Graphics and Imaging Lab

Ray tracing

Informática Gráfica

Grado en Ingeniería Informática

Goal

The goals of this assigment are:

- Develop the key structures and functions for rendering algorithms.
- Link mathematical concepts with practical code.
- Be a test benchmark for potential new primitives, sensors or acceleration structures.

1 Sensor

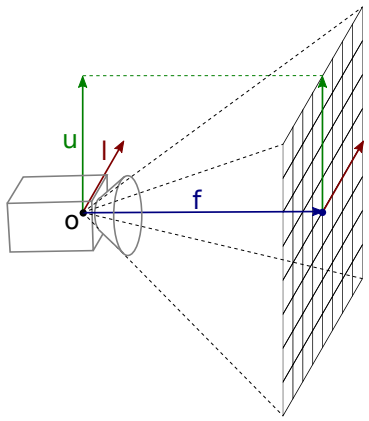

Figure 1: *The geometrical elements that compose a pinhole camera.*

The rays will be cast from a sensor camera model. The model to follow is a pinhole camera (see Figure 1). Bear in mind that the size of the projection plane (given by the up and left vectors) should be proportional to the resolution of the image (in pixels). Otherwise the geometry will appear distorted according to the aspect ratio.

You must know the boundaries of each pixel (related to the projection plane's size and the image resolution) in order to cast multiple rays per pixel. The number of rays per pixel will be a command line parameter.

2 Geometrical primitives

The geometry will consist on the following primitives:

- **Sphere:** given by its center and radius.
- **Planes:** infinite plane given by its normal and its distance to origin.

Each primitive can be intersected with a ray: this results into a system of equations in which the ray has the equation of a line and the specific geometry has its implicit equation. The system has to be solved by substitution to get the parameter of a ray.

Additional to its geometrical information, each primitive will have an **emission** property, an red-green-blue tuple, which represents the color of the primitive.

This application is a good benchmark for testing these new primitives.

3 Rendering

The application will be a renderer that casts several rays per pixel and averages in the corresponding pixel coordinate the emission of the closest intersected primitive per ray. The scenes will be hardcoded in the source code, so therefore there is no need for developing a specific file format. The image will be saved for further visualization.

It is recommended to test the effect of the number of objects and the image resolution on the rendering time.

4 Optional extensions

The students can choose one (or more) of the following extensions, that will enhance further evaluable assignments:

Other geometrical primitives. Add other (several) different geometrical primitives defined by their implicit equation, such as cones, cylinders, ellipsoids, disks or triangles. You can get creative here.

Acceleration structures. For scenes with many objects, implement Bounding Volume Hierarchies or Kd-trees for accelerating rendering. The improvement on rendering time should be evaluated if implemented.

Parallelization. Parallelize the algorithm between several threads of execution. Use a parallelization strategy at image level: subdivide the image into regions (lines / rectangles / columns / pixels) and use a thread-safe task queue to set up the render regions. Enqueue all the render tasks (single producer) and have multiple threads popping tasks and performing them (multiple consumers). Try different strategies: different image regions and image regions sizes, different thread-safe queue implementations and different combinations between parallelization and acceleration structures.

Textures. Add textures that modulate the emission of an object.

Spectral rendering. Instead of having RGB emission properties for each object, include a spectral representation of color that is later converted to RGB using the standard CIE RGB curves. The spectral representation can be a vector of 8, 16 or 32 values, or even a spectral function (a function that depends on the wavelength) that is sampled in the corresponding 8, 16 or 32 values.

Other extensions. Any other extension considered by the students must be first discussed with the teacher, to avoid extensions that require too much work.

5 Implementation details

Programming language. The students can choose any programming language they consider. However, we recommend C++ due to the following:

- The compiled code should be more efficient than other languages.
- The code that will be provided for the second practical work is in C++ (unless you choose to stick with your own code for both of them).
- C++ enables to define operators such as + or * for any new data type. This makes the code more readable for data types such as vectors.

File formats. As you have used it for loading and saving images in previous assignments, we recommend the `.ppm` image format for writing the output of the ray tracer into an image. It is a very simple text file that can be read by most viewers and is very simple to read and write. Find the details in `http://netpbm.sourceforge.net/doc/ppm.html`. Also it can be used to load images as input if the optional extension of textures is included.

If the students decide to include the optional extension of triangle meshes and want to load them, we recommend the `.ply` file format. It is again a very simple text file. The description of such format, along with some free `.ply` models can be found in `https:`

`//people.sc.fsu.edu/~jburkardt/data/ply/ply.html`.

External libraries. The use of any kind of external library or downloaded source code is forbidden. The students are only allowed to use the standard library that comes with the chosen programming language. The only exception to this rule is to include code that is your own previous work (maybe coming from previous assigments at previous computer science subjects). If you include code that is your own previous work, you are required to document it propperly (when did you generate it, for what assignment and what subject).

Submission

This assignment does not have to be submitted for grading. However, it is advised that it is finished by the recommended deadline in order to balance the workload.