MPCS 51087 – Homework 4

Due March 16, 2017, 11:59 P.M.

1 Introduction to Ray Tracing

Ray tracing is a powerful method for rendering three-dimensional objects with complex light interactions. Many other physical phenomena are also analgous to ray tracing – for example, simulating neutrons passing through matter. Figure 1 shows the basic idea for a reflective object. An observer (the eyeball) is viewing an object through a window (the rectangle). The object is illuminated by a light source, which is emitting rays of light (the arrows) in many directions. The observer will see rays that reflect off of the object.

Our task is to render the image seen by the observer through the window. To do so, the simulation can do one or both of the following:

- Simulate light rays starting at the light source and ending at the observer.
- Simulate light rays starting from the observer and going backwards to the light source.

We will implement the latter scheme in this assignment. We will simulate a reflective sphere in black-and-white, illuminated by a single light source. A serial version can be implemented in about 100 lines of code.

2 Vector Notation

Solving the problem involves the use of 3D vectors. A vector $\vec{V}$ has scalar components,$\vec{V} = (V_X, V_Y, V_Z)$. Adding or subtracting two vectors yields a vector, $\vec{V} - \vec{U} = (V_X - U_X, V_Y - U_Y, V_Z - U_Z)$. Multipying a scalar and a vector yields a vector, $t\vec{V} = (tV_X, tV_Y, tV_Z)$. Dividing a vector by a scalar yields a vector, $\vec{V}/t = (V_X/t, V_Y/t, V_Z/t)$. The dot product (a multiplication between two vectors) yields a scalar, $\vec{V} \cdot \vec{U} = V_X U_X + V_Y U_Y + V_Z U_Z$. The norm (or magnitude or length) of the vector is a scalar defined by $|\vec{V}| = \sqrt{V_X^2 + V_Y^2 + V_Z^2}$.

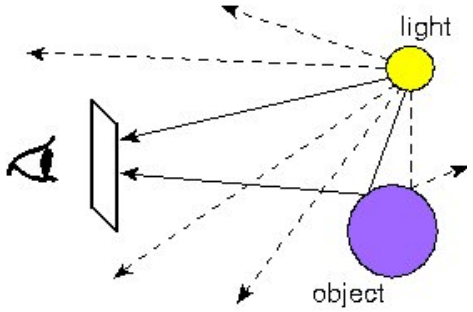

Figure 1: Illustration of Ray-Tracing. Image credit: [1]

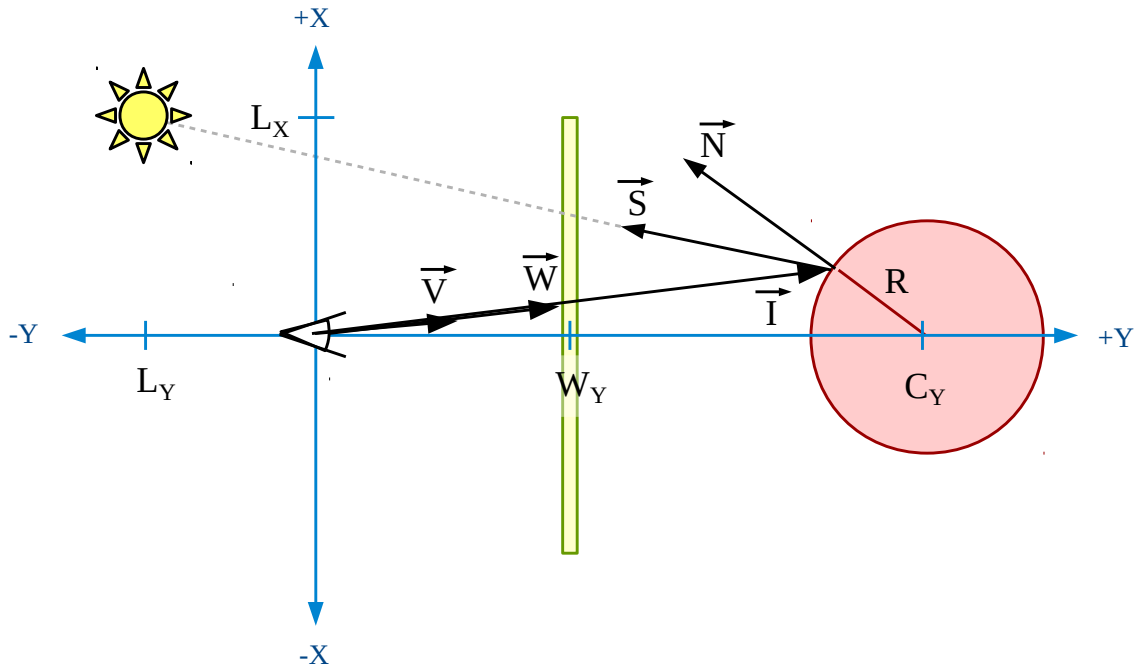

Figure 2: 2D Diagram for Ray-Tracing. Image credit: R. Rahaman

3 Theory

This section describes the theory used in our implementation. The implmentation itself is brief and is described in Algorithm 1 below, so you may skip this section as needed.

Figure 2 shows a 2D cross-section of the problem with the information we need. The observer is at the origin and faces the positive y-direction. The sphere is located at $\vec{C} = (C_X, C_Y, C_Z)$ and has radius R. The window is parallel to the (x,z)-plane at $y = W_Y$ and has bounds $-W_{max} < W_X < W_{max}$ and $-W_{max} < W_Z < W_{max}$. In the simulation, we will represent the window as an $n \times n$ grid, G. The light source is located at $\vec{L} = (L_X, L_Y, L_Z)$.

Our task is to simulate many rays originating from the observer (so-called "view rays") with randomly-selected directions. The steps are as follows:

- **Select the direction of view ray ($\vec{V}$).** Let $\vec{V}$ be a unit vector representing the direction of the view ray. In spherical coordinates, We will randomly select its component angles (θ, ϕ) such that $0 < \theta < \pi$ and $0 < \phi < \pi$. Then we will get $\vec{V}$ in Cartesion coordinates:

$$V_X = \sin\theta\cos\phi$$
$$V_Y = \sin\theta\sin\phi$$
$$V_Z = \cos\theta$$

- **Find the intersection of the view ray with the window ($\vec{W}$).** Knowing that the window is at W_Y, the window's point-of-intersection with the view ray is given by the vector $\vec{W}$:

$$\vec{W} = \frac{W_Y}{V_Y}\vec{V}$$

  If the view ray is outside the window ($|W_X| > W_{max}$ or $|W_Z| > W_{max}$), we reject it and chose a new $\vec{V}$.

- **Find the intersection of view ray with sphere ($\vec{I}$).** Let $\vec{I}$ be the sphere's point-of-intersection with the view ray. To find $\vec{I}$, we solve the following system of equations:

$$\vec{I} = t\vec{V}$$
$$\left|\vec{I} - \vec{C}\right|^2 = R^2$$

These are the equations of the view ray and the sphere, respectively. Solving for t yields:

$$t = \left(\vec{V} \cdot \vec{C}\right) - \sqrt{\left(\vec{V} \cdot \vec{C}\right)^2 + R^2 - \vec{C} \cdot \vec{C}}$$

which can be back-substituted to get $\vec{I}$. If t does not have a real solution ($(\vec{V} \cdot \vec{C})^2 + R^2 - \vec{C} \cdot \vec{C} < 0$), then view ray does not intersect the sphere and we choose a new $\vec{V}$.

- **Find the observed brightness of the sphere (b).** Next, we want to find the brightness of the sphere that is observed at $\vec{I}$. To do so , we:

  - *Find the unit normal vector ($\vec{N}$).* The unit normal vector $\vec{N}$ is perpindicular to the sphere's surface at $\vec{I}$.

$$\vec{N} = \frac{\vec{I} - \vec{C}}{\left|\vec{I} - \vec{C}\right|}$$

  - *Find the direction to the light source ($\vec{S}$).* The direction to light source (somtimes called the "shadow ray") is represented by the unit vector $\vec{S}$.

$$\vec{S} = \frac{\vec{L} - \vec{I}}{\left|\vec{L} - \vec{I}\right|}$$

  - *Find the brightness (b).* The brightness can be found from $\vec{S}$ and $\vec{N}$ using "Lambertian shading".

$$b = \begin{cases} 0 & \vec{S} \cdot \vec{N} < 0 \\ \vec{S} \cdot \vec{N} & \vec{S} \cdot \vec{N} \geq 0 \end{cases}$$

- **Add the brightness to the window's grid.** We find (i, j) such that $\vec{G}(i, j)$ is the position of $\vec{W}$ on the window's grid G and let:

$$G(i, j) = G(i, j) + b$$

4 Implementation

Algorithm 1 describes a ray-tracing implementation. As described above, the observer is at the origin and is facing the positive-y direction. The sphere is located at $\vec{C} = (C_X, C_Y, C_Z)$ and has radius R. The light source is located at $\vec{L} = (L_X, L_Y, L_Z)$. The window is parallel to the (x,z)-plane at $y = W_Y$, has bounds $-W_{max} < W_X < W_{max}$ and $-W_{max} < W_Z < W_{max}$, and is repsresented by an $n \times n$ grid, G.

Algorithm 1 Ray Tracing Algorithm

allocate $G[1 \dots n][1 \dots n]$ ▷ The window is respresented on the grid G
$G[i][j] = 0$ for all (i, j)
for $n = 1 \dots N_{rays}$ **do**
    repeat
        $\theta =$ a random decimal between $(0, \pi)$ ▷ The direction of the view ray in spherical coords
        $\phi =$ a random decimal between $(0, \pi)$
        $\vec{V} = (\sin\theta\cos\phi, \sin\theta\sin\phi, \cos\theta)$ ▷ The direction of the view ray in Cartesian coords
        $\vec{W} = \frac{W_Y}{V_Y}\vec{V}$ ▷ The intersection of the view ray and the window
    until $|W_X| > W_{max}$ **and** $|W_Z| > W_{max}$ **and** $\left(\vec{V}\cdot\vec{C}\right)^2 + R^2 - \vec{C}\cdot\vec{C} < 0$
    $t = \left(\vec{V}\cdot\vec{C}\right) - \sqrt{\left(\vec{V}\cdot\vec{C}\right)^2 + R^2 - \vec{C}\cdot\vec{C}}$
    $\vec{I} = t\vec{V}$ ▷ The intersection of the view ray and the sphere
    $\vec{N} = \frac{\vec{I}-\vec{C}}{|\vec{I}-\vec{C}|}$ ▷ The unit normal vector at $\vec{I}$
    $\vec{S} = \frac{\vec{L}-\vec{I}}{|\vec{L}-\vec{I}|}$ ▷ The direction of the light source at $\vec{I}$
    $b = \text{MAX}\left(0, \vec{S}\cdot\vec{N}\right)$ ▷ The brightness observed at $\vec{I}$
    find (i, j) such that $G(i, j)$ is the gripoint of $\vec{W}$ on G
    $G(i, j) = G(i, j) + b$ ▷ Add brightness to grid

Figure 4 shows a sample image where $\vec{L} = (4, 4, -1)$, $W_Y = 10$, $W_{max} = 10$, $C = (0, 12, 0)$, $R = 6$.

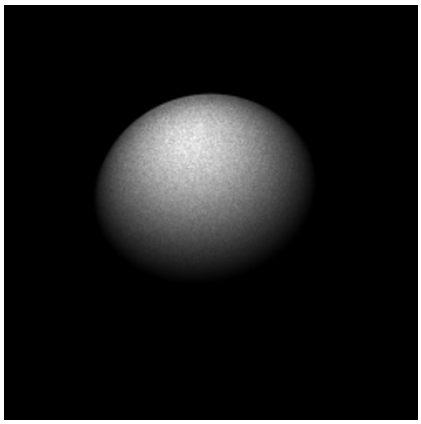

Figure 3: Ray-Traced Render of a Sphere Illuminated from Top Left

5 Questions

1. Make and test a serial version of Algorithm 1. Take user input for the number of rays and gridpoints; the other parameters can be hardcoded. (Hint: Define structs and functions for the vectors and vector operations. With these simple abstractions, the code can be written in about 100 lines.)

2. Use CUDA to parallelize Algorithm 1. Note that, while the rays are independent, updating the window could result in write conflicts. Therefore, some syncronization is necessary.

3. Compare the runtime of the serial version and CUDA versions as functions of problem size (the number of rays). For the CUDA version, use the maximum occupancy of the device.

4. For the CUDA version, perform a scaling study for a fixed, large number of rays and increasing occupany.

5. Show a sample image produced by your CUDA ray-tracing implmenentation.

6 References

1. Rademacher, Paul. *Ray Tracing: Graphics for the Masses.* `https://www.cs.unc.edu/~rademach/xroads-RT/RTarticle.html`

2. *Ray tracing (graphics).* `http://en.wikipedia.org/wiki/Ray_tracing_(graphics)`