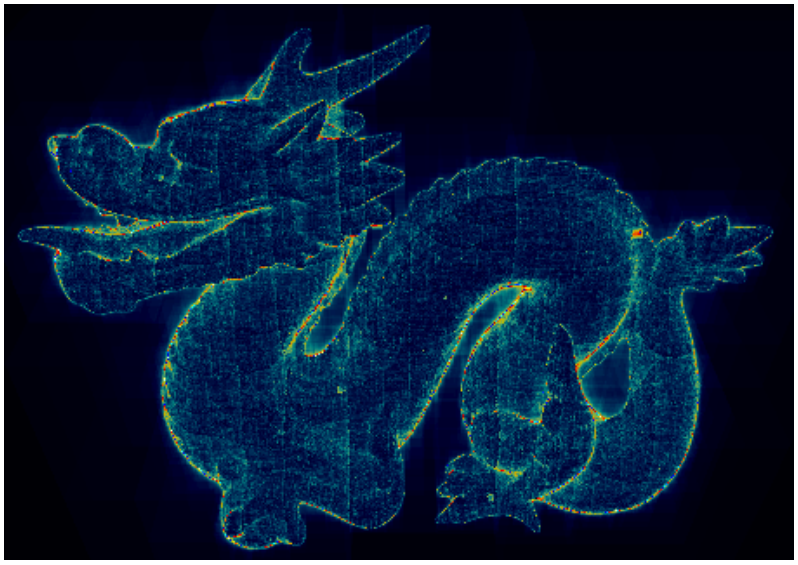
KU LEUVEN

FACULTY OF ENGINEERING

Computer Graphics Project Report



Moritz Wolter
April 8, 2016

Supervised by
Prof. Philip Dutré

Contents

Milestone 1

Essentials

1.1 Geometry

1.1.1 The triangle primitive

This section focuses on the basic math of ray tracing. In a ray tracer so called primary rays are shot from a camera onto a scene. Irreducible objects or geometric primitives are defined. These make up a scene. In order to compute an image from this given scene, the ray-object intersection point closest to the camera has to be found. Various primitives have been implemented, the most important one in the triangle. As triangle meshes can be used, to approximate more complex shapes. A triangle is defined by the three points $\mathbf{a}, \mathbf{b}, \mathbf{c}$, that make up its edges. If the edges are ordered counterclockwise the triagle normal can be computed using the formula: [1]

$$\mathbf{n} = (\mathbf{b} - \mathbf{a}) \times (\mathbf{c} - \mathbf{a}) / \|(\mathbf{b} - \mathbf{a}) \times ((\mathbf{c} - \mathbf{a})\|. \tag{1.1}$$

To find triangle intersection points barycentric coordinates (α, β, γ) are used. These coordinates allow do describe any point in the plane of the triangle as:

$$\mathbf{p}(\alpha, \beta, \gamma) = \alpha\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c}, \tag{1.2}$$

$$\text{with } \alpha + \beta + \gamma = 1. \tag{1.3}$$

If $\mathbf{p}$ is inside of the plane the inequalities:

$$0 < \alpha < 1, 0 < \beta < 1, 0 < \gamma < 1. \tag{1.4}$$

Are satisfied. By substituting $\alpha = 1 - \beta - \gamma$ into the equation and setting $\mathbf{p} = \mathbf{r}(t)$ with $\mathbf{r} = \mathbf{o} + t * \mathbf{d}$ for a ray the following equation is obtained:

$$\mathbf{o} + t\mathbf{d} = \mathbf{a} + \beta(\mathbf{b} - \mathbf{a}) + \gamma(\mathbf{c} - \mathbf{a}). \tag{1.5}$$

Which can be brought into the form $\mathbf{Ax} = \mathbf{b}$ with $\mathbf{x} = \begin{pmatrix} \beta & \gamma & t \end{pmatrix}^T$:

$$\begin{pmatrix} a_x - b_x & a_x - c_x & d_x \\ a_y - b_y & a_y - c_y & d_y \\ a_z - b_z & a_z - c_x & d_z \end{pmatrix} \begin{pmatrix} \beta \\ \gamma \\ t \end{pmatrix} = \begin{pmatrix} a_x - o_x \\ a_y - o_y \\ a_z - o_z \end{pmatrix}. \tag{1.6}$$

[1] This section in based on Ray Tracing from the gound up page 362 and onward.

Equation 1.6 could in principle be given to a linear algebra subroutine for solution. If the inequalities 1.4 are satisfied and $t > \epsilon$. Where epsilon is a small positive number used to prevent self intersection. A valid intersection has been found. However this is not the most efficient way to proceed. Always solving the whole equation system is not necessary all the time. Using Cramers rule the a set of equations can be derived and the intersection method can return as soon as one condition is violated. This way unnecessary computations can be avoided[2].

1.1.2 The plane primitive

Another important primitive is the plane. These objects can be defined by a normal $\ltimes$ which determines, where the plane is and a point $\mathbf{a}$ which determines where it is.[3] For a point $\mathbf{p}$ on the plane the condition

$$(\mathbf{p} - \mathbf{a})^T \mathbf{n} = 0 \tag{1.7}$$

must hold. This is the same as asking for the cosine of the angle between the vector on the plane and normal to be zero. Or equivalently saying that the vectors $(\mathbf{p} - \mathbf{a})$ and $\mathbf{n}$ must form a 90° angle. To check if a given ray intersects a plane the ray must be plugged into equation 1.7 which yields:[4]

$$(\mathbf{o} + t\mathbf{d} - a)^T \mathbf{n} = 0. \tag{1.8}$$

Solving for t then leads to:

$$t = (\mathbf{a}^T \mathbf{n} - \mathbf{o}^T \mathbf{n}))/(\mathbf{d}^T \mathbf{n}). \tag{1.9}$$

If then $t > \epsilon$ a valid intersection was computed. The exact location of the hit point can be found from the ray equation $\mathbf{p}(t) = \mathbf{o} + t\mathbf{d}$.

The rectangle primitive

A rectangle is simply a subset of all the points contained in a plane therefore the same equations can be used. Additionally the conditions

$$\|p_x\| < 1.0, \|p_y\| < 1.0, \tag{1.10}$$

have to be enforced to turn a the x,y-plane into a rectangle around the origin. This unit plane can later be rotated and scaled using transformations to fit into a desired scene.

The circle primitive

A circle in the x,y plane is obtained by a procedure analogue to the one which lead to rectangles if instead the condition

$$\sqrt{p_x^2 + p_y^2} < 1.0 \tag{1.11}$$

is used. This circle can similarly be transformed to fit into any scene.

[2]See Ray Tracing from the Ground up page 366 for further details.
[3]Ray Tracing from the ground up page 31.
[4]Ray Tracing from the Ground up page 54.

1.1.3 The sphere primitive

A sphere is a set of points located within a given distance r around a specified point **c**. For any point **p** within this sphere the condition:

$$\|\mathbf{p} - \mathbf{c}\| \leq r \tag{1.12}$$

$$\text{or } (\mathbf{p} - \mathbf{c})^T(\mathbf{p} - \mathbf{c}) \leq r^2 \tag{1.13}$$

must hold. For a sphere around the origin with radius one $\mathbf{c} = 0$, $r^2 = 1$ and substituting $\mathbf{p} = \mathbf{o} + t\mathbf{d}$ the expression below is obtained:

$$(\mathbf{o} + t\mathbf{d})^T(\mathbf{o} + t\mathbf{d}) - 1 = 0 \tag{1.14}$$

Now using an equality instead of an inequality to compute the intersection with the outer bound of the sphere. The distributive property of the dot product leads to:

$$t^2\mathbf{d}^T\mathbf{d} + 2t\mathbf{o}^T\mathbf{d} + \mathbf{o}^T\mathbf{o} - 1 = 0. \tag{1.15}$$

The obtained quadratic equation can be solved using standard methods with:

$$a = \mathbf{d}^T\mathbf{d} \tag{1.16}$$

$$b = 2\mathbf{d}^T\mathbf{o} \tag{1.17}$$

$$c = \mathbf{o}^T\mathbf{o} - 1 \tag{1.18}$$

$$\text{then } t = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{1.19}$$

It is possible for learn the number of intersections from the discriminant $d = b^2 - 4ac$. If $d < 0$ there will be no intersections zero if $d = 0$ and two if $d > 0$. If two intersections are found the one with the smaller t should be used, as it will be closer to the camera.[5]

1.2 Object Transformations

In order move, rotate or scale the objects defined earlier four dimensional transformation matrices are defined. Four dimensions are used, as three dimensional translation can be expressed as a matrix operation.
Scaling can be done by using[6]:

$$T_{scale} = \begin{pmatrix} a & & & \\ & b & & \\ & & c & \\ & & & 1 \end{pmatrix} \quad T^{-1}_{scale} = \begin{pmatrix} 1/a & & & \\ & 1/b & & \\ & & 1/c & \\ & & & 1// \end{pmatrix} \tag{1.20}$$

Translation is working with:

$$T_{translate} = \begin{pmatrix} 1 & & & d_x \\ & 1 & & d_y \\ & & 1 & d_z \\ & & & 1 \end{pmatrix} \quad T^{-1}_{translate} = \begin{pmatrix} 1 & & & -d_x \\ & 1 & & -d_y \\ & & 1 & -d_z \\ & & & 1 \end{pmatrix} \tag{1.21}$$

[5] Ray Tracing from the Ground up page 57
[6] Ray Tracing from the ground up page 404

Rotation around the tree axes:

$$T_x = \begin{pmatrix} 1 & & & \\ & cos(\theta) & -sin(\theta) & \\ & sin(\theta) & cos(\theta) & \\ & & & 1 \end{pmatrix} \tag{1.22}$$

$$T_y = \begin{pmatrix} 1 & & & \\ & cos(\theta) & -sin(\theta) & \\ & sin(\theta) & cos(\theta) & \\ & & & 1 \end{pmatrix} \tag{1.23}$$

$$T_z = \begin{pmatrix} cos(\theta) & & sin(\theta) & \\ & 1 & & \\ -sin(\theta) & & cos(\theta) & \\ & & & 1 \end{pmatrix} \tag{1.24}$$

$$\tag{1.25}$$

The inverse rotations can be obtained by using $-\theta$ in place of θ or by transposing the matrices. Several transformations can be combined by matrix multiplication.
Instead of transforming objects it is common practice in ray tracing to transform rays. Prior to intersection testing a ray ($\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$) is multiplied with the inverse transformations matrix. The obtained transformed ray is then intersected with the unchanged object. The hit point is found and transformed back into the original space using the transformation matrix. [7] For normals the process is slightly more complicated. [8] In the unchanged space the normal must be orthogonal to an infinitely small vectors along the surface $\mathbf{n}^T\mathbf{m} = 0$. The same must be true for the transformed space $\mathbf{n'}^T\mathbf{m'} = 0$. Like points differences between points will be transformed by the original transformation matrix $\mathbf{Tm} = \mathbf{m'}$. Furthermore a matrix multiplied with its inverse is the identity $\mathbf{T}^{-1}\mathbf{T} = \mathbf{I}$. This leads to the derivation:

$$\mathbf{n}^T\mathbf{m} = 0 \tag{1.26}$$

$$\mathbf{n}^T\mathbf{I}\mathbf{m} = 0 \tag{1.27}$$

$$\mathbf{n}^T\mathbf{T}^{-1}\mathbf{T}\mathbf{m} = 0 \tag{1.28}$$

$$(\mathbf{n}^T\mathbf{T}^{-1})(\mathbf{T}\mathbf{m}) = \mathbf{n'}^T\mathbf{m'} = 0 \tag{1.29}$$

$$(\mathbf{n}^T\mathbf{T}^{-1})(\mathbf{T}\mathbf{m}) = \mathbf{n'}^T(\mathbf{T}\mathbf{m}) \tag{1.30}$$

$$\mathbf{n}^T\mathbf{T}^{-1} = \mathbf{n'}^T \tag{1.31}$$

$$\Rightarrow \mathbf{n'} = \mathbf{T}^{-T}\mathbf{n} \tag{1.32}$$

Thus normals will be transformed by the transposed inverse transformation matrix.

[7] Ray Tracing from the Ground up page 418.
[8] Fundamentals of Computer Graphics, Lecture 3 Transformations, Philip Dutré, slide 60

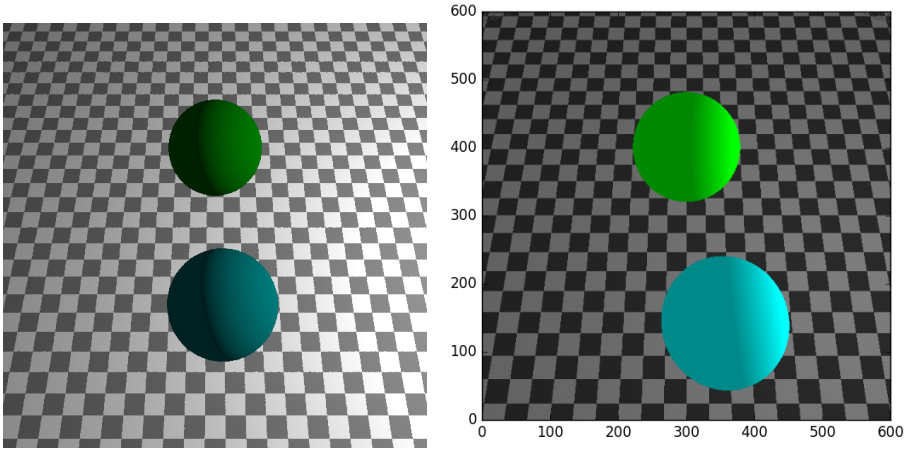


Figure 1.1: Two similar scenes rendered using comparable rendering code in Java and Python .

1.3 Point Lights

1.4 Shading

1.5 Camera

1.6 Python or Java

In this section two basic ray tracing implementations using only diffuse shading without shadows in Java and Python will be compared. A scene consisting of a plane with a chess like procedural texture as well a two spheres will be set up in both languages. The rendered 600x600 pixel images are shown in figure1.1. With the result of the java code on the left and the python version on the right. The first image took 1.18s to compute while the second was ready after 16.61s. If the python code is executed faster if it is run a second time as compiled python files will be generated, but for this scene it never falls under 10s. A popular method to speed up python code is to use a the typed Cython dialect.[9] However cythons multi-threaded capabilities are limited due to pythons global interpreter lock, additionally cython compiled classes can only be accessed from compiled code, which leads to a significant extra amount of code conversion work. Therefore the remainder of the project was done exclusively in Java. Javas strong typing avoids the python overhead which slowed down the ray tracing process considerably, as the python interpreter looses a lot of time determining the type of every object.

[9]Cython: The Best of Both Worlds,Computing in Science & Engineering, S. Behnel; R. Bradshaw et al., Computing in Science & Engineering `http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5582062`

Milestone 2

Textures and acceleration

2.1 Textures

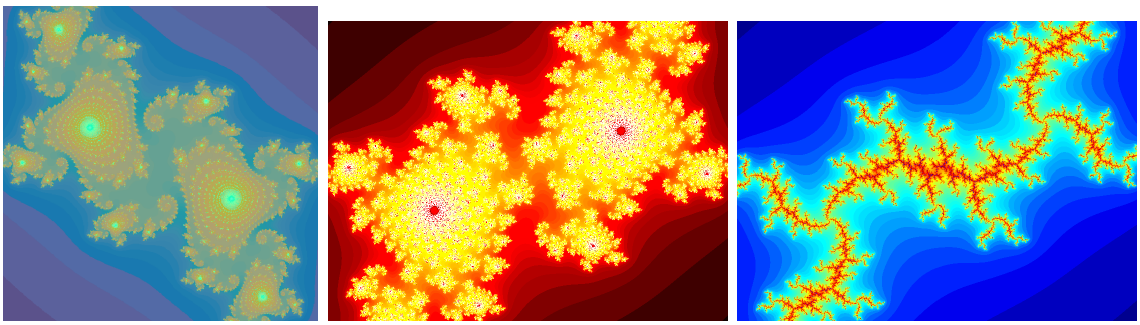


Figure 2.1: The Julia fractal shown using different seeds and colormaps.

Milestone 3

Special Effects