# 6 Appraisal

## 6.1 Evaluation Against Objectives

### 6.1.1 Meeting General Objectives

1. Images can be rendered in real-time with a generally high framerate and angles are also displayed in a readable way. Contrasting colours are used for different types of object and text, producing images suitable for greyscale printing and reading at a greater distance.
2. Users can choose between angles in degrees or radians and can decide whether or not angles should be displayed for a particular light beam.
3. Users can change materials, shapes, beam angles and the number of light sources in order to produce a desired diagram or investigate the effects of angles and materials individually. Questions in two or three dimensions can also be set, but it is difficult to answer questions that weren't set using the software because the precision in setting angles is limited by the height of the sliders in pixels.
4. The software runs smoothly on school computers while the user orbits and zooms in real-time. However, there is no panning functionality because it is not essential and it is generally not so easy to use in 3-D space, so I would also have needed to implement ways of user-friendly ways of focusing on an object (so that the camera orbits around it). The application also runs smoothly while changing materials or shape, but can sometimes be slow when moving /orientating a ray box with the shape as a sphere, a convex lens or a concave lens.

### 6.1.2 Meeting Specific Objectives

- Input objectives:
    1. Users can turn perspective mode on and off through the menu bar or by pressing 'P' in the viewport. Orbiting and zooming is achieved using the mouse, but the user doesn't have the ability to pan. There are preset views which can be switched to through the menu bar or their corresponding keyboard shortcuts in the viewport. The user is restricted from zooming too far out or in so that they can quickly return to a normal level of zoom.
    2. The user can add a ray box with a light beam using the menu bar and one can be deleted (along with its light beam) by selecting it and pressing the delete button in the properties panel. No more than 10 ray boxes can exist at once in order to keep the application running smoothly without using too much processing power.
    3. The target object's shape can be selected from a menu list accessed through the menu bar. The available shapes are 'Cube', 'Cuboid', 'Triangular prism', 'Sphere', 'Convex lens', 'Concave lens' and 'Half-cylinder'.

4. A label can be set for each ray box (the default label is “Ray box”) by selecting it and changing the string in the label text field. Other objects cannot be labelled because each beam can be identified by its ray box and there is only one target object which is easily identified by its colour, shape and position. Labels are in blue to make them stand out (as nothing else in the viewport is blue) even in greyscale printing.
5. The visual properties of light beams can be set individually by selecting the beam’s ray box and using the form at the top of the properties panel. However, the beam’s colour can’t be changed and wavefronts cannot be displayed because I lacked the time needed to implement these.
6. The user can set the world and target materials independently via the menu bar and the menu bar also includes a button which opens a dialog where the user can specify the name and absolute refractive index (between 1 and 100) of a new material which will be added to the two lists in the menu bar.
7. The user must set angles by selecting a ray box and moving the sliders in the properties panel; allowing a ray box to be dragged would have made it hard for the user to change heading and pitch independently and typing an angle would be ambiguous due to the infinitely many positions and orientations of the ray box that would lead to that angle.
8. Light frequency or wavelength would have only a minute effect on the angle of refraction and the magnitude of this effect cannot be determined from a angles and refractive indices alone, meaning that this is beyond what needs to be known for examinations (it is also unlikely to be useful for challenge questions because it relies on further knowledge rather than a good understanding of what has been taught and the ability to apply mathematics) and implementing the effect for custom materials would require users to enter other parameters which they likely will not understand or need. Therefore, this is not a feature of the software.

- Output objectives:
    1. The content of the viewport always depends on the settings of the application and is updated whenever any of these are changed.
    2. The position and orientation of a ray box is updated in real-time when a slider is dragged (including while the slider is still being dragged). The relative position is also dependent upon the camera’s position and orientation which are also updated in real-time as the user drags or scrolls in the viewport
    3. The paths of light beams are always calculated based on the positions and orientations of their ray boxes, the current shape of the target object and the materials of both the world and the target. Snell’s law is followed and total internal reflection can also occur.
    4. Labels and angles are not 3-D objects and are overlaid on top of the rendering of 3-D space using constant font sizes, making them readable at any level of zoom unless labels and/or angles overlap each other.

5. Features and functionalities are grouped logically such as all individual ray box and light beam properties being accessed through the properties panel whereas all settings that apply to the application as a whole are accessed through the menu bar. Within the properties panel the top section is purely for visual properties or deleting a ray box, while the bottom section is all about ray box position and orientation. The menu bar is split into menus and within these there may be submenus. This logical grouping is intended to help users find the particular function they are looking for. The current values for settings and parameters are indicated by checkboxes and radio buttons, populated text fields, the positions of sliders and angles displayed in the viewport.
6. Users can save an image of the viewport to a GIF, JPEG or PNG file anywhere they have permission to write to on a secondary storage device. The user must enter the filename and it must not contain special characters such as ':' and '?'.
7. With this in mind, the background is black and everything else is quite bright. Different types of object as well as labels and angles are also quite different in brightness; angles are bright white, ray boxes are a darker white, beams are bright red, the target object is green (and semi-transparent) and the ray box labels are a somewhat dark blue.

- Processing objectives:
    1. Vertices are transformed so that the scene can be rendered from the camera's point of view taking perspective mode into account; the only problem with the rendering is that vertices behind the camera are sometimes treated as though they were in front of it. I believe this problem could be solved by clipping polygons that are only partially in view (such as by the Sutherland-Hodgman algorithm), but I haven't had the time to implement this. Each face of an object may have a different colour depending on how much it points towards the camera in extended screen space in order to give the illusion of the scene being lit from the position of the camera. Faces are also rendered correctly according to their depth and faces can be seen through other semi-transparent faces.
    2. All inputs are validated and invalid data is corrected or changed to an appropriate value or the user is notified of the invalid entry. When a value is changed, the relevant parts of the user interface are updated, such as a ray box label in the viewport or whether perspective mode is checked in the menu bar.
    3. Each beam path is based on its ray box's position and orientation, the angles it meets faces at and the absolute refractive indices of the world and target. The path is recalculated when any of these variables are affected and the geometry of each beam is generated from its path and regenerated when the path changes.

4. File names are validated and the file format is determined from the typed file extension or selection from the file type dropdown box (in which case the corresponding file extension is appended to the filename). The user is alerted if a file with the same path already exists and the user can choose to overwrite the file or cancel saving. The user is also notified if they attempt to save to an invalid area or if there is not enough storage capacity.

- Storage objective:
    1. Images are stored as 24-bit colour depth bitmaps in GIF, JPEG or PNG format on a secondary storage device such as a hard disk, memory stick or CD. Because they can be stored in any of these common file formats, they can be opened by many applications and printed for insertion into students' books and folders.

## 6.2 User Feedback

[Omitted]

## 6.3 Analysis of User Feedback

The feedback from my client, Mr. Wilkinson was very positive and indicates that my application meets all of his needs and is very useful, with his only criticism being the difficulty in moving ray boxes.

## 6.4 Possible Extensions

As stated earlier, I did not have time to implement visualisation of wavefronts or a colour picker for light beams. Wavefronts would be best implemented as 3-D objects rather than 2-D overlays so that they would stay in position as the camera moved and would have accurate spacing which would be difficult to emulate in 2-D. Java has a built-in class for colour choosers, so implementing different coloured light beams wouldn't be very difficult, as I could add a colour chooser to the properties panel or a button that shows the current colour and opens a dialog where the user can choose a new colour.

Another feature that would have been useful is the visualisation of normals to the points of contact between beams and faces. These 2-D lines drawn on top of the 3-D rendering; one end would be a turning point of the beam's path (these points are stored in the beam object) and the other would have been calculated along with the beam's path and also stored in the beam object. This could also lead to arcs being drawn between the beam and the normal, indicating the angle of incidence or the angle of refraction. Arcs would also be in 2-D because there are methods for drawing arcs, whereas 3-D objects are composed of straight edges and need to have thickness in order to be viewable from all angles.

I also would have liked to implement polygon clipping in an attempt to remove the rendering bug where vertices behind the camera are treated as though they were in front of it. The generation of beam geometry could also have been improved to give each beam a rounder cross-section (using more vertices in each ring) and more consistent thickness (by rotating each ring of vertices halfway between the incident and refracted rays). To avoid angles overlapping or being confused for each other there could be a system to detect the length of the next ray and hence the distance of an angle from the point of contact with the face could be set more appropriately.