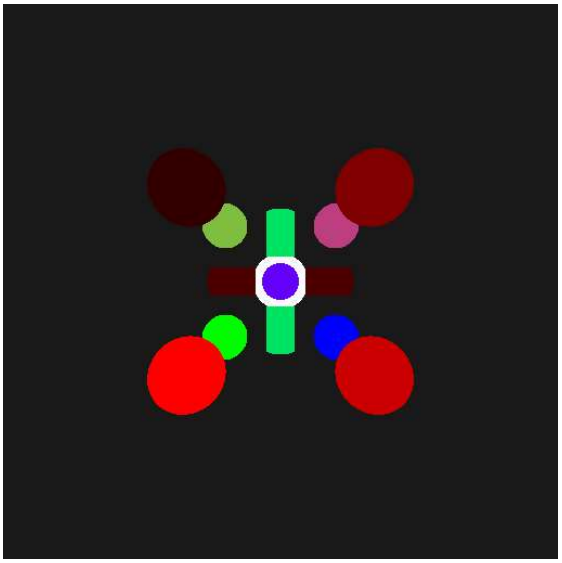
CSCI 5607 HW1A: Ray Casting

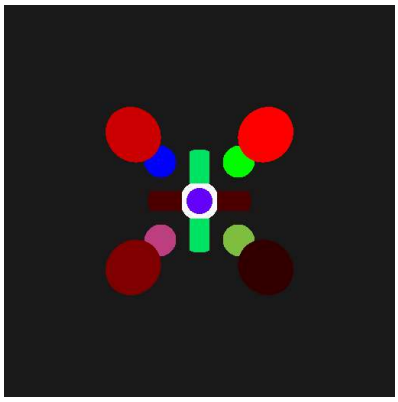
Noah Hendrickson

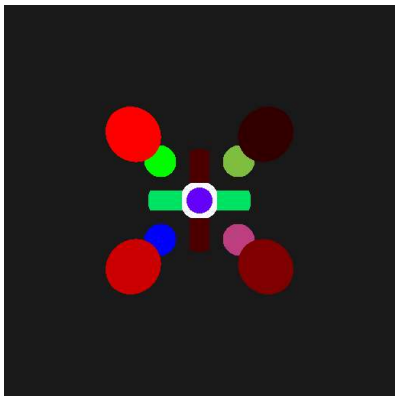
Base Scene:



This base scene was created with a size of 512x512, an eye position of <0,0,0>, viewdir = <0,0,1>, hfov = 90, updir = <0,1,0>, bkgcolor = (0.1,0.1,0.1) and various objects centered around a white sphere at <0,0,10>. Spheres have a radius of 1 and cylinders a radius of 0.1 and length of 5. The red spheres denote the face of the cube that faces towards the origin. This scene will be used to see the effects of all the different changes in parameters. None of the object positions or colors will be changed through the testing. Each intermediate image and its parameters will also be kept in an "inputs" and "outputs" folder.

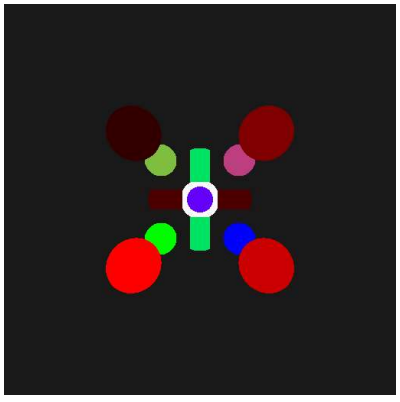
Varying updir:



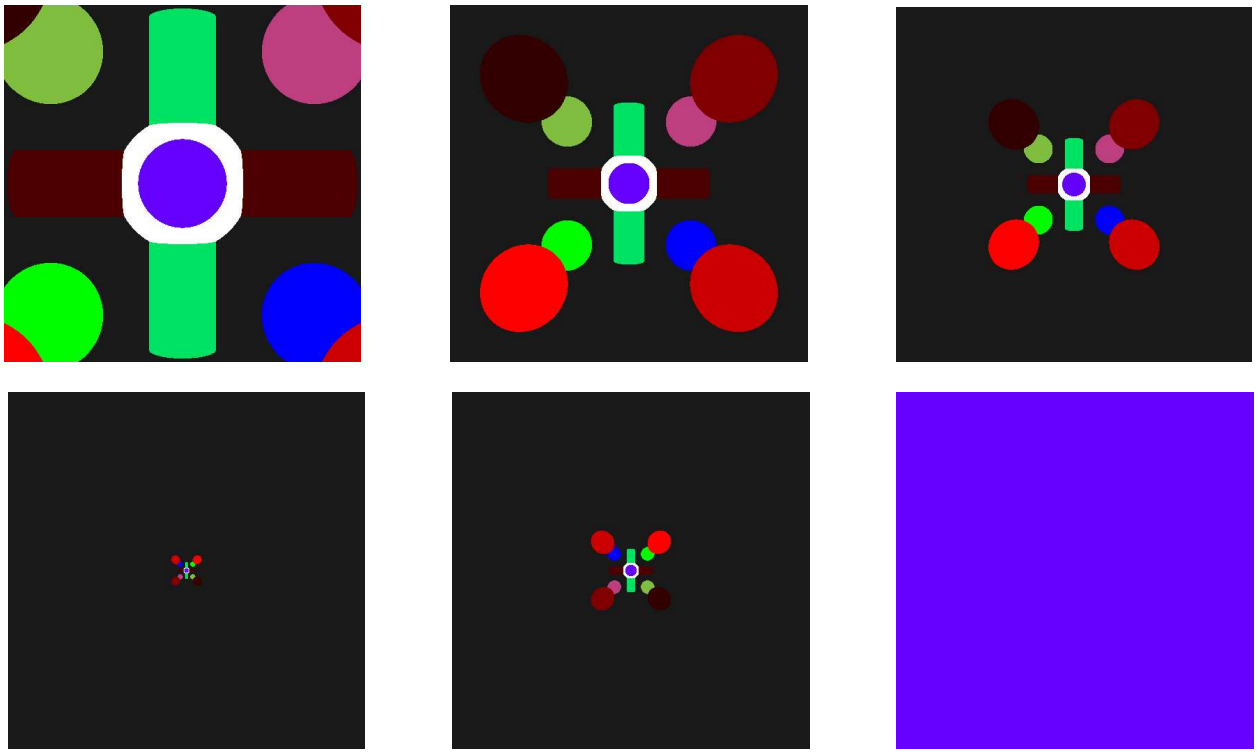


The first of these two images on the left shows an updir = <0,-1,0>. The second image on the left shows an updir = <1,0,0>. When the up direction's y component is inverted, it is like the entire scene was rotated 180 degrees–it was flipped and mirrored. When the up direction is only in the positive x direction (the second image to the left) it is like the entire scene was rotated 90 degrees in the positive x direction (to the right). From this we can glean the information that the direction of the up vector determines the roll around the forward axis.

It is interesting to note, as well, the third image to the right. The third image has updir = <0,0,1> which is the same as the viewing direction. This is interesting because it is the same direction as the viewing direction! A little noise was added to the viewing direction to solve for the fact that you can't take the cross product, but the important point is that the image didn't actually change at all! This is because no matter the updir, the u produced by the cross product between view and up will be the same because changing the updir (if it is on the same axis as the view) doesn't actually change the plane they span, thus, the same image.

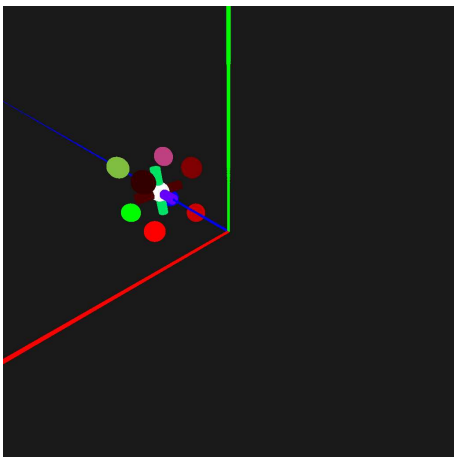


Varying hfov:



I think messing around with fov is always very cool. I love seeing in games how, as you make your FOV higher, your screen gets more stretched. This is what is happening here. Starting from top left, and reading from left to right the hfov's go 30, 60, 90, 200, 230, 360 A very low FOV will get you an image that is very small. It's like looking through a really small window into a big world. You can really only see what is right in front of you. As you make that window bigger, you get to see more and more. However, as you may notice, what is represented here is, as the FOV gets larger, the image goes out but then starts coming back in! And it is flipped and mirrored! I'm not entirely sure why but it sure is neat! 🙂

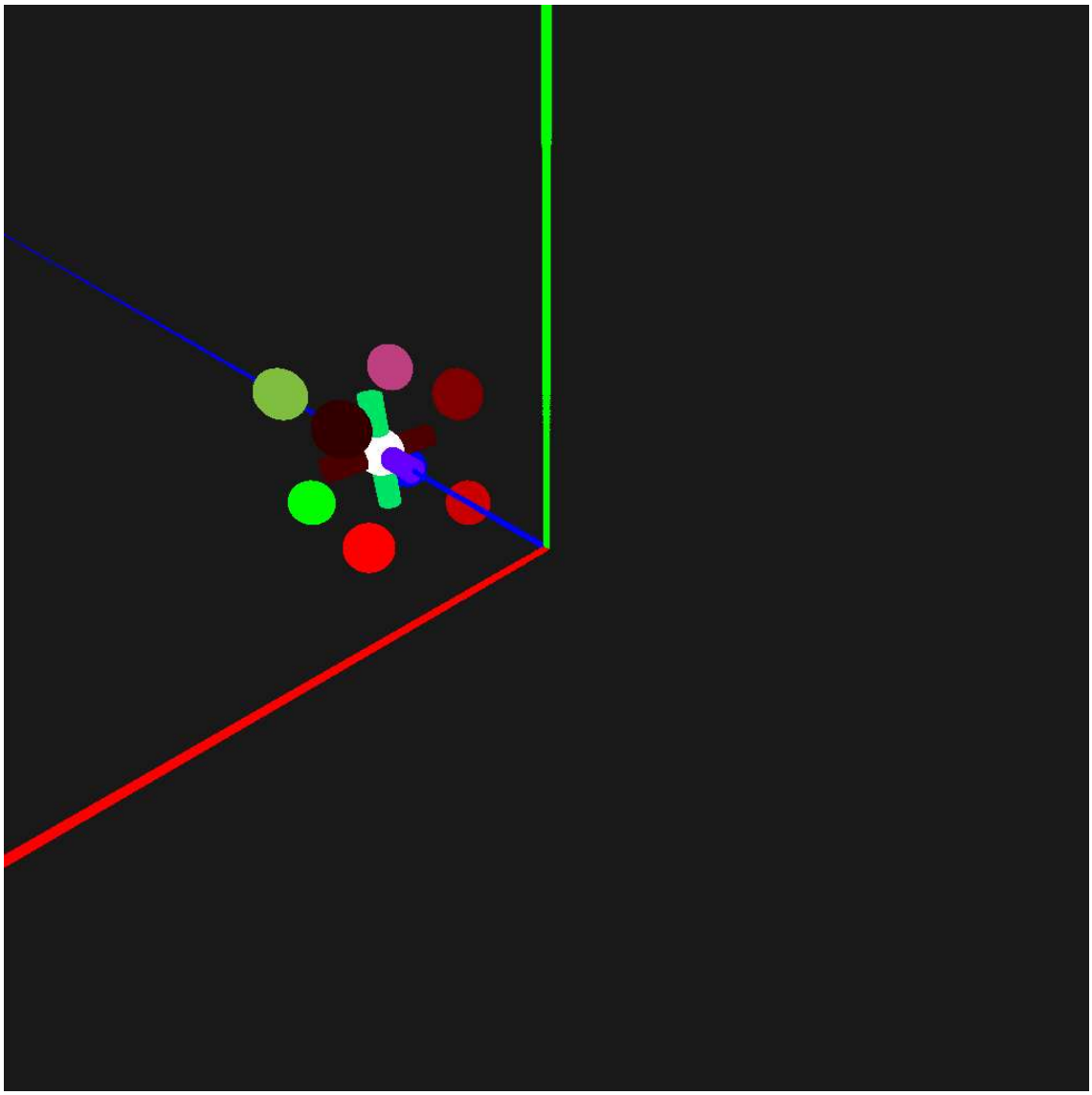
General Distortion (Or Not):



I think a good example to start with here is the image to the left (general1.ppm). This image varies the viewing direction, eye position, and adds in some axes (for visualizing). I think this does good to show how it does not introduce much distortion. Moving the eye and the viewing direction will not distort nearly as much as changing the FOV or changing the up direction (though even between those two things, the up direction does not "distort" much). This can be seen even between the last two examples. The FOV is easily the biggest distorting parameter. Raising the FOV will distort more, while lowering it will distort less, until a point where it will get too close and start distorting more again. I found that the most "natural" looking FOVs occurred between 30 and 90.

General:

Couple things I wanted to mention here that stood out to me. It was really interesting to see what would happen when the view direction and the up direction were the same. Sometimes unintended rotation would occur, and sometimes it would just not be visible at all. I solved this by just adding a very small amount of noise to the y value of the viewing direction, nearly unnoticeable to the eye, and I have not seen it not work since then. Another interesting thing is that, especially in bigger versions of these smaller resolution images (such as below), the jagged edges are very noticeable. It is very easy to tell that only one ray was used in the center of the pixel, and its clear that shooting multiple rays per pixel would be a good idea.



The third and final thing was just an example of the parallel view, using this general image above. (it's on the next page 😛)

I just think it’s really cool! I’m not entirely sure why in parallel the cylinders in the y and x don’t go to their maximum distance but uh yeah. Also, I noticed that when rendering the opposite end of a really long cylinder to the starting position, it got really jagged. And when I say really long I mean like 3000 units. Probably just numerical issues but idk it could be in my calculations as well. (if there’s any feedback to give on that I would very much appreciate some)