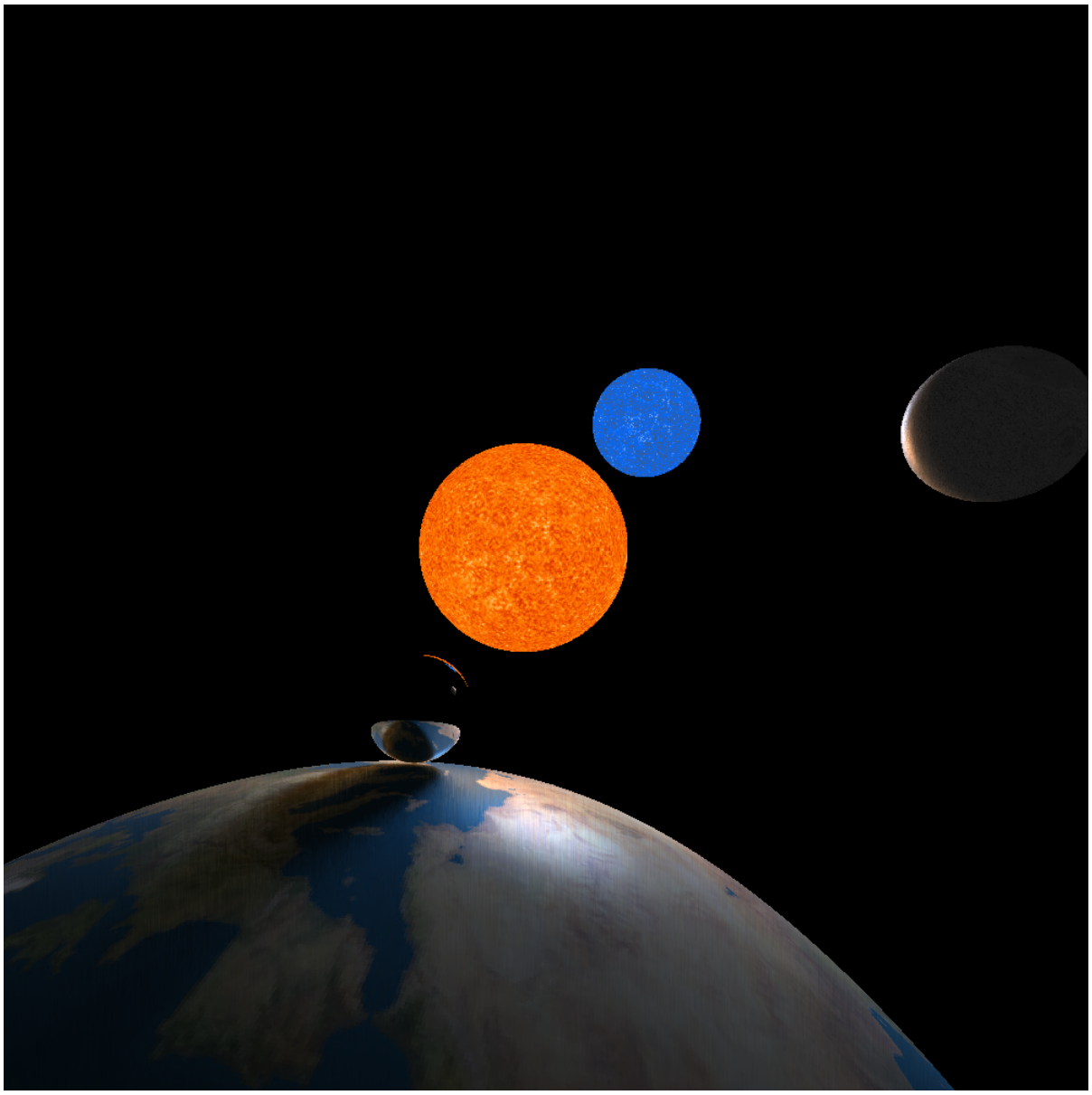
EasyRay User Manual



Alex Aulabaugh • 2017

Version 1.0

Table of Contents

0
Introduction

EasyRay provides an interface for generating images of scenes through a technique called ray tracing. Ray tracing reverses the process of a normal camera taking a photo in that it follows the paths of light out of the camera into the scene where it bounces off of objects and reaches a light source. This technique, while computationally costly, is capable of easily producing realistic effects in images.

EasyRay is written entirely in Java so that Java's Garbage Collection process can be used to effectively manage large amounts of memory allocation associated with the high performance computing application of ray tracing. The first draft of this program was written in C++, but memory leaks became quickly out of control, limiting the performance demands which could be placed on the system. That draft was written for the Spring 2016 iteration of *CS419: Production Computer Graphics* at the University of Illinois at Urbana Champaign. This program was written one year later for the Spring 2017 iteration of *CS242: Programming Studio* as a final project.

Though EasyRay requires no coding to use, the full source code is provided with this release so that extensions and changes can be made by users. The general code structure is provided in this manual for that reason. The focus of *CS242: Programming Studio* was code design principles, so it is the author's hope that the structure is logical and easy to modify. The modification of EasyRay's GUI is more difficult partially due to the nature of the Java's Swing GUI toolkit, and a good understanding of Swing is recommended. Please see http://docs.oracle.com/javase/tutorial/uiswing/.

EasyRay can be run out of the box by most operating systems simply by executing the runnable jar *EasyRay.jar*. If the project is imported into an IDE, the main file for the project is located in *src/gui/RenderingInterface.java.*

1
General Code Structure

The following sections detail the workflow of EasyRay's ray tracing implementation. This is recommended reading for those who wish to touch the source code to add their own extensions or modifications. It is not needed to understand the usage of the GUI to control ray tracing - for that information please see part 2. This section will assume the reader is familiar with Java, Object Oriented Programming principles, and vector math.

1.1 Scene, Camera, Settings

The GUI of EasyRay primarily provides an interface to easily edit configuration files for the *scene*, the *camera*, and the *settings*. The *scene* details the world and all in it. This can include geometric objects like spheres, as well as point light sources. Ray tracing *cameras* have specifications like position and direction, and can generally be thought of as physically in the scene. Cameras also specify the dimensions of the film and image capture specifications discussed later. The *settings* include output file specifications like filename and image dimensions (the film can be stretched and scaled), as well as global constants like maximum recursion depth. It also specifies which *camera* and *scene* configurations to use.

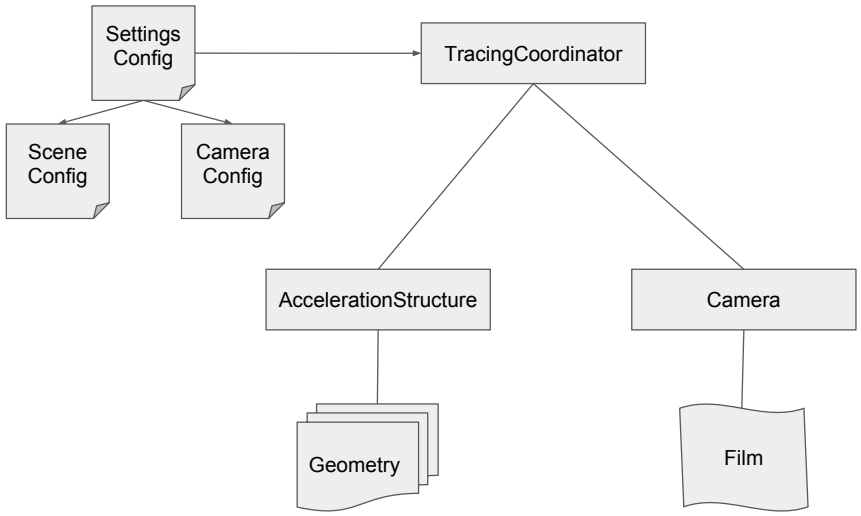


Fig1: EasyRay's simplified rendering structure

EasyRay represents these things as depicted in Fig1 above. The Settings Config file (*src/config/settings*) includes references to a Scene Config file (*src/config/scene*) and a Camera Config file (*src/config/camera*). These are all plain text files formatted by new line and space delimiters. Their structure is interpreted by a TracingCoordinator class and by the geometry classes. EasyRay's GUI panels are also must be able to interpret these configuration files. Unfortunately this means one type of geometry must specify its configuration format twice, once in its own file and once in its GUI panel - this is a design oversight should be patched in future iterations.

As shown in Fig1, the Settings Config file is supplied to a class called *TracingCoordinator* (*src/rendering/TracingCoordinator.java*). This coordinator wraps the entire ray tracing process, containing the camera and the geometry.

In EasyRay, geometry is stored in an *AccelerationStructure* (*src/acceleration*). An acceleration structure is a ray tracing concept to speed computation - more on this later. The *TracingCoordinator* creates the *AccelerationStructure* to the specifications of the configuration files, and stores it.

The *TracingCoordinator* also creates and stores a *camera* (*src/rendering/camera.java*). EasyRay's *camera* models a pinhole camera (see https://en.wikipedia.org/wiki/Pinhole_camera). It has both an eye point and a film - light from the scene converges on the eye point, passing through the film. Film in EasyRay's camera is an array of conceptual pixels which have physical locations and dimensions in the scene.

1.2 Geometry Hierarchy

In EasyRay, the main task is to render geometry, which can be arranged and lit in interesting ways. To enable lighting, all geometry must be able to determine if and where it intersects a ray. This requirement is fulfilled in different ways by different types of geometry, so a geometry abstract class is required.

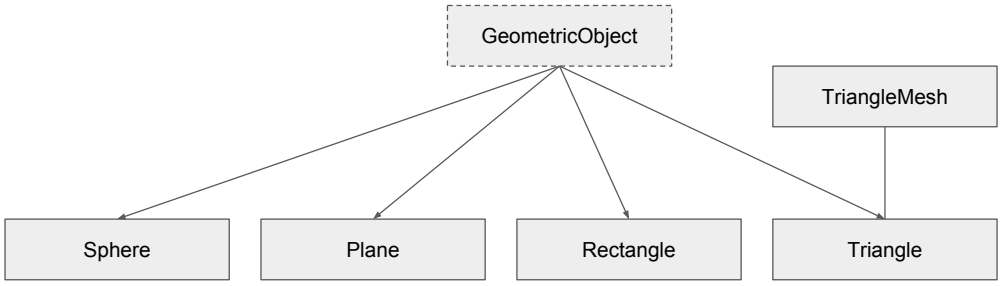


Fig2: Geometry Hierarchy

As depicted in Fig2, all basic geometry is descendant from the *GeometricObject* (*src/geometry/GeometricObject.java*) abstract class. This *GeometricObject* defines tasks all geometry must implement:

- Return a color given a point on the object
- Return a normal vector given a point on the object
- Return all intersection points a given ray has with the object
- Return one or more reflection rays given an incident ray and an intersect point
- Return one or more refraction rays given an incident ray and an intersect point

In addition to these tasks, there are features all geometry have in common which must be accessed on demand. These are namely:

- A default color
- A specular constant (See https://en.wikipedia.org/wiki/Phong_reflection_model)
- Reflectivity and refractivity constants
- Whether the object emits light
- A bounding box (For acceleration structures)

These make up the basic properties of the geometry. Each subclass of *GeometricObject* adds properties to describe the location and shape of the object - some of them include things like texture files.

TriangleMesh (*src/geometry/TriangleMesh.java*) is a bit of an exception among EasyRay's geometry. It contains many of the properties of a *GeometricObject*, but it is not a *GeometricObject*. Rather, it maintains an ArrayList (java.util) of *Triangle* objects which are read in from an obj file. It simply passes down any properties it has to all of the triangles of the mesh. The *TriangleMesh* contains several affine transformation functions not available to *GeometricObjects* including:

- Stretch
- Scale
- Rotate

These are supplied so that the mesh can be manipulated and placed anywhere in the scene. Obj files accepted by *TriangleMesh* should have all points centered around the origin so that the transformations work properly. Additionally, many triangle meshes require scaling up to be visible. This is the case for the provided Stanford Bunny example obj file (*src/meshes/bunny.obj*).

1.3 Rendering Internals

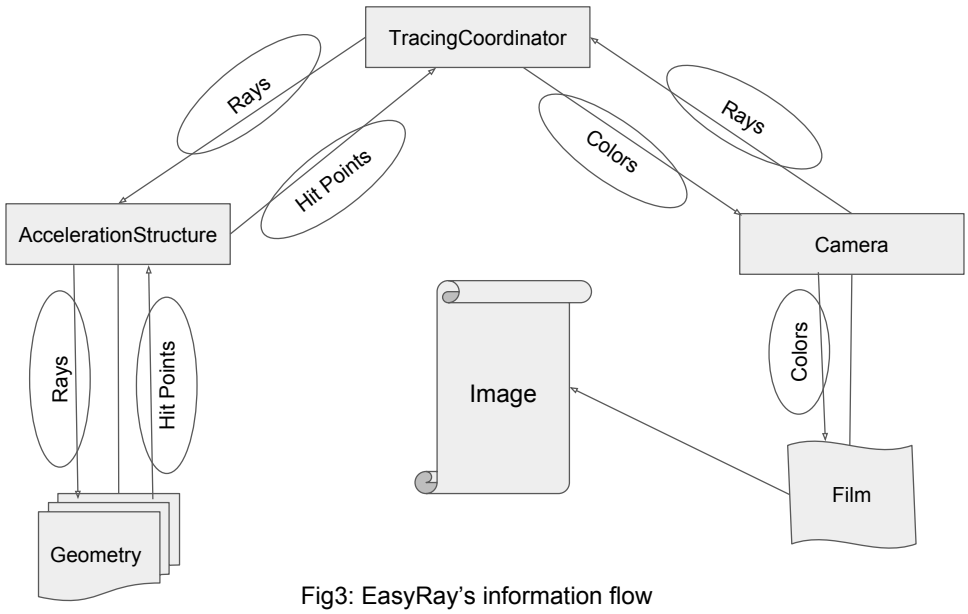


Fig3: EasyRay's information flow

Fig3 outlines the general flow of information between classes during rendering. Before the GUI was implemented, initiation of each step of the process was performed in *TracingCoordinator*'s constructor. Now it is the GUI panels which call *TracingCoordinator*'s public methods to perform the ray trace. There are two:

- loadFromFile
- renderToFile

loadFromFile reads the serialized information about the scene, camera, and settings from disk and loads it into its internal data structures. When reading specifications for geometry, it passes the encoded specifications to the appropriate *GeometricObject*'s constructor, thus abstracting knowledge of their on-disk representation away. However, since there is only one type of camera, *TracingCoordinator* parses the *Camera* specifications directly and calls *Camera*'s constructor with the parameters as arguments. A natural step before extending the *Camera* would be to move this parsing into the *Camera* class to resemble the *GeometricObject* classes.

Additional information on rendering internals is provided by the block comments within the source code.

2
Graphical User Interface

This section gives detail on the GUI for EasyRay, and is intended for all users. It includes detailed descriptions of all features, as well as examples of usage. For implementation details, see part 1.

2.1 Rendering Panel

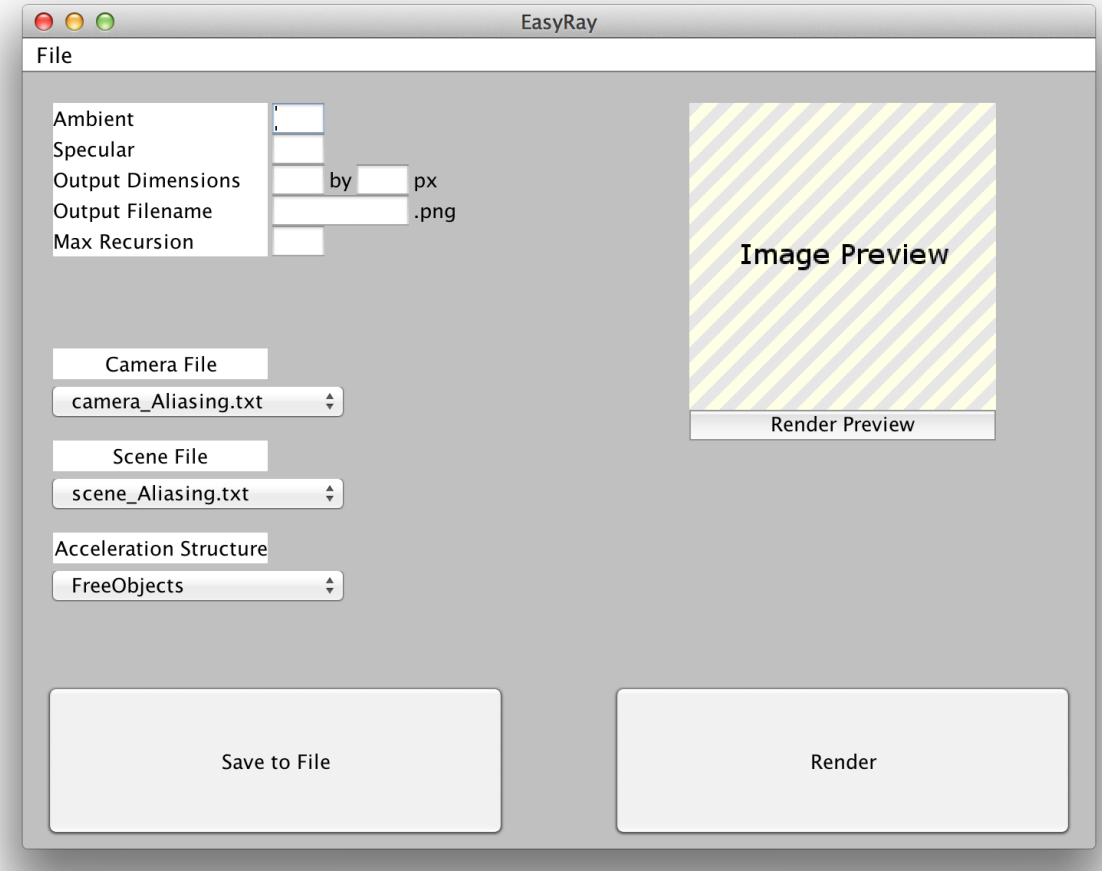


Fig4: The Default Rendering Panel

The default rendering panel - which is the first screen shown when launching EasyRay - is shown in Fig4. This panel defines the top level settings for rendering such as global constants, which camera to use, which scene to use, and any acceleration structures. It also allows a preview of the image which will be rendered with the current settings to be generated.

Let's look at the components on this window:

- The first input box, labeled "Ambient," describes the amount of background lighting, i.e. how bright objects are before illumination. A typical value is 0.13. A higher value means a brighter scene.
- The input box labeled "Specular" controls the weight assigned to the specular component of the lighting model. It is recommended this is kept at 1.0. For more information about specular components, see the Wikipedia page "Specular reflection."
- The "Output Dimensions" input boxes determine the dimensions of the output image file.
- The "Output Filename" input box determines what name the output image file is given.
- The "Max Recursion" input box determines how many times light rays can be reflected or refracted before they are no longer tracked. Higher numbers yield better images with multiple mirrors or transparent objects, but are computationally more expensive.
- The following drop down boxes determine which camera, scene, and acceleration structures to use. An acceleration structure is simply a way to store objects in the scene to speed computation. Images with many objects should use an acceleration structure, otherwise FreeObject (no structure) works perfectly well.
- The "Save to File" button at the bottom will prompt you to name and save this rendering configuration as a file, which can be loaded later.
- The render button performs the ray tracing and displays the output in a window, as well as saving it on disk, in the same directory as EasyRay.jar.
- The preview box on the top right will generate a small preview of the image with the current settings. However, certain values are restricted to ensure fast computation, so this image will not exactly resemble the final output.
- The File menu at the top allows the creation or loading of existing rendering configurations, scene configurations, or camera configurations. "Reload Files" manually updates this EasyRay instance with any changes to the filesystem.

Let's go through a brief example.

- Fill out the info boxes to resemble Fig5 below, and push Render Preview.

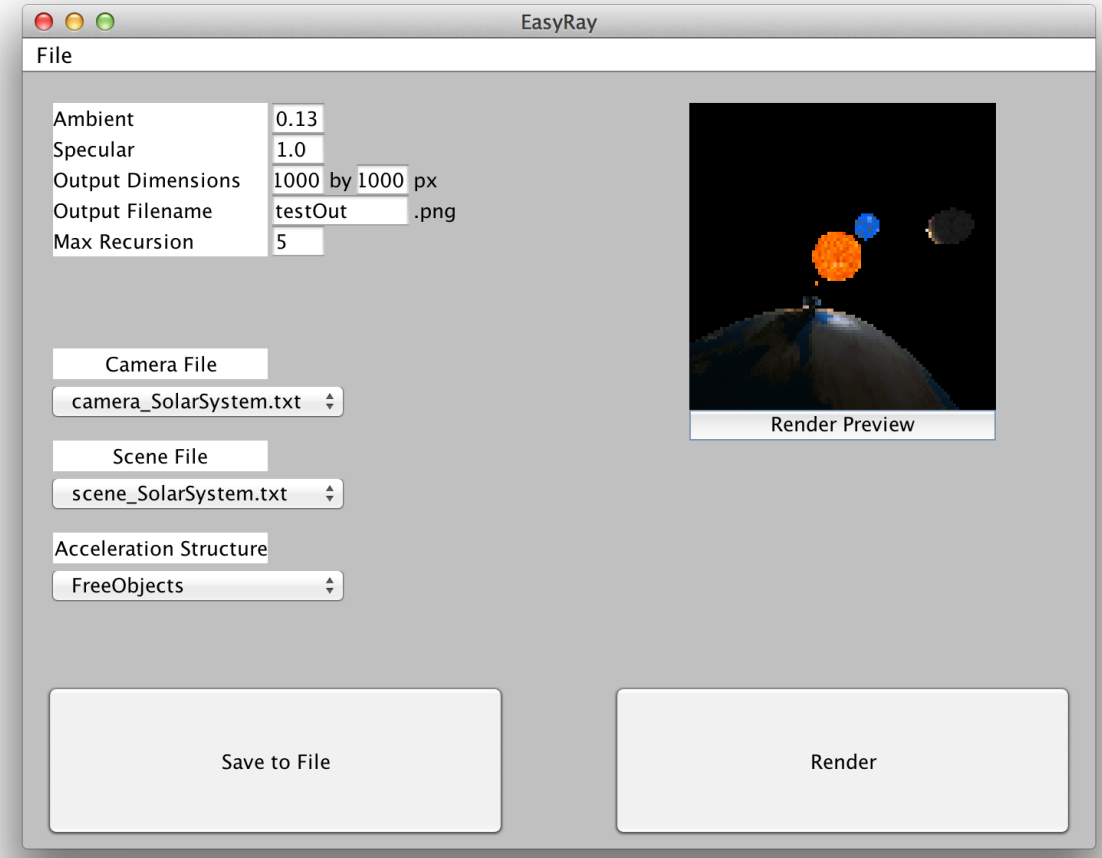


Fig5

- The preview should be generated nearly instantly. You may play with the ambient and specular components to see their effect on the preview - remember to push “Render Preview” to update the image with any changes.
- To generate the final image, push “Render.” This should take less than a minute, and open a window with the final rendered image. Additionally, you should see testOut.png in EasyRay’s folder.
- To load a preset, open File > Load > Rendering and select settings_Aliasing.txt. This should edit the values on the rendering panel.

2.2 Camera Panel

The Camera Panel controls the camera’s position and other attributes. To open the Camera Panel, select File > Load > Camera and select camera_Aliasing.txt. This will open the camera settings for the aliasing example as loaded at the end of the rendering panel example. You should see the following window open (Fig6).

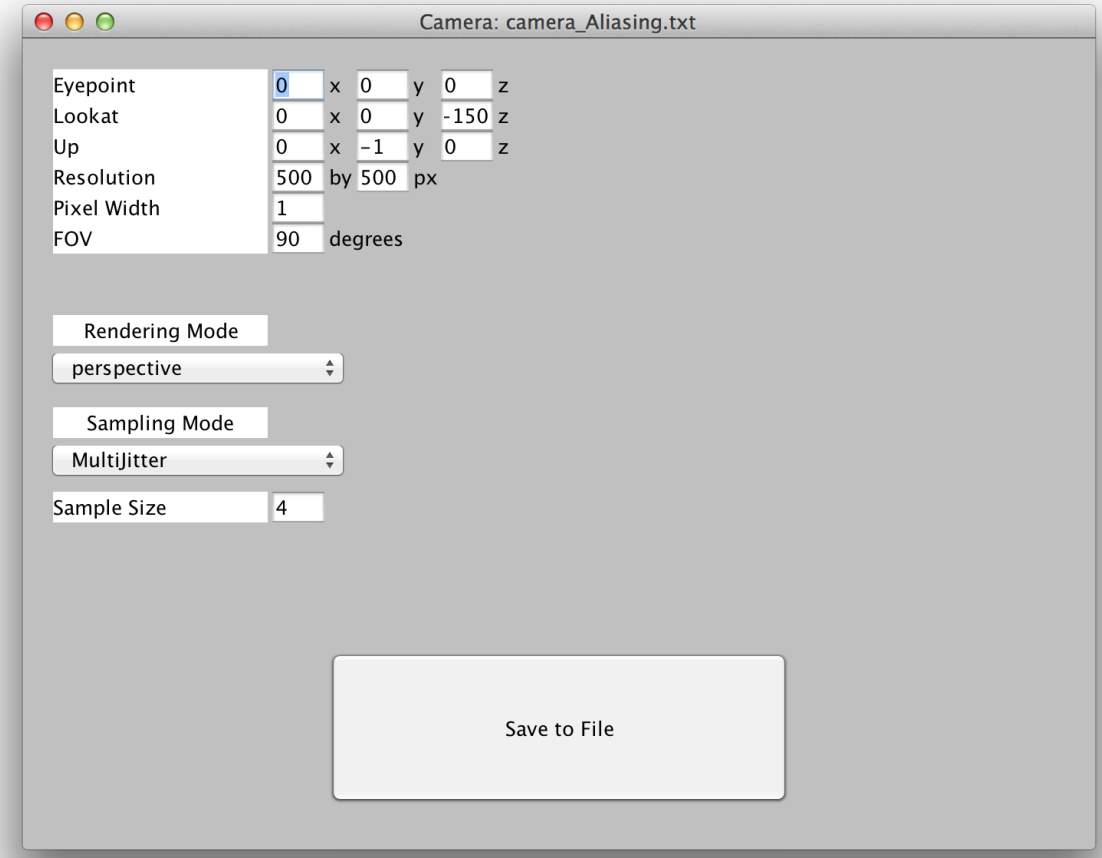


Fig6

The rendering panel. should still be open.

Let's look at all the components on this panel:

- The "Eyepoint" info boxes determine the location of the camera in the scene. Z is defined as into the screen, x is left-right, and y is up-down.
- "Lookat" defines a point that the camera is pointed at.
- "Up" defines the a point that the top of the camera is facing.
- "Resolution" specifies the number of pixels on the camera's film.
- "Pixel Width" represents the size of the pixels. Larger pixels means rays will be more spread out, and the image will be wider.
- "FOV" means Field of View. Please see the related Wikipedia article for an explanation.
- "Rendering Mode" can be "perspective" or "orthographic." Please see the related wikipedia articles for "Perspective (graphical)" and "Orthographic projection."
- Sampling mode determines how many samples are taken for each pixel, which essentially translates to anti-aliasing. There are multiple techniques to choose from,

as well as a sample size. More samples means more anti-aliasing, but mean more computational cost.

Let's do an example.

- Please have "settings_Aliasing.txt" for the rendering panel, and the "camera_Aliasing.txt" camera panel open.
- Render a preview of the scene.

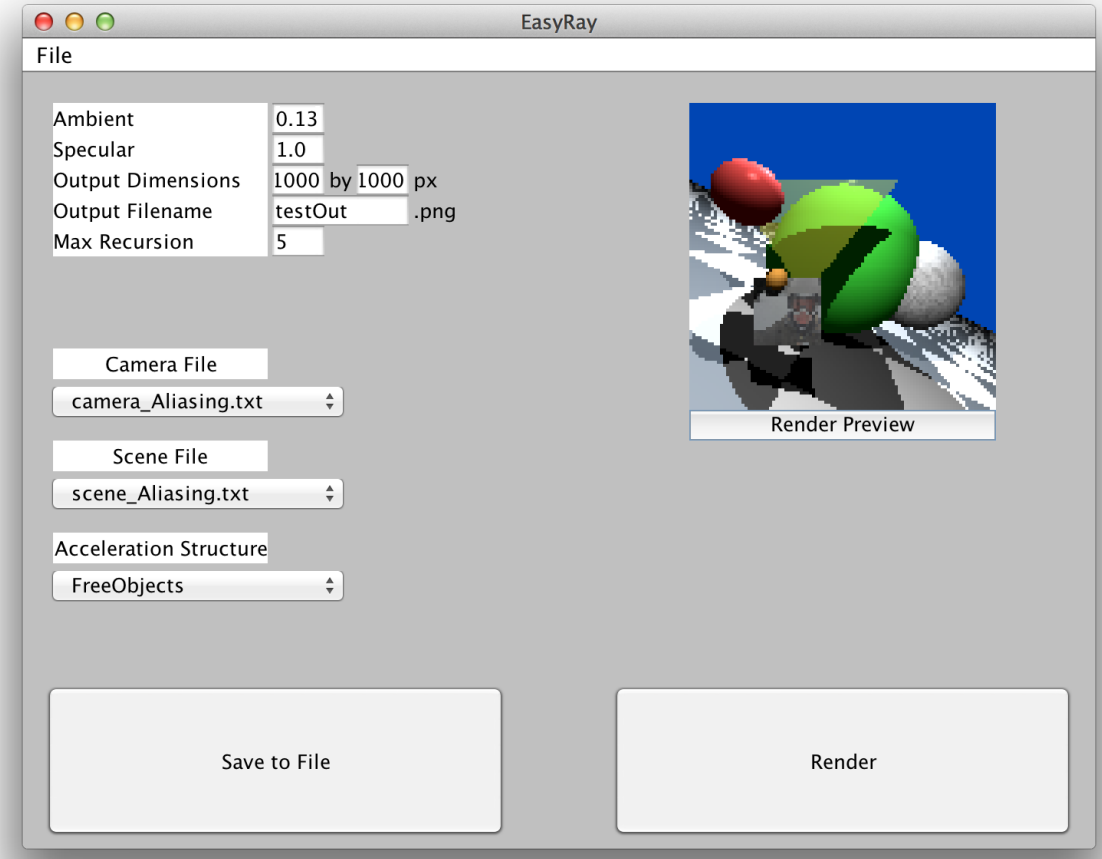


Fig7

- Tweak the camera "Eyepoint" to 50, 0, 0.
- Click, "Save to File" and keep the default name.
- Re-render the preview in the rendering panel. The camera has moved. Continue to tweak this to test the features.

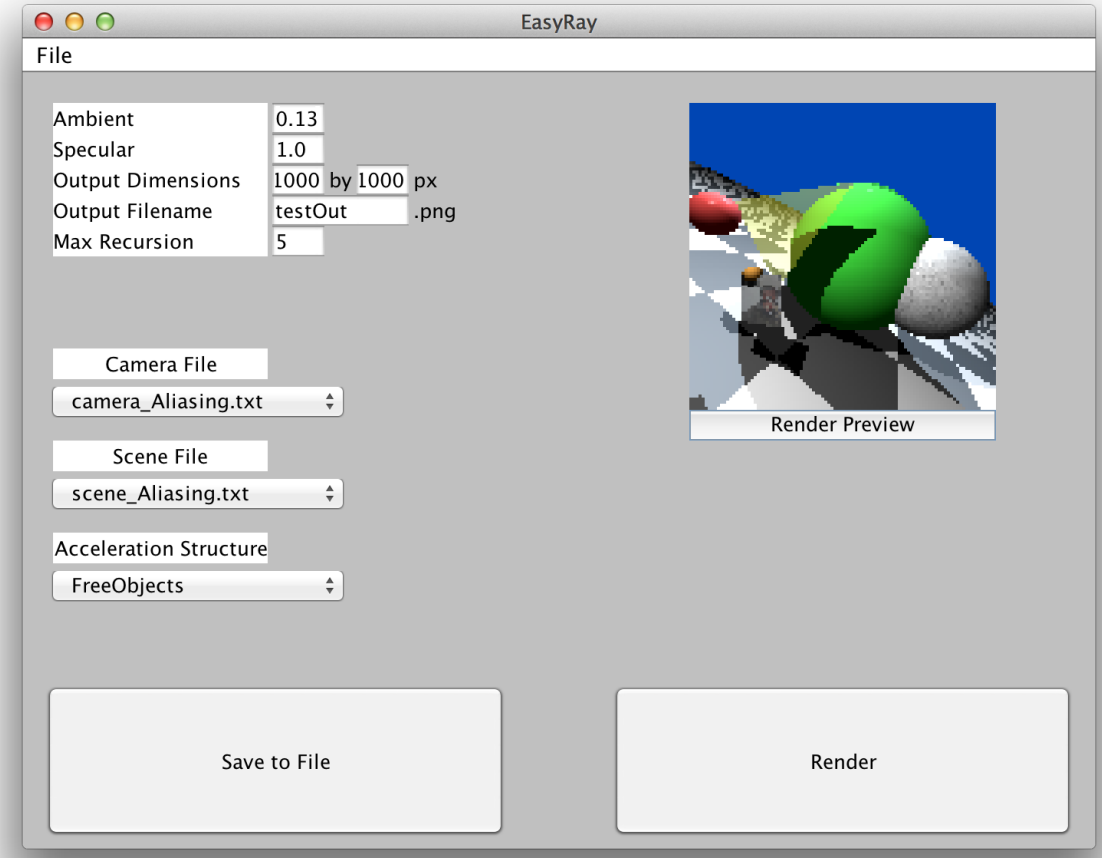


Fig8

2.3 Scene Panel

The Scene panel describes the geometry and lights in the rendering scene. To open the scene panel, go to File > Load > Scene and select “scene_Aliasing.txt”. The following panel should display:

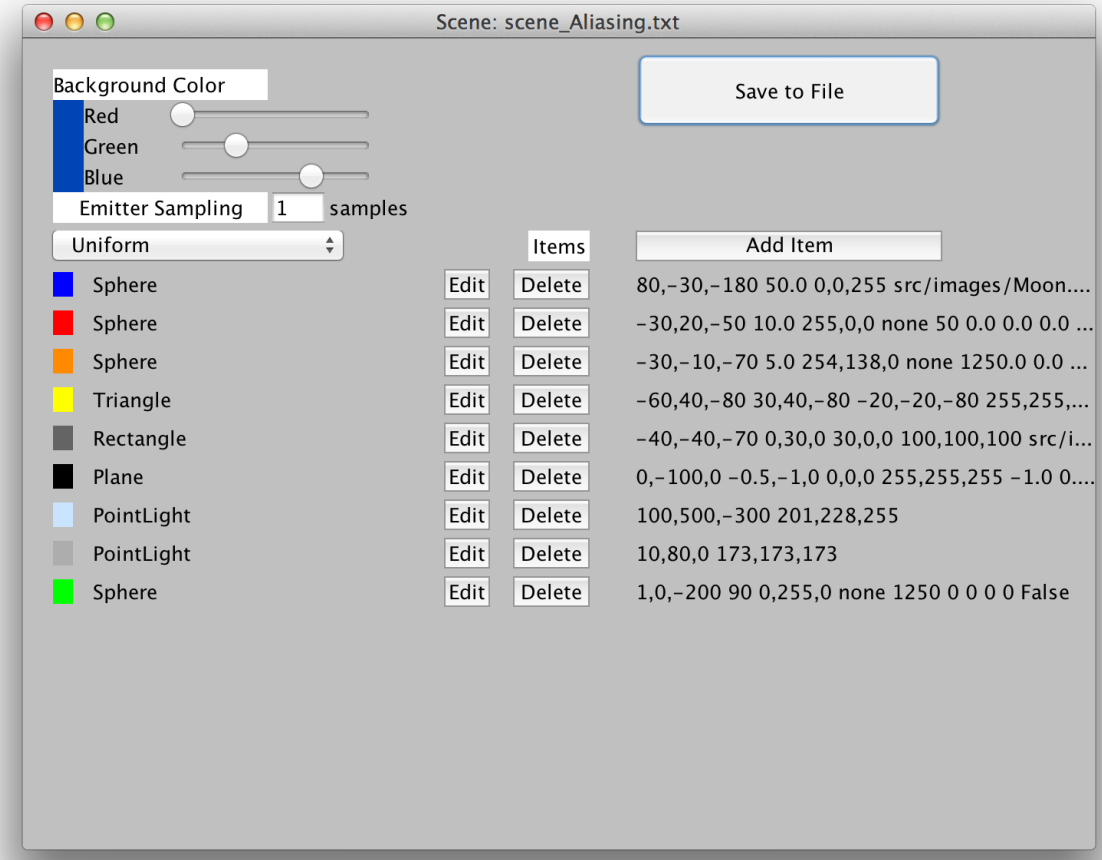


Fig9

This describes everything in the “Aliasing” scene. Let’s go through the elements:

- There are sliders for the RGB values of the scene’s background color. Sliding them should change the preview to their immediate left.
- The “Emitter Sampling” info box describes the number of rays a light emitting object in this scene will trace to determine lighting. More means a better image, but is computationally more expensive. The drop down box below this determines the sampling technique for emitters.
- The “Save to File” button on the upper right will save these settings to a scene file.
- Below this, there is a list of all items in the scene. Each item has several components:
  - An overall color for the object is displayed on the far left.
  - The “Edit” button launches a geometry panel so that the object can be edited.
  - The “Delete” button removes the object.
  - The text on the far right is the serialized representation of this object’s settings. This is to help differentiate objects to users who are familiar with the serialization, and can be safely ignored by most users.
  - The “Add Item” button will allow the user to place more items in the scene.

Let's go through an example:

- Ensure that "settings_Aliasing.txt" is loaded in the rendering panel, and "scene_Aliasing.txt" is loaded in the scene panel.
- Edit the sliders in the scene panel to a new value. Press "Save to File" and keep the default filename.
- Press "Render Preview" in the rendering panel, the background should be altered.

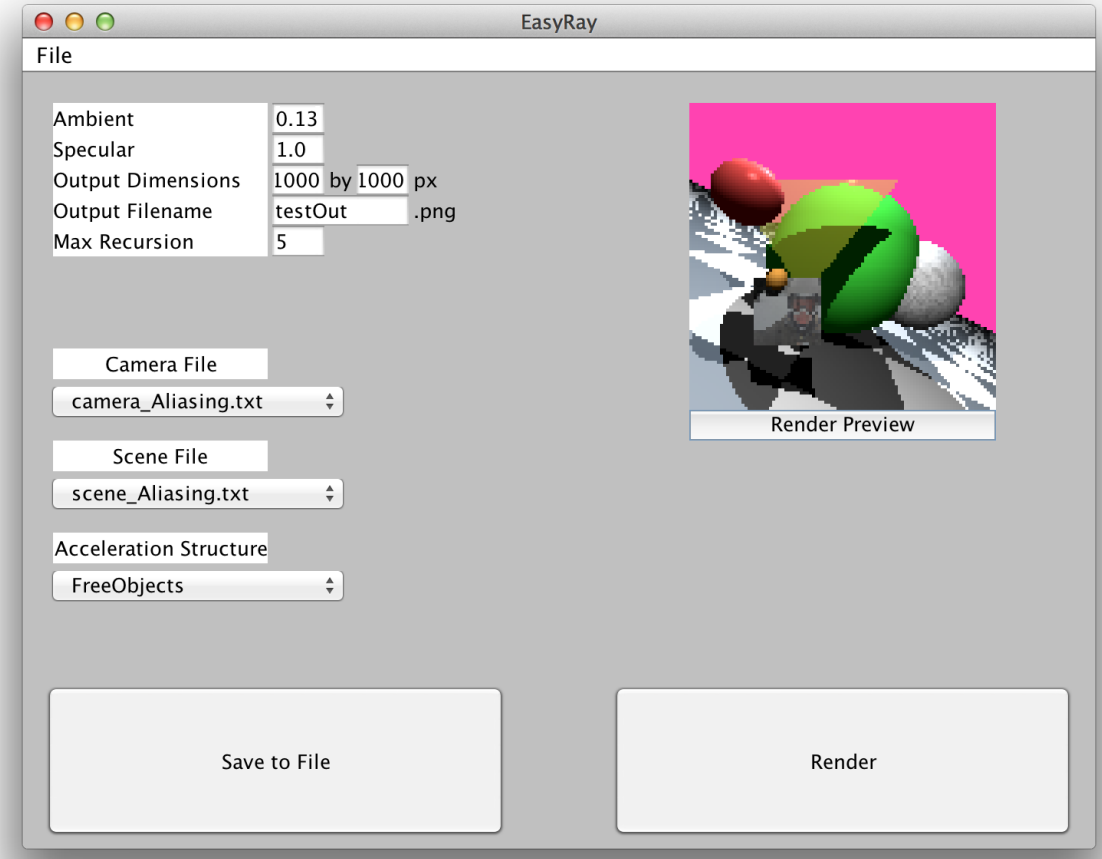


Fig10

- Edit the other options to see different features. We will next go through adding and editing items into the scene.

2.4 Geometry Panels

Geometry Panels, and other item panels, define items in the scene. As such, they are launched from the scene panel when adding or editing an item. There are several types of these panels, and we will go through them all.

2.4.1 Sphere

A sphere panel, intuitively, defines a sphere. Its panel looks like Fig11 below.

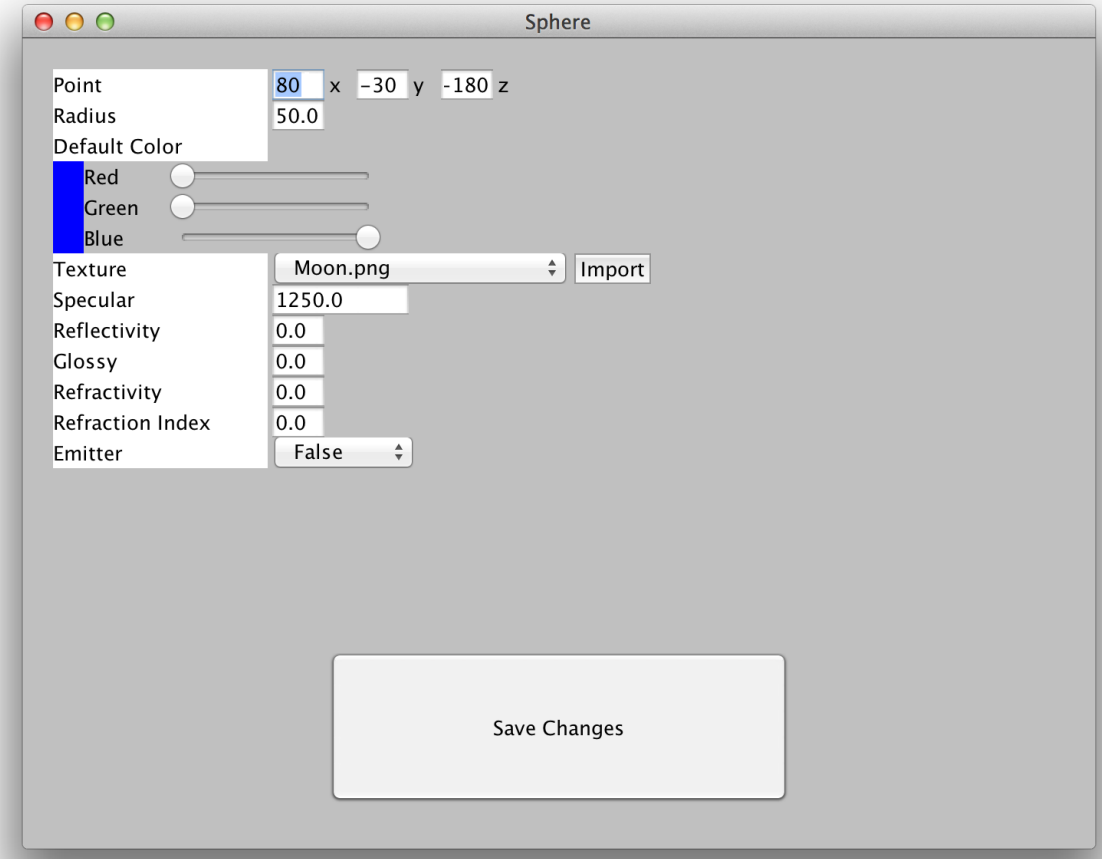


Fig11

Lets go through the elements:

- The “Point” and “Radius” info boxes are self-explanatory.
- The “Default Color” RGB sliders define the color of the object if there is no specified texture. The colored bar to their left is a preview of the color.
- The “Texture” drop down box allows the selection of an image texture, or for one to be imported from anywhere on a user’s filesystem. Note that importing from somewhere on a user’s filesystem will copy that image into EasyRay’s working files.

- The “Specular” info box defines the specular exponent of this object. This determines how smooth or rough the object appears with the lighting. Higher numbers are more mirror-like. For more information, see the Wikipedia page on “Specular reflection.”
- The “Reflectivity” info box defines the degree to which incoming light is reflected by the object. 1.0 is complete reflection.
- The “Glossy” info box determines how glossy reflection will be. This is achieved by sampling. A lower number means a glossier reflection.
- The “Refractivity” info box determines the degree to which incoming light that is refracted.
- The “Refraction Index” info box represents the physics concept of an index of refraction. The air is defined to have refraction index of 1 in EasyRay.
- The “Emitter” drop down box determines if the sphere is an emitter or not.

Let’s do an example:

- Ensure that “settings_Aliasing.txt” is loaded in the rendering panel, and “scene_Aliasing.txt” is loaded in the scene panel.
- Render the scene.

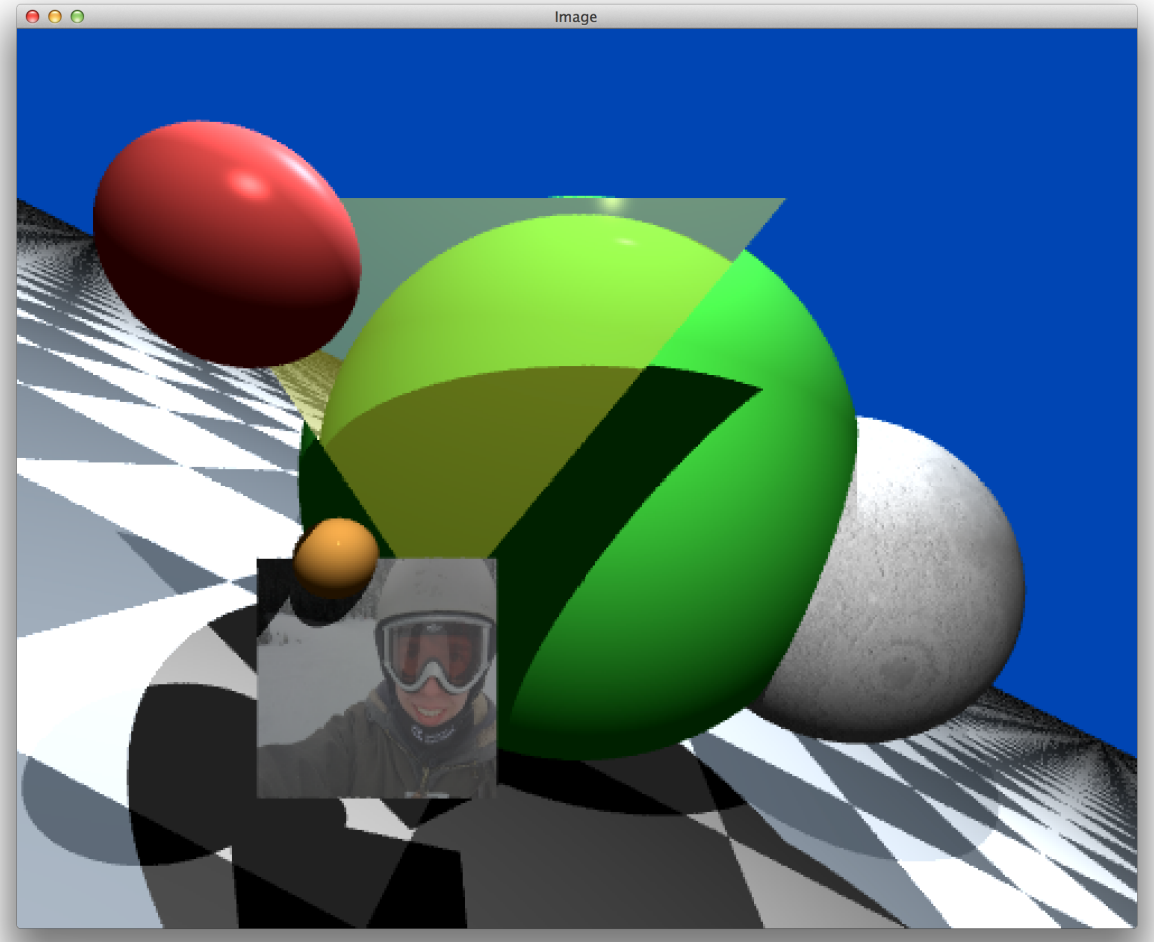


Fig12

- Click “Edit” for the second sphere. It should be red.
- Edit the radius to 5.0 and save changes.
- Save the scene to the default file.
- Render the scene, the red sphere should have shrunk.

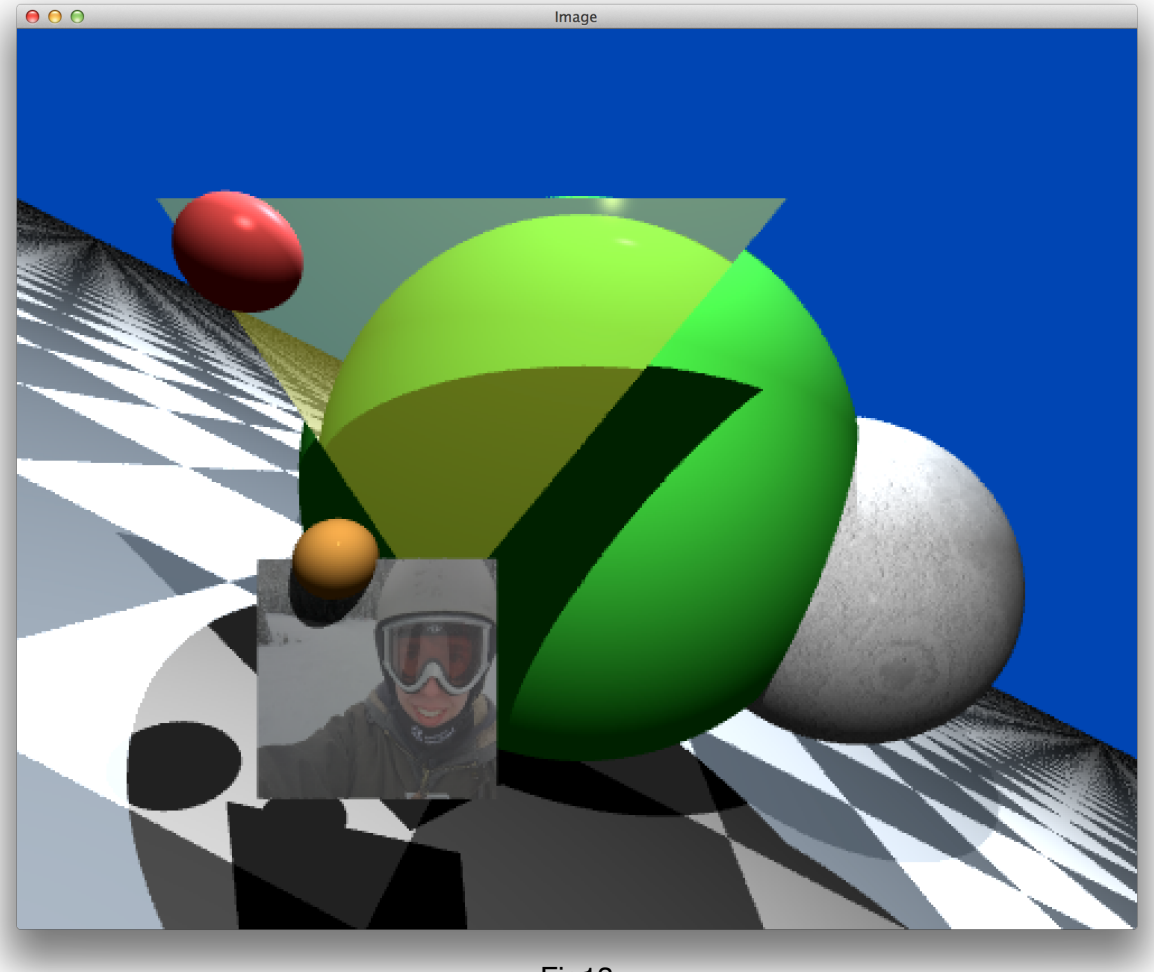


Fig13

- Continue to edit settings to understand the functionality.

2.4.2 Triangle

The Triangle panel defines a single triangle. This is different from a Triangle Mesh, defined later. Its panel looks like Fig14 below.

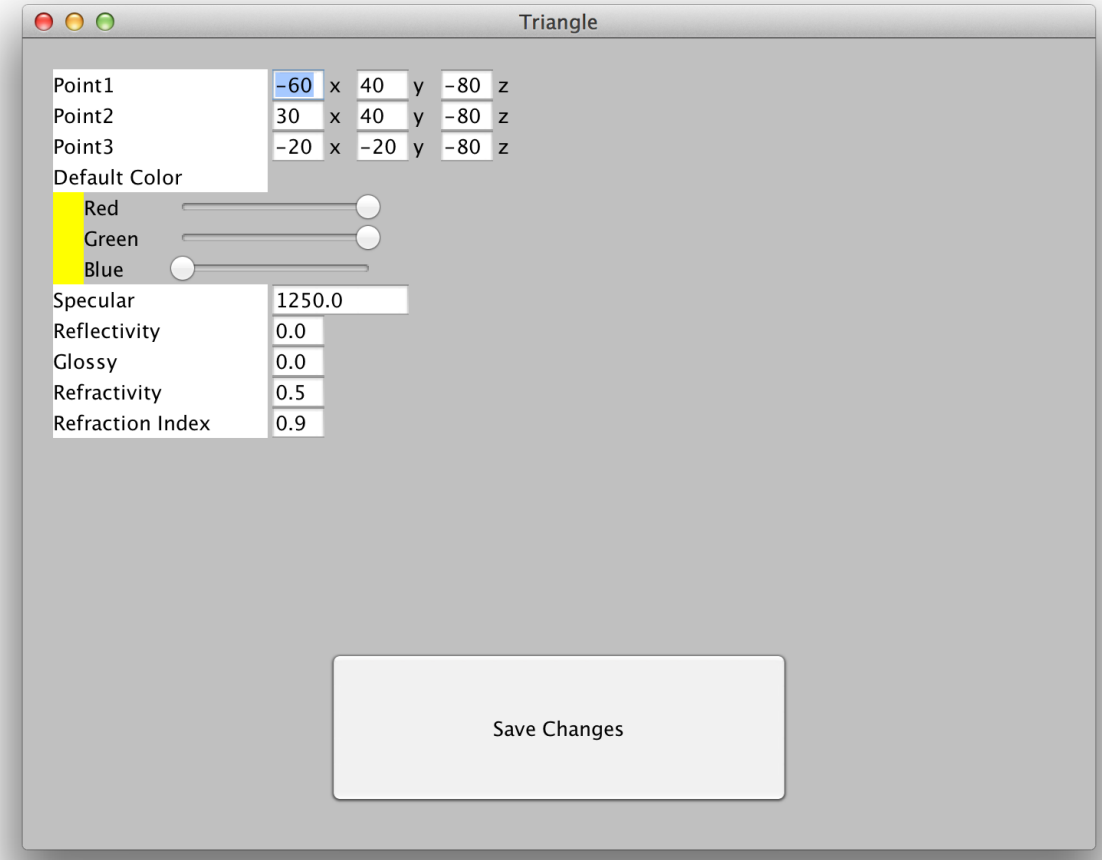


Fig14

Most of the elements are exactly the same as the sphere's panel, and will not be repeated here. See section 2.4.1 for these. The difference is that the "Point" and "Radius" boxes have been replaced with Point1, Point2, Point3, which are the vertices of the triangle. Additionally, there is no option for a texture. As these points are intuitive, no example will be provided.

2.4.3 Rectangle

Intuitively, the Rectangle panel defines a rectangle, or a bounded plane. The panel looks like Fig15 below.

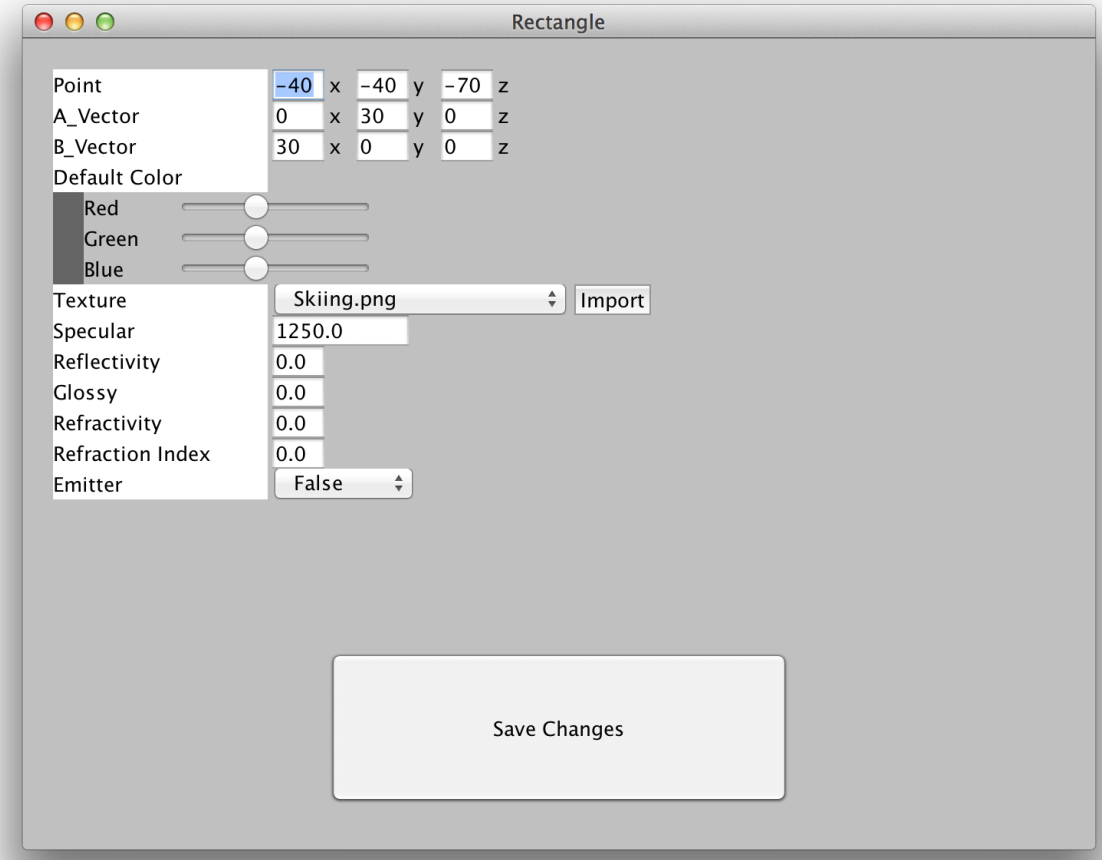


Fig15

Many of the elements here have been discussed in section 2.4.1, and will not be repeated. What is different is that the rectangle is defined by a point and two vectors. The point represents the bottom left corner of the rectangle. The A_Vector represents a vector from the bottom left of the rectangle to the top left of the rectangle, and the B_Vector represents a vector from the bottom left of the rectangle to the bottom right of the rectangle.

2.4.4 Plane

A plane is a special type of geometry in that it is not held by an acceleration structure. This is because it goes on to infinity, and would not work in a finite acceleration structure. The plane's panel looks like Fig16 below.

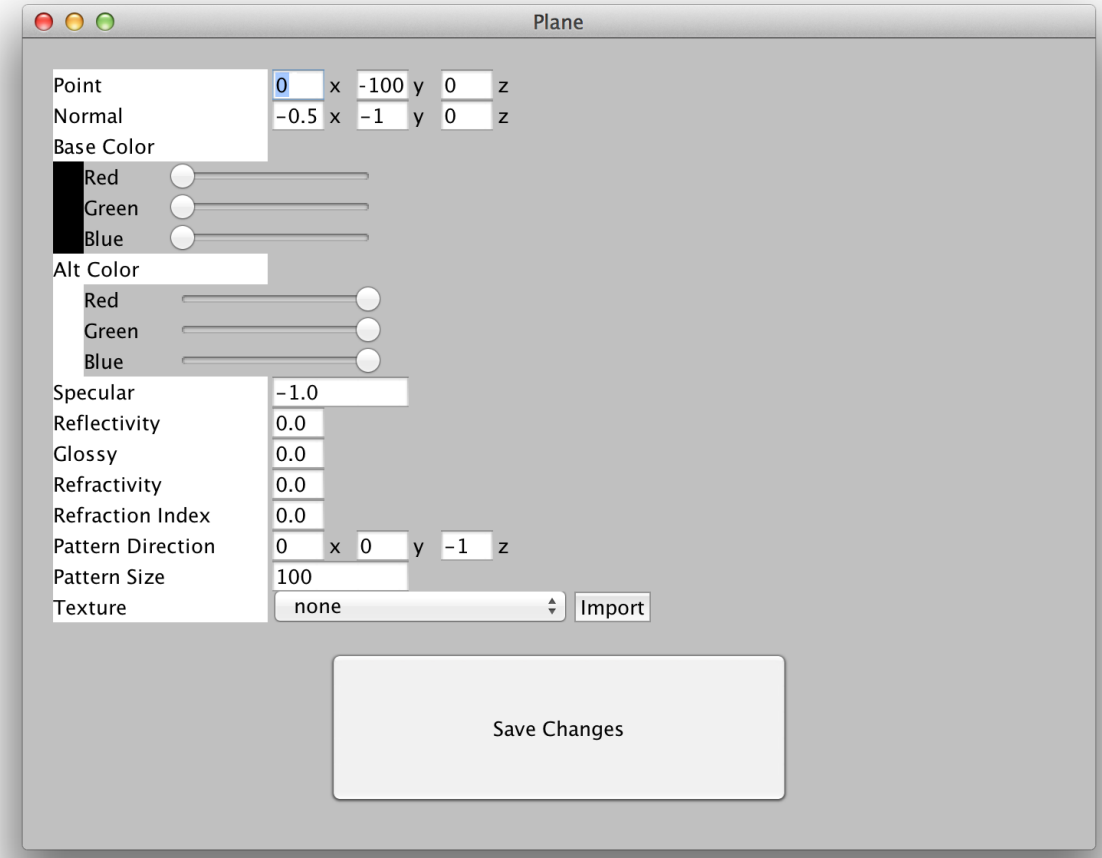


Fig16

Many of the elements of this panel are explained in section 2.4.1 and will not be repeated here. Planes in EasyRay are defined by a point and a normal vector. They have two default colors, which alternate in a checkerboard pattern. The size of the pattern is specified by the “Pattern Size” info box, and the direction the squares face is defined by the “Pattern Direction” info box. If it is wished, this pattern can have a specified image which repeats, this is defined by the “Texture” field.

2.4.5 Point Light

A point light is not part of the geometry hierarchy, but it is defined like geometry because it exists in the scene like an object. The Point Light panel looks like Fig17 below.

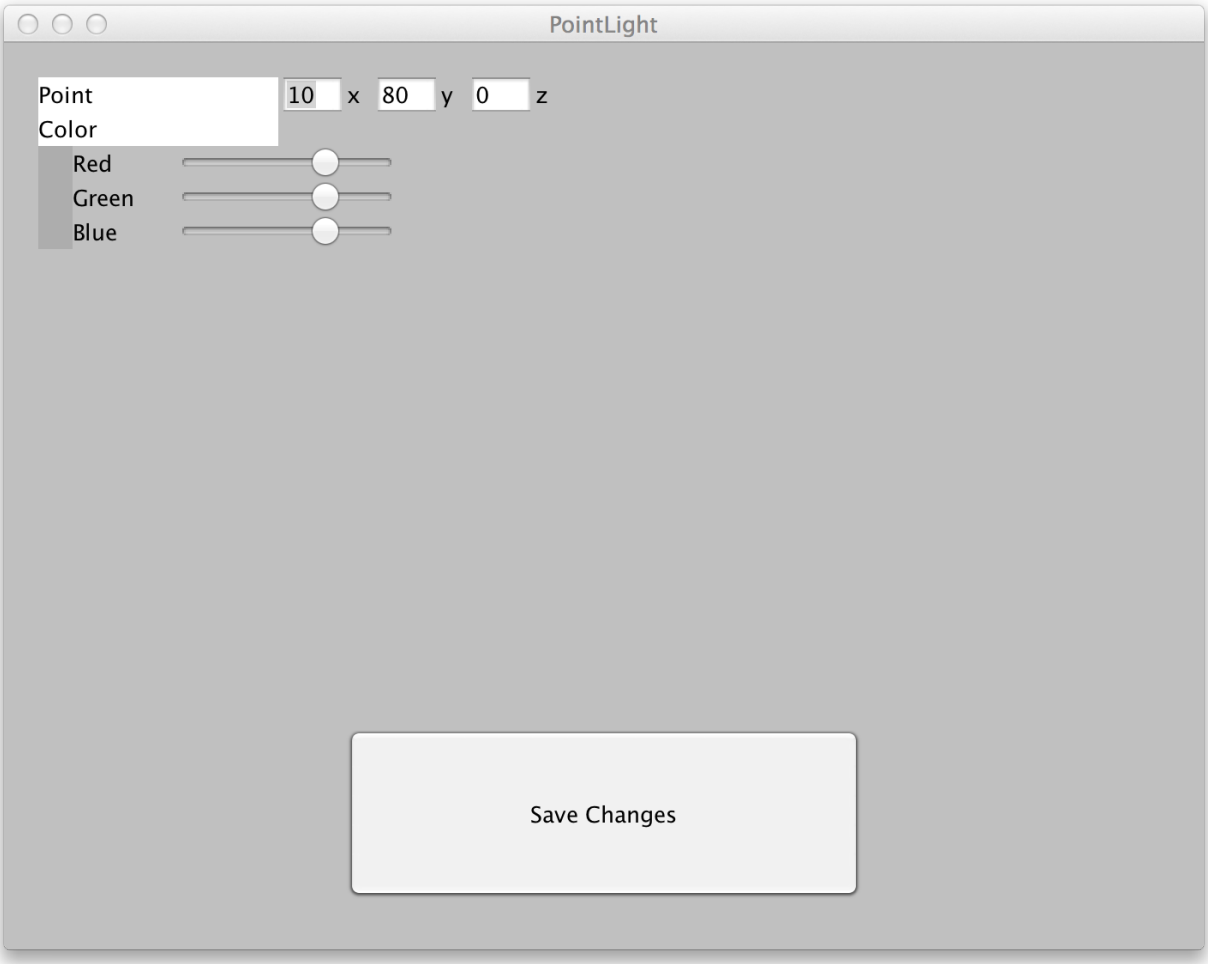


Fig17

This is a very simple panel, with only a point and a color to specify.

2.4.6 Mesh

A “Mesh” represents a triangle mesh, defined by a .obj file. It is not technically geometry (it is a wrapper for triangles), but it is treated like one object for simplicity. The Mesh Panel looks like Fig18 below.

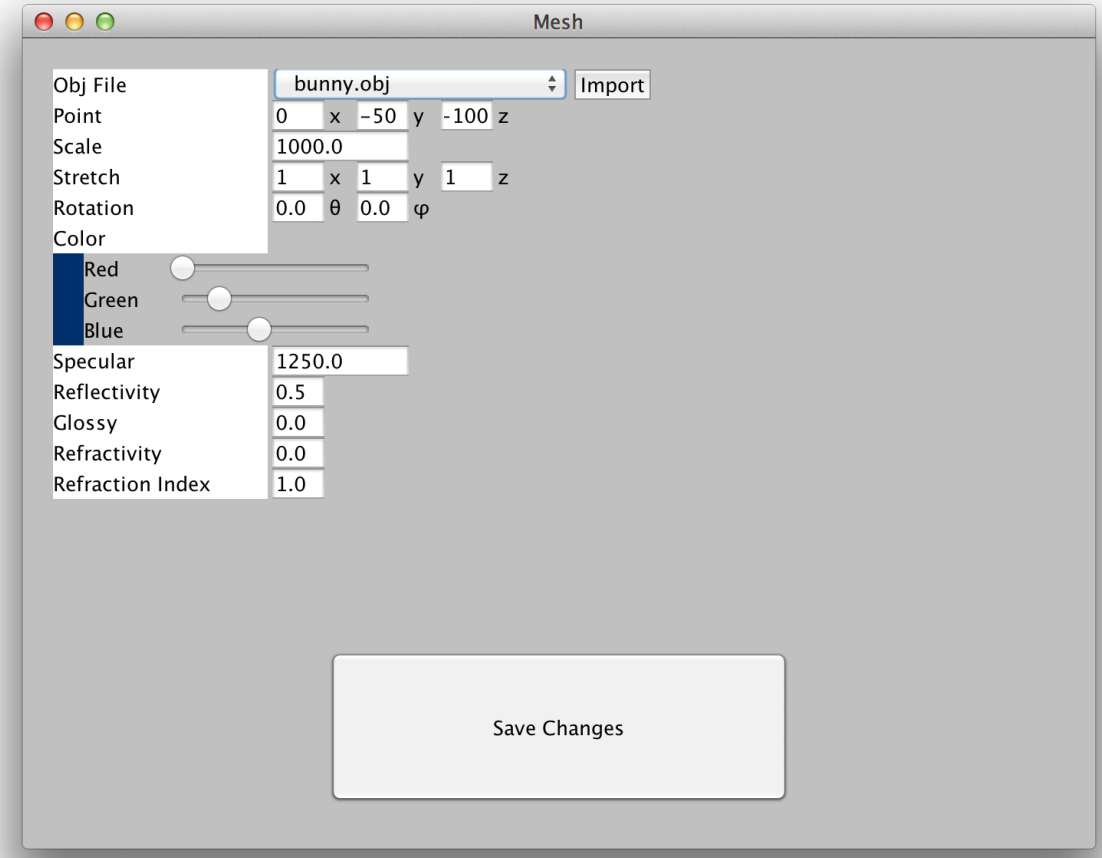


Fig18

This panel is relatively complicated. It first contains a drop down box for an .obj file, and a center point.

The Mesh is positioned by creating it at the origin, scaling it, rotating it, and transforming it. Scale moves all vertices out by the specified multiplier. Stretch performed scale, but specifies different multipliers for each axis. Rotation is specified in spherical coordinates of theta and phi, though this implementation is not perfect and won't work for some values.

3
Conclusion and Further Reading

This software is not meant to be a competitor for any existing commercial ray tracing software in existence. It was built as a learning exercise for the author both in general programming techniques and in ray tracing. Despite this, it is the hope of the author that examining EasyRay's behavior and inspecting the source code will help the reader become familiar with ray tracing, as the system is sophisticated enough to avoid triviality, but not sophisticated to the point that it is opaque to ray tracing beginners.

For further reading, the author has the following suggestions.

- https://graphics.pixar.com/library/MultiJitteredSampling/paper.pdf
- http://www.raytracegroundup.com/index.html
- https://www.ics.uci.edu/~gopi/CS211B/RayTracing%20tutorial.pdf
- https://www.scratchapixel.com/index.php?redirect

Thank you for reading - enjoy EasyRay!