BRDFs, etc.

Updated Scene Parser

- scene_parser.h
- scene_parser.C

Input Files

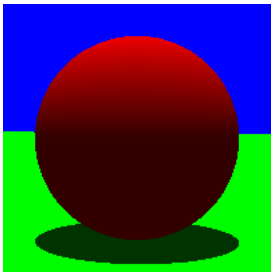
- scene4_01_sphere_shadow.txt
- scene4_02_colored_shadows.txt
- scene4_03_mirrored_floor.txt
- scene4_04_reflective_sphere.txt
- scene4_05_transparent_bar.txt
- scene4_06_transparent_bars.txt
- scene4_07_transparent_sphere_1.0.txt
- scene4_08_transparent_sphere_1.1.txt
- scene4_09_transparent_sphere_2.0.txt
- scene4_10_point_light_distance.txt
- scene4_11_point_light_circle.txt
- scene4_12_point_light_circle_d_attenuation.txt
- scene4_13_point_light_circle_d2_attenuation.txt
- scene4_14_faceted_gem.txt

Triangle Meshes (.obj format)

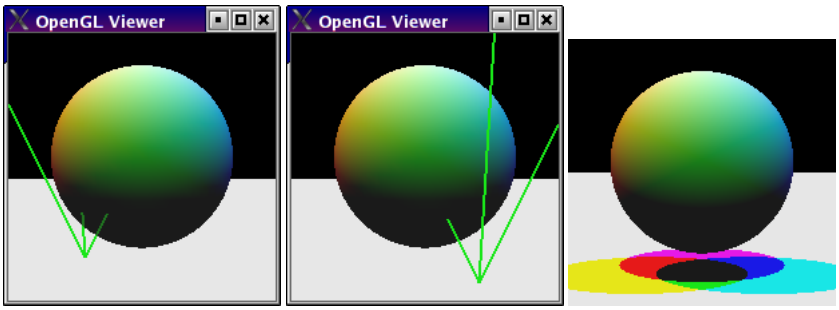
- diamond.obj

Sample Results

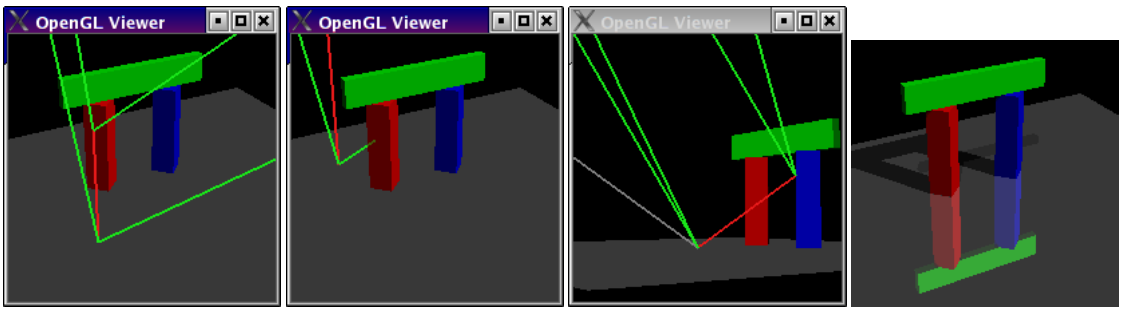
```
raytracer -input scene4_01_sphere_shadow.txt -size 200 200 -output output4_01.tga -shadows
```



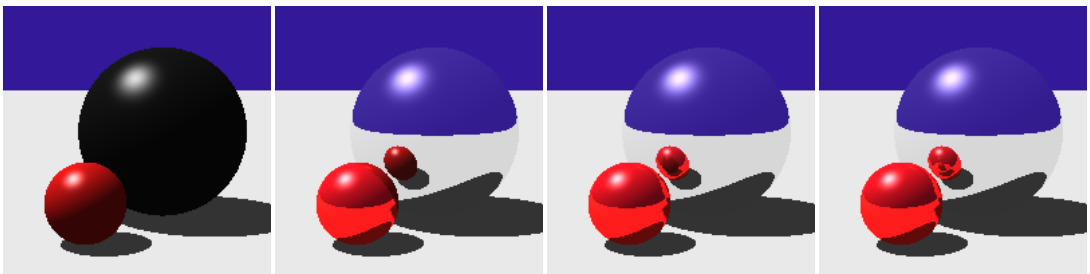
```
raytracer -input scene4_02_colored_shadows.txt -size 200 200 -output output4_02.tga -shadows -gui -tessellation 50 25 -gouraud
```



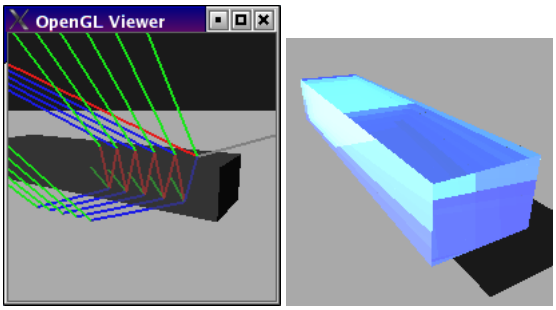

```
raytracer -input scene4_03_mirrored_floor.txt -size 200 200 -output output4_03.tga -shadows -bounces 1 -weight 0.01 -gui
```



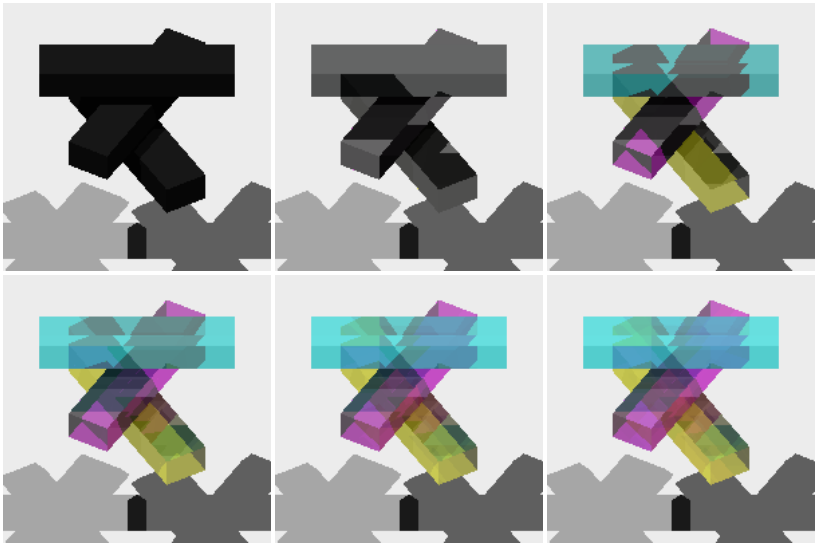

```
raytracer -input scene4_04_reflective_sphere.txt -size 200 200 -output output4_04a.tga -shadows -bounces 0 -weight 0.01
raytracer -input scene4_04_reflective_sphere.txt -size 200 200 -output output4_04b.tga -shadows -bounces 1 -weight 0.01
raytracer -input scene4_04_reflective_sphere.txt -size 200 200 -output output4_04c.tga -shadows -bounces 2 -weight 0.01
raytracer -input scene4_04_reflective_sphere.txt -size 200 200 -output output4_04d.tga -shadows -bounces 3 -weight 0.01
```



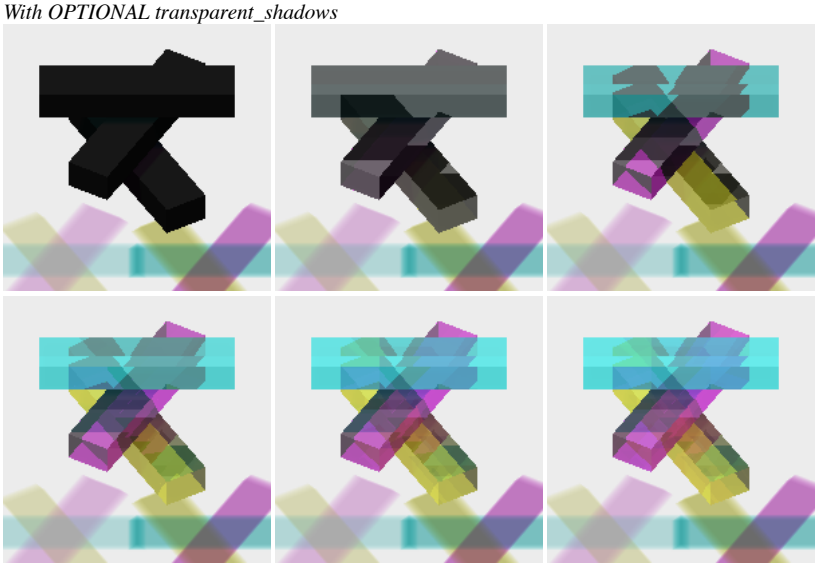
```
raytracer -input scene4_05_transparent_bar.txt -size 200 200 -output output4_05.tga -shadows -bounces 10 -weight 0.01 -shade_back -gui
```



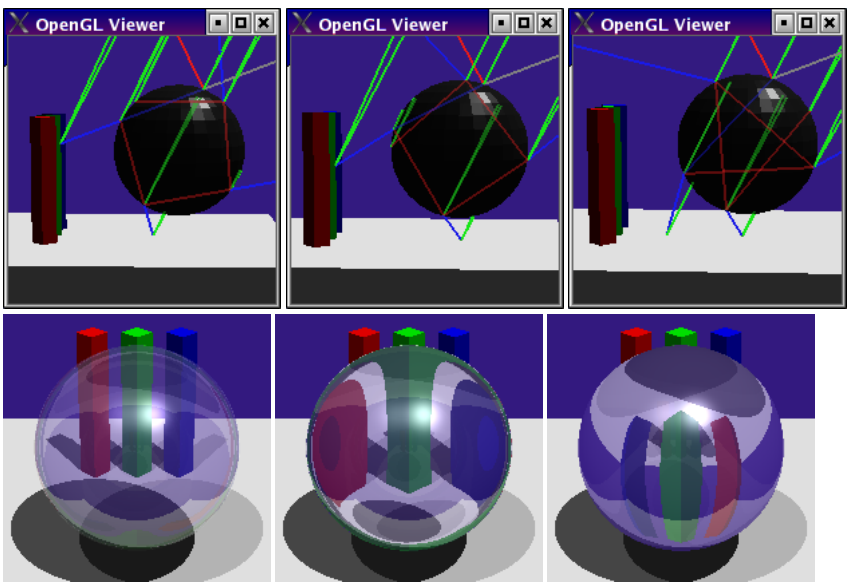

```
raytracer -input scene4_06_transparent_bars.txt -size 200 200 -output output4_06a.tga -shadows -bounces 0 -weight 0.01 -shade_back -gui
raytracer -input scene4_06_transparent_bars.txt -size 200 200 -output output4_06b.tga -shadows -bounces 1 -weight 0.01 -shade_back -gui
raytracer -input scene4_06_transparent_bars.txt -size 200 200 -output output4_06c.tga -shadows -bounces 2 -weight 0.01 -shade_back -gui
raytracer -input scene4_06_transparent_bars.txt -size 200 200 -output output4_06d.tga -shadows -bounces 3 -weight 0.01 -shade_back -gui
raytracer -input scene4_06_transparent_bars.txt -size 200 200 -output output4_06e.tga -shadows -bounces 4 -weight 0.01 -shade_back -gui
raytracer -input scene4_06_transparent_bars.txt -size 200 200 -output output4_06f.tga -shadows -bounces 5 -weight 0.01 -shade_back -gui
```



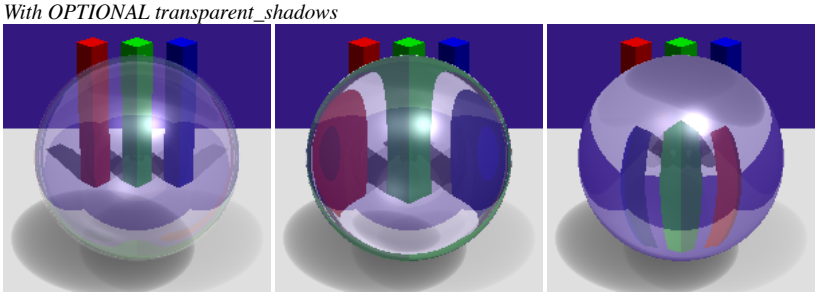
With OPTIONAL transparent_shadows



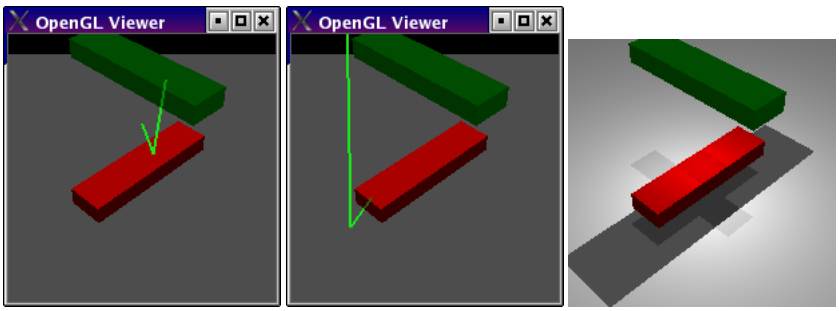
```
raytracer -input scene4_07_transparent_sphere_1.0.txt -size 200 200 -output output4_07.tga -shadows -bounces 5 -weight 0.01 -shade_back -q
raytracer -input scene4_08_transparent_sphere_1.1.txt -size 200 200 -output output4_08.tga -shadows -bounces 5 -weight 0.01 -shade_back -q
raytracer -input scene4_09_transparent_sphere_2.0.txt -size 200 200 -output output4_09.tga -shadows -bounces 5 -weight 0.01 -shade_back -q
```



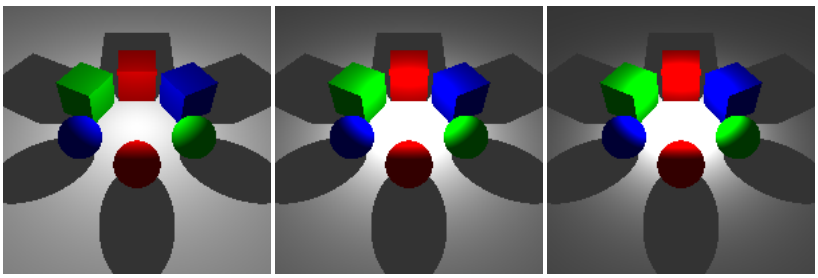
With OPTIONAL transparent_shadows



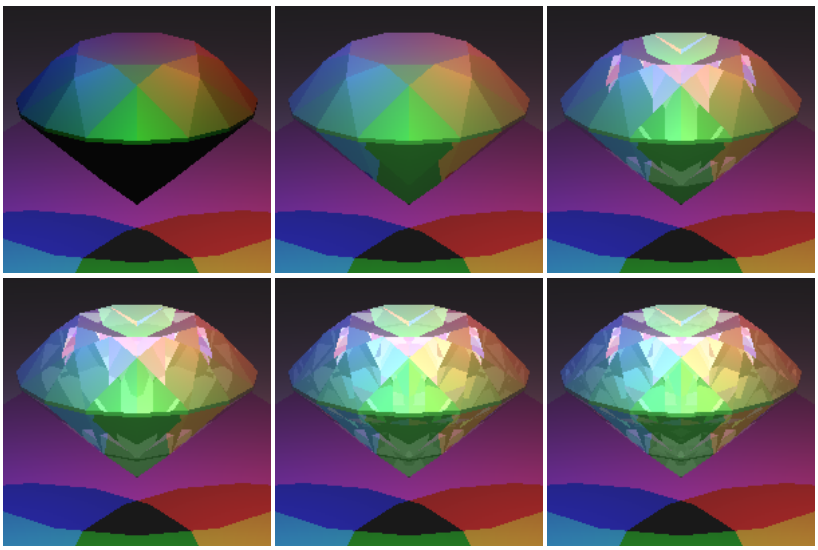
```
raytracer -input scene4_10_point_light_distance.txt -size 200 200 -output output4_10.tga -shadows -gui
```



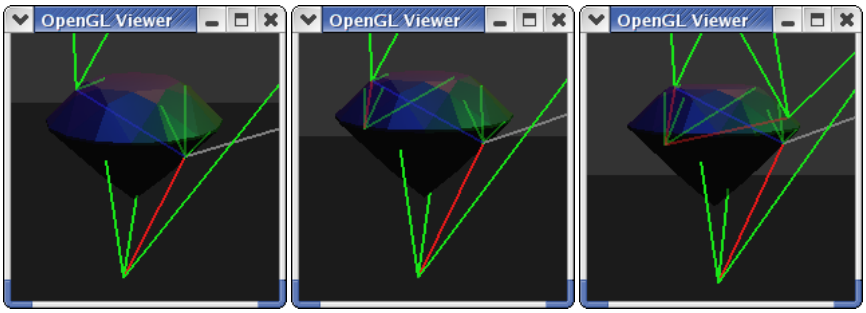

```
raytracer -input scene4_11_point_light_circle.txt -size 200 200 -output output4_11.tga -shadows
raytracer -input scene4_12_point_light_circle_d_attenuation.txt -size 200 200 -output output4_12.tga -shadows
raytracer -input scene4_13_point_light_circle_d2_attenuation.txt -size 200 200 -output output4_13.tga -shadows
```



```
raytracer -input scene4_14_faceted_gem.txt -size 200 200 -output output4_14a.tga -shadows -shade_back -bounces 0 -weight 0.01
raytracer -input scene4_14_faceted_gem.txt -size 200 200 -output output4_14b.tga -shadows -shade_back -bounces 1 -weight 0.01
raytracer -input scene4_14_faceted_gem.txt -size 200 200 -output output4_14c.tga -shadows -shade_back -bounces 2 -weight 0.01
raytracer -input scene4_14_faceted_gem.txt -size 200 200 -output output4_14d.tga -shadows -shade_back -bounces 3 -weight 0.01
raytracer -input scene4_14_faceted_gem.txt -size 200 200 -output output4_14e.tga -shadows -shade_back -bounces 4 -weight 0.01
raytracer -input scene4_14_faceted_gem.txt -size 200 200 -output output4_14f.tga -shadows -shade_back -bounces 5 -weight 0.01
```



```
raytracer -input scene4_14_faceted_gem.txt -size 200 200 -shadows -shade_back -bounces 1 -weight 0.01 -gui
raytracer -input scene4_14_faceted_gem.txt -size 200 200 -shadows -shade_back -bounces 2 -weight 0.01 -gui
raytracer -input scene4_14_faceted_gem.txt -size 200 200 -shadows -shade_back -bounces 3 -weight 0.01 -gui
```



See the main Assignments Page for submission information.
