Imatge Sintètica - Assignment 1

Ricardo Marques

2017/2018

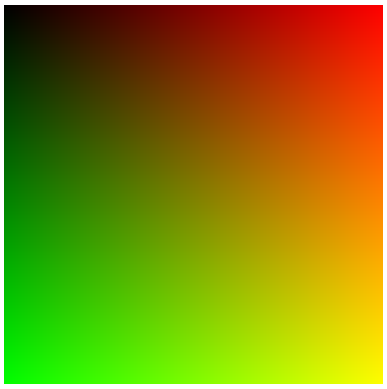

Figure 1: Visualization of each pixel's coordinates in normalized device coordinates (NDC).

1 Introduction

Go to the aula global of this course, **download** the program RTIS - Ray Tracer for "Imatge Sintètica", **open** the file VisualStudio2015.sln, **compile** it and **execute** it. RTIS is composed of a set of classes which will be the base of your first ray tracer. In this assignment, we will **only use three** of those classes: Vector3D, Matrix4x4, and Film.

- The Vector3D and Matrix4x4 classes allow you to make basic vector and matrix operations.
- The Film class allows you to manipulate and save an image.

Please take a look at the interface of these classes (defined in the header files). This will give you an idea of the main operations you can make using the provided framework.

2 Objective

This assignment consists of a set of exercises which will make you get familiar with the provided framework of RTIS. At the end of this assignment you should have gained the following competences:

- Understand and manipulate transformation matrices.
- Use the transformation matrices to transform points, vectors and normals.
- Traverse, paint and save an image using the Film class.

3 Transformations

3.1 Context

3D transformations are ubiquitous in computer graphics. In ray tracing, they are mainly used to place objects and cameras on the 3D scene, and to change coordinate system (e.g., to switch from world space to object space, or from camera space to screen space, etc.). Knowing how to construct 3D transformations and apply them to the different 3D entities of a ray tracer (such as points, vectors, normals, rays, etc.) is thus an essential skill for your project.

3.2 Your Task

3.2.1 Transformations exercise

In the main function of the file main.cpp, uncomment the function *transformationsExercise()*. Then, go to the body of *transformationsExercise()* (which you can find above the *main()*). Parts of *transformationsExercise()* are already written, while other parts are missing. Your task is to complete the missing parts so that the *transformationsExercise()* function **constructs** and **shows** the following matrices:

1. Identity (note that by default, a Matrix4x4 is initialized with the identity matrix).
2. A translation matrix $\mathbf{T}$ where $\Delta_x = 1$, $\Delta_y = -2$, $\Delta_z = 3$.
3. A scale matrix $\mathbf{S}$ where $s_x = 2$, $s_y = 1$, $s_z = -1$.
4. A rotation matrix $\mathbf{R}_x$ of $60°$ around the **x**-axis.
5. A rotation matrix of $\mathbf{R}_a$ $30°$ around the axis $(1, 1, 1)$.

Confirm that the matrices you constructed are correct by **comparing** the output generated by your function **with the file *transformationsExercise.txt***, provided for this assignment in the output folder. Once this is done, compute and show the following matrices:

6. $\mathbf{T}^\intercal$, the transposed of $\mathbf{T}$
7. $\mathbf{T}^{-1}$, the inverse of $\mathbf{T}$
8. Verify that $\mathbf{T}\,\mathbf{T}^{-1} = \mathbf{Id}$ (the identity matrix)

Using the operator '*' (multiplication) combine the matrices $\mathbf{T}$ and $\mathbf{S}$ in different orders and visualize the resulting matrix:

9. $\mathbf{T} * \mathbf{S}$

10. $\mathbf{S} * \mathbf{T}$

Note the the result is different and understand why! (if necessary make a 2D example on your notebook where you first scale and then translate a point, and then do the same in reverse order).

3.2.2 Normal transform exercise

Uncomment the function *normalTransformExercise()* and complete it so as to **replicate the example** of *transforming normals* that was seen in the Theory class. The *normalTransformExercise()* should perform the following computations:

- Declare and initialize the vector $\mathbf{v}$ and the normal $\mathbf{n}$ with the values seen in class (recall that both have type Vector3D).
- Declare and initialize a transform matrix $\mathbf{S}$ with the values seen in class.
- Compute and verify the value of $\mathbf{v}'$.
- Using the method *transformVector()* of the Matrix4x4 class, compute the result of transforming $\mathbf{n}$ directly using the transform $\mathbf{S}$ (i.e., $\mathbf{S}\,\mathbf{n}$).
- Using the function *dot()* defined in Vector3D.h, show that $(\mathbf{S}\,\mathbf{n}) \cdot \mathbf{v}' \neq 0$.
- Compute $\mathbf{n}'$ using the correct method and show that $\mathbf{n}' \cdot \mathbf{v}' = 0$.

4 Painting an Image

4.1 Context

In ray tracing (and in many computer graphics applications) the final result must be stored as an image. Therefore, the programmer must know how to manipulate the image values, i.e., its pixels values. In this exercise we will see how to set the pixels values for the whole image, one by one.

4.2 Your Task

Consider an image with resolution $(resX, resY)$ and a pixel with coordinates $p = (col, lin)$ which belongs to the same image. Then, the center of the pixel p in *normalized device coordinates* (NDC) is given by:

$$p_{NDC} = (x, y) = \left(\frac{col + 0.5}{rexX}, \frac{lin + 0.5}{resY} \right) \tag{1}$$

Complete the skeleton of the function *paintingAnImageExercise()* by using Eq. (1) to visualize the NDC of each pixel. This can be achieved by creating an image which stores the x and y coordinates of each pixel (in NDC) in its red and green canals (the blue canal remaining with value 0). The final result should look like the image shown in Fig. 1.

Attention! The method *setPixelValue()* of the film class receives first the pixel column and then the pixel line!

5 Filtering an Image

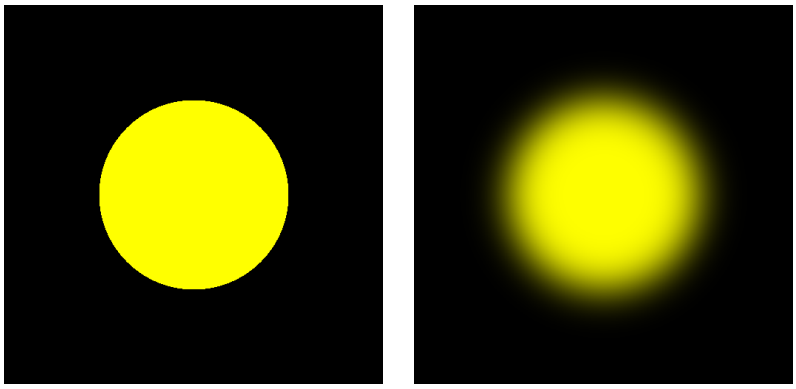

Figure 2: The blurring effect obtained when applying a filter to an image. Left: original image. Right: Blurred (filtered) image. The blur has been generated by iteratively applying a filter of size 9x9, during 100 iterations.

5.1 Context

When dealing with image synthesis it is common to resort to post processing techniques to improve the final result or to achieve some extra effects. One of these techniques is called filtering and it allows blurring the image in order to achieve an effect such as the one shown in Fig. 2. To reach this effect, the value of a pixel in the final (filtered) image is computed by making an average using the neighboring pixels. Fig. 3 shows an example

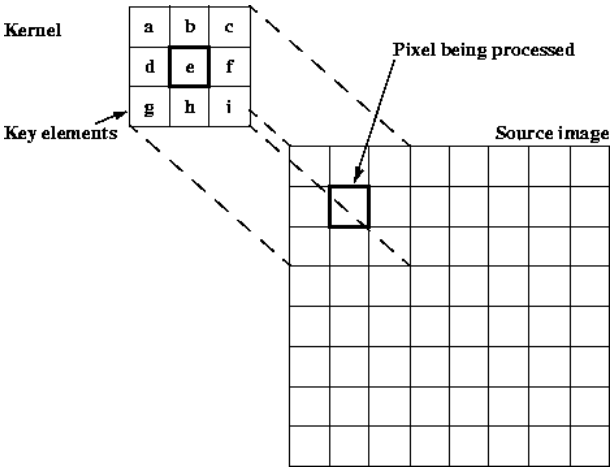

Figure 3: Example of application of a square filter of size 3x3. Image from [gm-soft.it].

5.2 Your Task

Go to Aula Global of IS and **download the skeleton** of the function *filteringAnImageExercise()*. **Copy** it to your file *main.cpp* and **call** it from the *main()* function. Note that, in its current form, this function **only** generates the original image of Fig. 2 (left). Your task is to complete this function's skeleton so as to achieve the blurring effect of Fig. 2 (right). **Test the impact** of the filter size and of the number of passes in the final result.

Extra exercise: use a Gaussian filter to attribute a weight to each of the neighboring pixels when making the average. Doing so will make nearby pixels have a larger weight in the final blurred pixel value than those which are far away. Check the impact of the filter width in the final result.