Project Report : Final Evaluation

CSE333 : Computer Graphics

Group - 4 : Bhavye Gupta (2017038), Karan Tandon (2017058)

We have implemented a single-pass Volume Renderer that uses Ray marching in OpenGL and QtCreator.

Milestones achieved:

- Volume Ray Marching
- Ray - OBB intersection
    - Sampling and Compositing
- Real-Time Volume Rendering
- Early ray termination optimisation
- Intensity to RGBA transfer function
- User modifiable transfer function mappings
- Using volume datasets
    - Creating a proxy cube
    - Volume datasets (e.g., from .raw files) fully supported
    - Implicit models also supported

Bonus milestones achieved:

- Blinn-Phong shading (For implicit modelling)

Algorithms and implementation details:

1. **Ray marching**: We implemented volume rendering using ray marching for this project. What it does is shoot out a ray from the eye of the camera towards each of the pixels. If the ray intersects an object in the scene, it records the starting point and the ending point of the ray for that object. Then it uses small steps to traverse and composite the values associated with all the voxels on that path. Finally, this value or the output of function which takes this accumulated value is given to the pixel and the pixel is rendered on the screen.

2. **Ray - OBB intersection**: The problem is approached by computing all t-values for the ray and all planes belonging to the faces of the OBB. The box is considered as a set of three slabs. For each slab, there is a minimum and a maximum t-value, and these are called t_i^{max} and t_i^{min}, $\forall\ i\ \in \{u,\ v,\ w\}$. After that compute the t^{min} and t^{max}. Now, if $t^{max} < t^{min}$, then the ray intersects the box otherwise it misses.

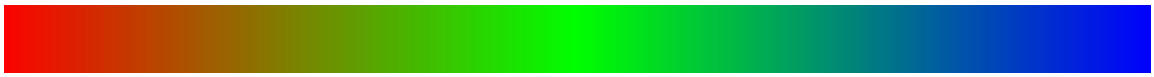
$$t^{min} = min\ (t_u^{min},\ t_v^{min},\ t_w^{min})$$
$$t^{max} = min\ (t_u^{max},\ t_v^{max},\ t_w^{max})$$

Pseudocode

```
    RayOBBIntersect(o, d, A)
    returns ({REJECT, INTERSECT}, t);
1 : t^min = -∞
2 : t^max = ∞
3 : p = a^c - o
4 : for each i ∈ {u, v, w}
5 :     e = a^i · p
6 :     f = a^i · d
7 :     if(|f| > ε)
8 :         t1 = (e + h_i)/f
9 :         t2 = (e - h_i)/f
10 :        if(t1 > t2) swap(t1, t2);
11 :        if(t1 > t^min) t^min = t1
12 :        if(t2 < t^max) t^max = t2
13 :        if(t^min > t^max) return (REJECT, 0);
14 :        if(t^max < 0) return (REJECT, 0);
15 :    else if( - e - h_i > 0 or - e + h_i < 0) return (REJECT, 0);
16 : if(t^min > 0) return (INTERSECT, t^min);
17 : else return (INTERSECT, t^max);
```

3. **Compositing**: We have used the formula provided in the link here (on slide 34).

4. **Transfer Function**: The isovalue obtained is linearly interpolated between two of the three colors depending upon the isovalue.



0.0 (isovalue range) 1.0

5. **Implicit modelling**: In addition to loading volume dataset, we also provide for using implicit models. One such object that you can choose is a sphere centered at the origin with given radius. The sphere is modelled according to the equation:

(**p** - **c**).(**p** - **c**) - R^2 = 0, where **p** is the vector position of a general point in the space, **c** is the center of the sphere and R is the radius.
For a point on the ray **e** + t.**d** to lie on the sphere it must satisfy this equation. After, substituting **e** + t.**d** in the equation of the sphere, we checked whether the result was <= 0, implying the point is inside the sphere.

6. **Blinn - Phong Shading**: We performed Blinn-Phong shading according to the following equation.

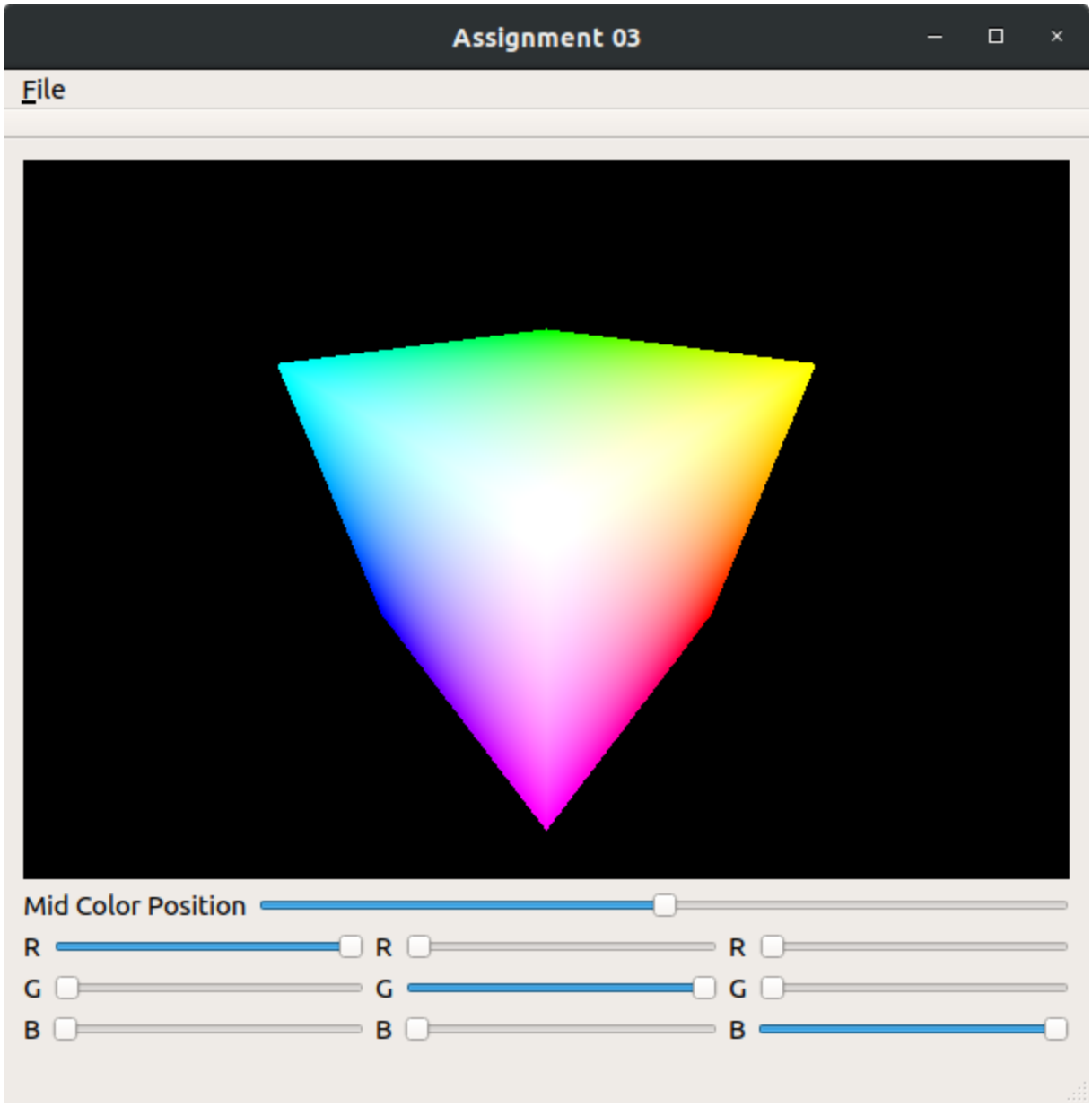
$$C = k_a L_a + k_d L_d max(n.l, 0) + k_s L_s max((r.v)^{\infty}, 0)$$

7. **Early ray termination**: When the alpha value goes above a certain threshold, we terminate the ray, and display the accumulated color.

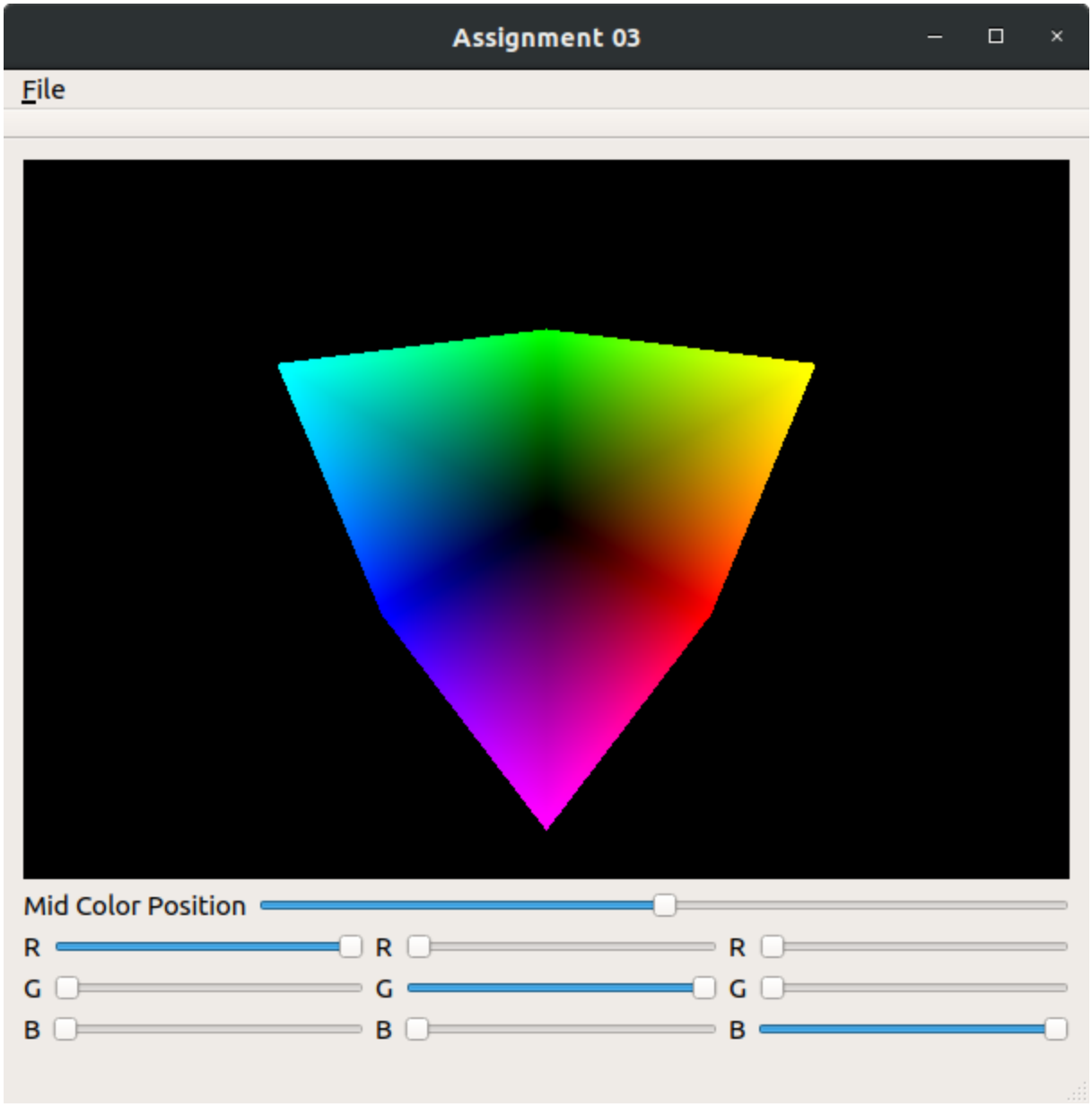
8. **Ability to change transfer function**: We have provided a GUI which allows the user to change the transfer function easily in real-time.

9. **Issues faced**: The perspective of the loaded volumes is not correct

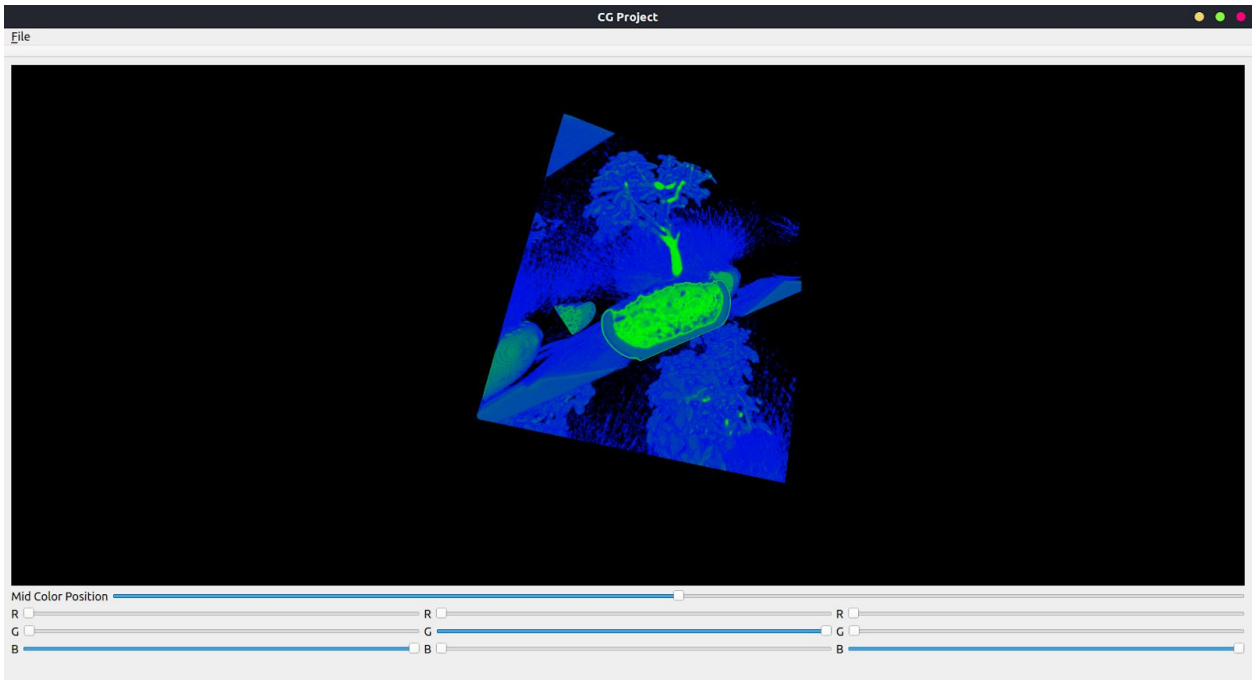
Results: (next page)



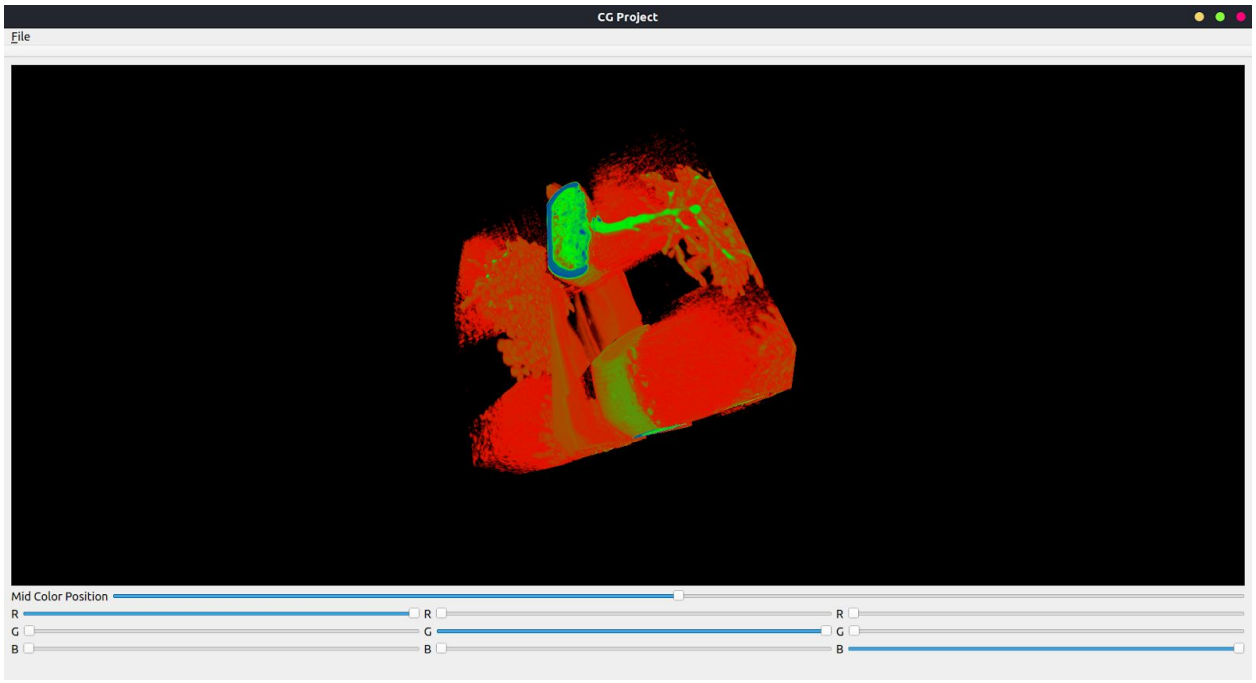
We computed the entry points during ray marching for the proxy cube. Then we used that as the color itself.

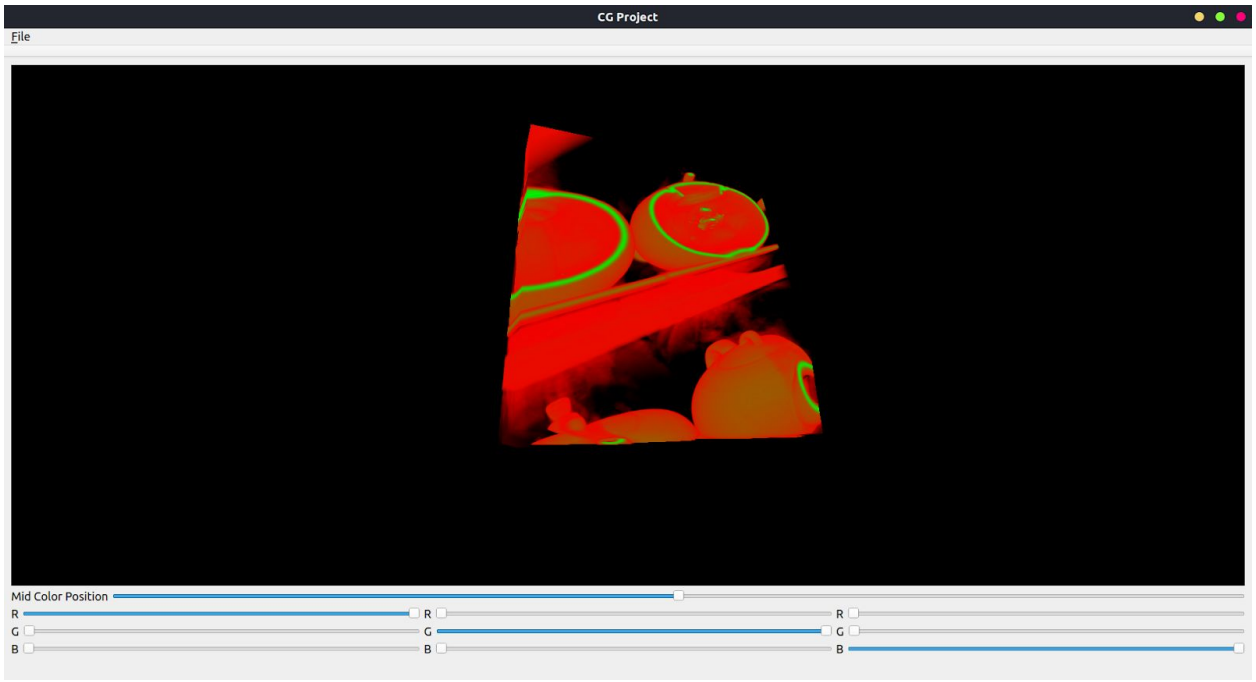


We computed the exit points during ray marching for the proxy cube. Then we used that as the color itself.

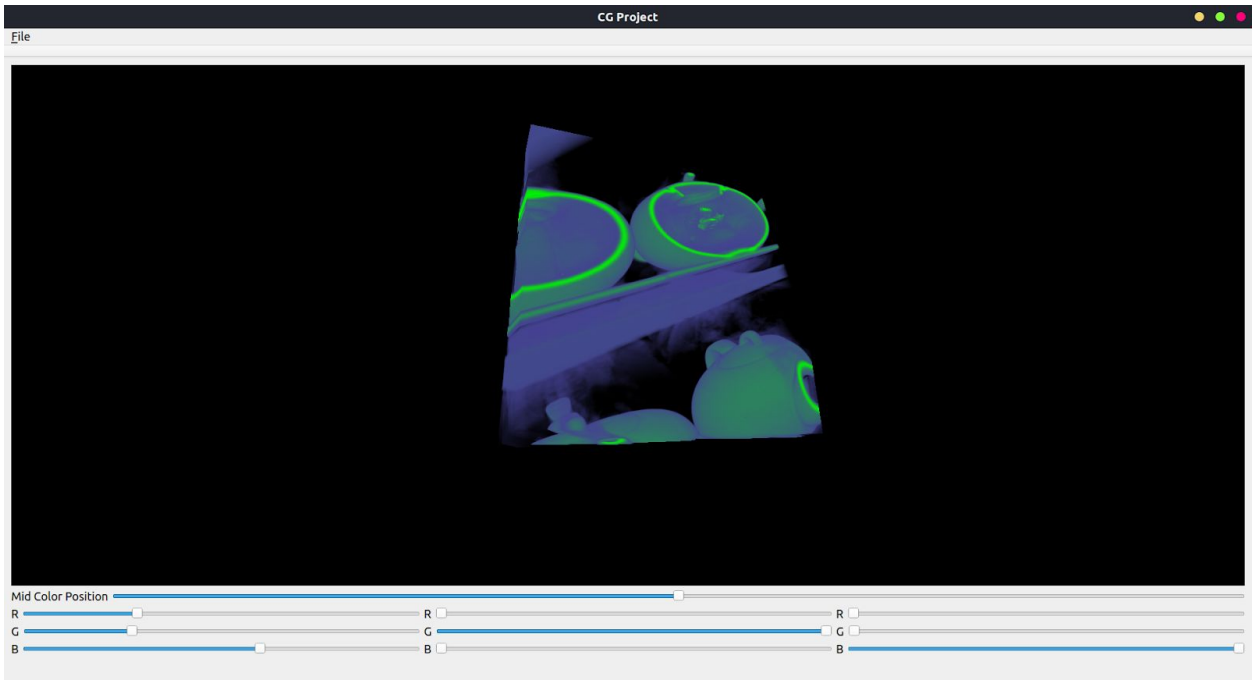


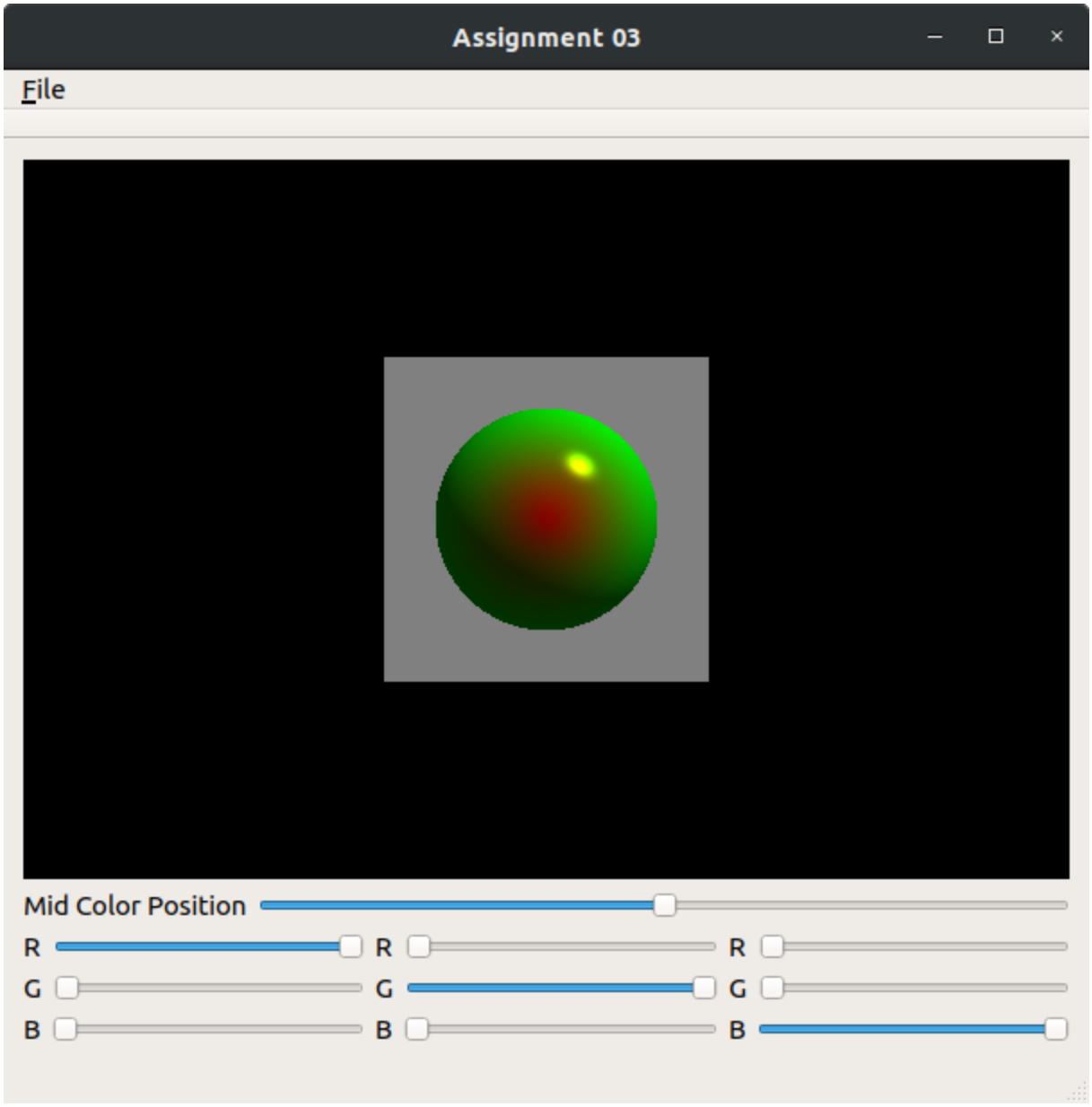
Bonsai tree (256 x 256 x 256)



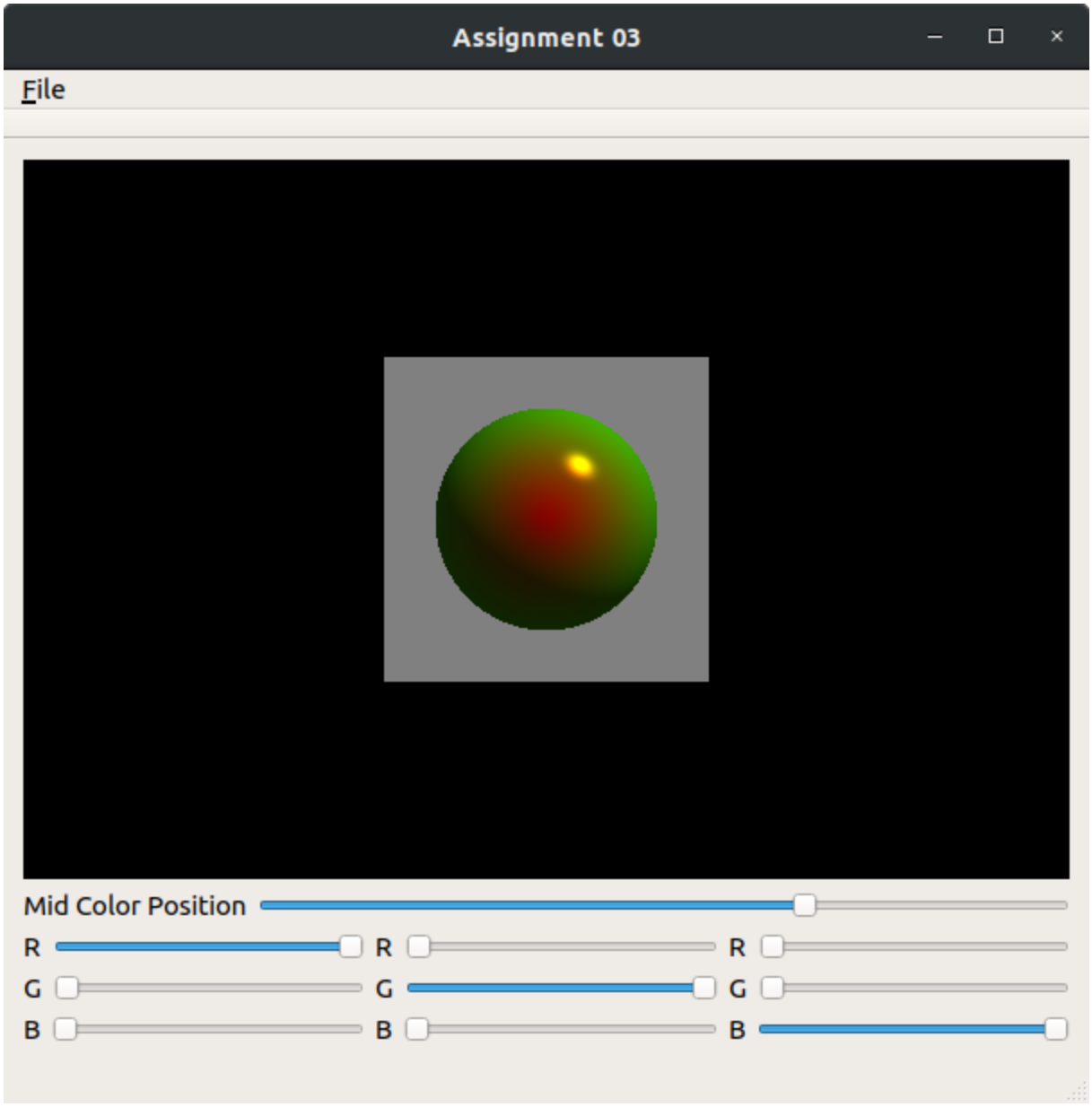


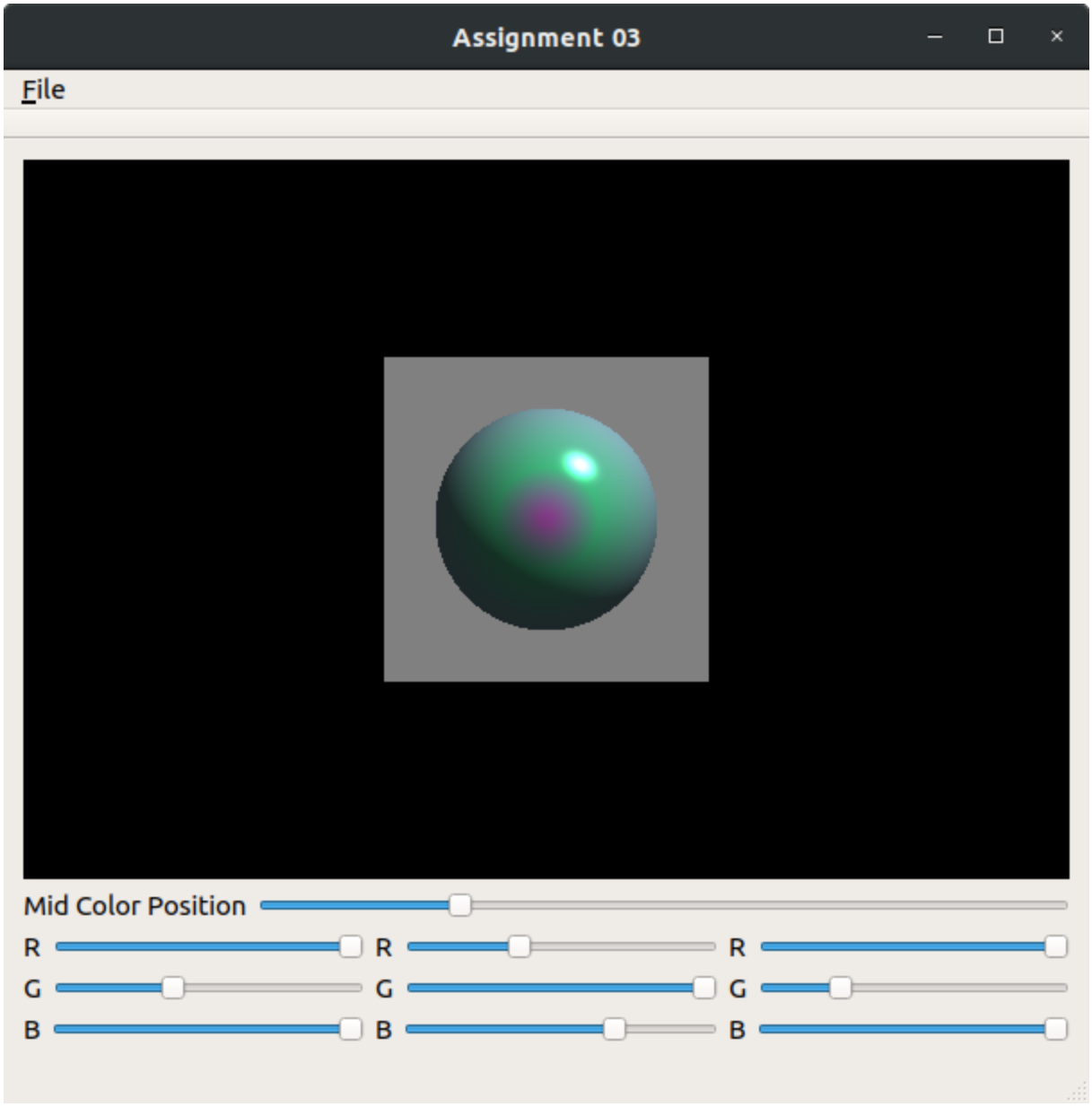
Teapot (256 x 256 x 178)

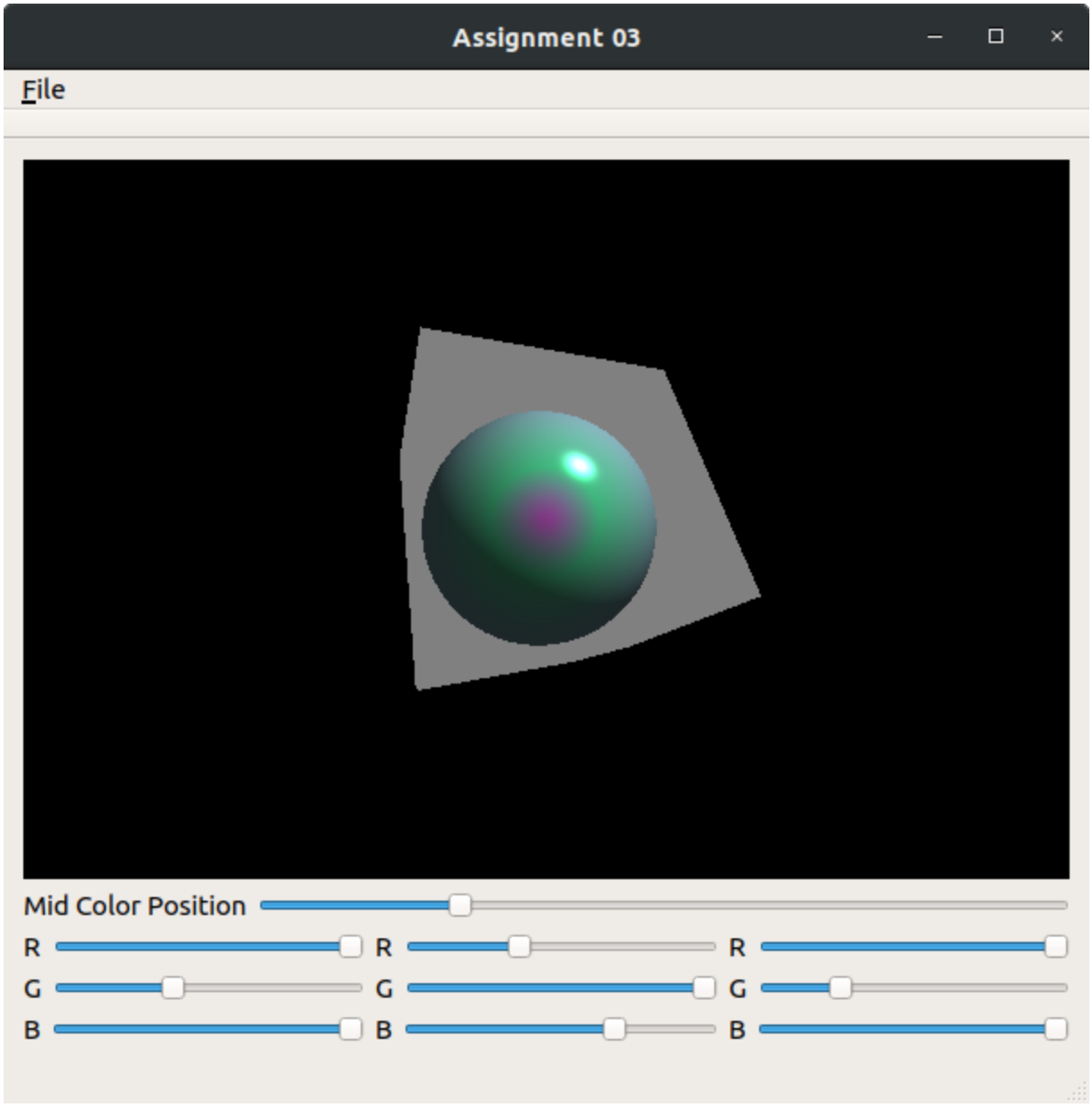




Implicit model of a sphere


Assignment 03
File
Mid Color Position
R
R
R
G
G
G
B
B
B


Assignment 03
File
Mid Color Position
R
R
R
G
G
G
B
B
B


Assignment 03
File
Mid Color Position
R
G
B
R
G
B
R
G
B

References:

- http://antongerdelan.net/opengl/raycasting.html
- Tomas Akenine-Moller, Eric Haines, and Naty Hoffman. 2008. Real-Time Rendering (3rd ed.). A. K. Peters, Ltd., Natick, MA, USA.
- http://web.cse.ohio-state.edu/~parent.1/classes/681/Lectures/VolumeRendering.pdf
- Lecture slides, assignments and labs.