Lab 8: 3D Transformations (deel 2)

3D Computer Graphics

Introduction

No new jar file is provided for this Lab. The idea is that you keep on working on your version of the rendering framework which you obtained after finishing the exercises of Lab 7. However, it is strongly advised that you make a new fresh copy of the project of Lab 7 and rename this copy to 3DCG_Lab8. This will make it easier for you to look again at the work you did in each Lab when you study for the exam later on.

The main goal of this Lab is to add support for applying multiple transformations to multiple objects in the scene. But we will break this down in two parts. First, we will focus on applying multiple transformations to one object.

Exercise 1

a) Open the file `app1.cfg`. The first line of this file tells us that the first application renders the scene described in the `simpleScene.sdl` file.

b) Open this sdl file and change its content into:

```
background 1 1 1
light -10 10 10 1 1 1
diffuse 0 0 1
ambient 0 0 0.5
scale 3 1 1
sphere
```

c) What image do you expect to get if you would render this scene?

d) Run `App1` to check your answer.

Next, we would like to add support for applying multiple transformation to one object in the scene.

e) Change the line

```
scale 3 1 1
```

in `simpleScene.sdl` by

```
translate 0 2 0
rotate 45 0 0 1
scale 3 1 1
```

f) What kind of image would you like to get on your screen when these transformation commands are specified? Use both reasonings (as explained in the slides of this Lab) to answer this question. Make sure that you get the same result irrespective of the reasoning used. (Ask your tutor for help if necessary!)

g) Think about how our current implementation of the rendering framework deals with these transformation commands.

h) Run `App1` again to check your answer to g).

Let's now make all the necessary changes to our rendering framework to render this scene properly.

Exercise 2

First, we will add the feature of pre- and postmultiplying matrices.

a) Open the `Matrix` class and add the following method to this class.

```
public void preMult(Matrix a){
  float sum = 0;
  Matrix tmp = new Matrix(this);
  for(int r=0; r<4; r++){
    for(int c=0; c<4; c++){
      for(int k=0; k<4; k++){
        sum +=a.m[r][k]*tmp.m[k][c];
      }
      m[r][c] = sum;
    }
  }
}
```

b) The method `preMult` updates a matrix by premultiplying it with the matrix (a) given as an argument.

c) Study the implementation of this method. Is this implementation correct?

d) Add a similar method `postMult` to the `Matrix` class which updates a matrix by postmultiplying it with the matrix (a) given as an argument. Implement this method yourself.

Exercise 3

In this exercise, we will adapt the `SceneFactory` class so that it keeps track of the current composite transformation matrix T_c (and its inverse) while parsing an sdl file.

a) Open the `SceneFactory` class.

Note the instance variable `currTransfo` which we added in the previous Lab to store the one transformation which has to be applied to the only shape in the scene. We will now use this instance variable to store the current composite transformation (to be more precise, its matrix and its inverse matrix).

Have a look at the `processTransfo` method of the `SceneFactory` class. Currently, it sets the `currTransfo` variable to a new `Transfo` object if a tranformation token is encountered. Instead, we want it to "transform" the `currTransfo` variable so that the `currTransfo` variable also takes into account the new `Transfo` object.

b) Adapt the code which is executed when a transformation token is encountered by calling the — not yet implemented — method `transform` on the `currTransfo` object which gets the new `Transfo` object as an argument.

c) Implement the method `transform` in the `Transfo` class. Use the slides of this Lab as a guide. Note that this implementation should only contain two lines of code!

d) Run `App1` again. If you carried out all the previous exercises correctly, you should get the expected image of a sphere transformed by three transformations on your screen.

Note that very little code was required to add support for applying multiple transformations to one object. Let's now extend this idea to multiple objects in the scene.

In the slides of this Lab, an alternative reasoning is explained to arrive at the transformations and the order in which they have to be specified in the sdl file. This alternative reasoning does not consider how an object has to be transformed with respect to a given coordinate system but instead, considers how the coordinate system itself has to be transformed with respect to the previous coordinate system. This alternative reasoning is by far the easiest (and often the only possible) way to specify multiple transformations to multiple objects.

Note however that this alternative reasoning is only relevant for the person who has to describe the scene (by specifying the correct commands in the sdl file). In other words, no implementation changes are required to support this alternative reasoning.

Even better, our current implementation already supports applying multiple transformations to multiple objects in the scene! This is illustrated by the next exercise.

Exercise 4

a) Open the `simpleScene.sdl` file and add the following two lines at the end:

```
translate 0 -2 0
sphere
```

b) Think about the image you expect to get before running the application. Use the alternative reasoning!

c) Check your answer by running `App1`.

This confirms the fact that our rendering framework already supports multiple transformations applied to multiple objects in the scene. However, we would like to implement a mechanism which allows to save a transformation and return to a previously saved tranformation as this makes life a lot easier for the person who has to create the sdl file. As explained in the slides of this Lab, this requires

1. a stack implementation to keep track of all the saved transformations,
2. support for `push` and `pop` commands in the sdl file.

The implementation of these new features is the subject of the following two exercises.

Exercise 5

a) Create a new class `TransfoStack` in the `transfo` package. As its name already suggests, this class represents a stack of `Transfo` objects in order to keep track of all the saved transformations.

b) Use the `java.util.Stack` class to create a private instance variable `stack` which will hold the stack of `Transfo` objects.

c) Make sure you understand the difference between the three methods `peek`, `pop` and `push` of the `java.util.stack` class by studying their javadoc.

d) Create a default constructor in the `TransfoStack` class which initializes the stack with one element: the transformation which has no effect at all.

We will also implement three methods `peek`, `pop` and `push` in our `TransfoStack` class because we want similar — but slightly different — operations than the ones provided by the `java.util.Stack` class. However, make use of the methods of the `java.util.Stack` class to implement your own version of these methods!

e) Implement a `peek` method in the `TransfoStack` class which returns the transformation at the top of the stack without removing it from the stack.

f) Implement a `pop` method in the `TransfoStack` class which simply removes the transformation at the top of the stack. There is no need to return this transformation.

g) Implement a `push` method (without arguments) which adds a copy of the transformation at the top of the stack to the stack.

h) Convince yourself that your implementation of the `push` and `pop` method does exactly what you want: saving the current composite transformation and restoring the last saved composite transformation, respectively. (See the slides of this Lab for more information.)

h) Add a `transform` method to the `TransfoStack` class which transforms the top of the stack with the `Transfo` object provided as an argument to this method. This method will be used to update the current composite transformation (to be more precise, its matrix and its inverse matrix) when a new transformation command is encountered in the sdl file. Make proper use of a method you implemented before!

i) Open the `SceneFactory` class.

Currently, this class has a private instance variable `currTransfo` to keep track of the current composite transformation. This variable has to be replaced by our stack implementation which holds the current composite transformation at the top of the stack.

j) Remove the private instance variable `currTransfo` in the `SceneFactory` class and add a local variable `stack` of the type `TransfoStack` to the `createScene` method of the `SceneFactory` class. Immediately initialize this local variable by using the default constructor.

k) Give this `stack` object as a third argument to the `processTransfo` method call in the `createScene` method of the `SceneFactory` class.

l) Adapt the implementation of the `processTransfo` method so that it changes the provided `stack` object appropriately. This requires very little code changes if you know what you are doing.

m) Solve the remaining compiler error.

n) Run `App1` again to make sure that your stack implementation is bug-free.

Next, we add support for `push` and `pop` commands in the sdl file.

Exercise 6

a) Add the token `PUSH` and `POP` to `Token.java`.

b) Adapt the implementation of the `processTransfo` method of the `SceneFactory` class so that it correctly handles `push` and `pop` commands in the sdl file. Again, very little code changes are required if you make proper use of methods you implemented before.

c) Open the `simpleScene.sdl` file and change its content into:

```
background 1 1 1
light -10 10 10 1 1 1
diffuse 0 0 1
ambient 0 0 0.5
translate 0 2 0
push
rotate 45 0 0 1
scale 3 1 1
sphere
pop
rotate -45 0 0 1
scale 3 1 1
sphere
```

c) What image do you expect to get if you would render this scene?

d) Run `App1` to check your answer.

Exercise 7

Adapt the `simpleScene.sdl` file so that you get the following image when running `App1`. Note that you are not allowed to change the position of the camera! Make proper use of `push` and `pop` commands but don't overdo it!