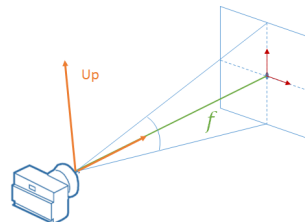


1.1 Primary Ray-Generation for a Perspective Camera Model (30 Points)

Have a look at `camera/perspectivecamera.cpp` and fill out the missing section. A *Perspective Camera Model* can be defined by the following parameters:

- Camera origin (center of projection) **position**
- Viewing direction **forwardDirection**
- (Vertical) full opening angle *angle* of the viewing frustum (in degrees) **fovAngle**
- Up-vector **upDirection**



Given the above camera description, derive the **ray.direction** from the camera to a relative screen coordinate $x, y \in [-1, +1]$. The projection plane is perpendicular to the **camera.forwardDirection**. You will have to incorporate the *focus* (distance from camera position to image plane along the **forwardDirection**).

You will incorporate the *aspect ratio* as part of the `SimpleRenderer` class in exercise 1.3. You can achieve different aspect ratios by not using the entire ranges for x and y in the rendering function.

For example a 16:9 image would use $x \in [-1, +1]$ and $y \in [-\frac{9}{16}, +\frac{9}{16}]$.

1.2 Ray-Surface Intersection (50 Points)

Given a ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$ with origin $\mathbf{o} = (o_x, o_y, o_z)$ and direction $\mathbf{d} = (d_x, d_y, d_z)$, derive the equations to compare the parameter t for the intersection point(s) of the ray and the following implicitly represented surfaces:

- An infinite plane $(\mathbf{p} - \mathbf{a}) \cdot \mathbf{n} = 0$ through point $\mathbf{a} = (a_x, a_y, a_z)$ with surface normal $\mathbf{n} = (n_x, n_y, n_z)$, where any point $\mathbf{p} = (x, y, z)$ that satisfies the equation lies on the surface. Use this to fill the missing section in `primitive/infiniteplane.cpp`.
- Consider a triangle with vertices V_0, V_1 and V_2 . Fill the missing section in `primitive/triangle.cpp` using what you have learned in the lecture.
- A sphere $(p_x - C_x)^2 + (p_y - C_y)^2 + (p_z - C_z)^2 = r^2 \Leftrightarrow (p - C)^2 = r^2$ with center $C = (C_x, C_y, C_z)$, radius $r \in \mathcal{R}$ and point $p = (p_x, p_y, p_z) \in \mathcal{R}^3$ on its surface. Compute the values of t for which the ray intersects the sphere. Use this to fill the missing section in `primitive/sphere.cpp`.

1.3 Ray Tracing (20 Points)

Have a look at `renderer/simplerenderer.cpp` and fill out the missing section. Generate a ray and intersect all objects of your scene with it; assign unique colors of your choice to the objects. The program should generate an image which should look like the one on the left. By moving the camera you can create more interesting perspectives. Try to create your own scenes, by manipulating the `ex1.cpp`. Compute the time for image generation using `std::chrono::steady_clock` and print it afterwards. Modify the `common/Ray.h` and `renderer/simplerenderer.cpp` to implement a simple ray counter, which can be used to calculate the number of processed rays per second. Print this number as well (*Hint: A static member variable for the Ray-struct is usually how such counting is handled*).