Proseminar Visual Computing
Summer Semester 2020

Assignment 3

Guidance

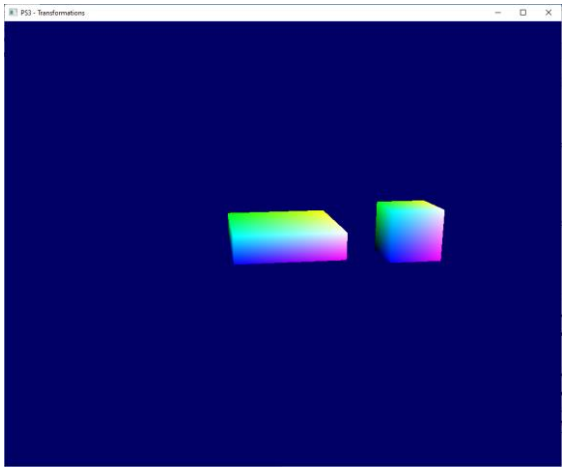

Figure 1: scene described in the example code

Global variables:

Several variables are declared at the beginning of the source code. Here are the important ones:

`nbObjects` defines the number of objects (cubes) in the scene. It is initially set to 2.
Modify it to add more objects in the scene.

Several matrices are declared, represented as static arrays of 16 numbers. For instance, `float ProjectionMatrix[16]` and `float ViewMatrix[16]` represent the projection matrix and the view matrix of the camera.

Each object in the scene is associated with a model matrix. The array `float ModelMatrix[2][16]` is a list of 2 model matrices (one for each object).

When you add more objects in the scene, you should also increase the size of this array (example: `float ModelMatrix[5][16]` if your scene contains 5 objects)

Each triangle mesh is represented by a set of buffers so it can be rendered. In this code, each objects gets a Vertex Buffer Object (VBO) which contains the 3D coordinates of its vertices, a

Color Buffer (CBO) which contains the RGB color of the vertices, and an Index Buffer (IBO) which contains the indices of the vertices so the triangles can be rendered in the appropriate order.

To handle several objects, we also declared static arrays of buffers: `GLuint VBOs[2], GLuint CBOs[2], GLuint IBOs[2]`.

When you add more objects in the scene, you should also increase the size of these arrays.

At the end, the object i corresponds to the buffers `VBOs[i], CBOs[i], IBOs[i]` and the matrix `ModelMatrix[i]`.

Functions:

The scene parameters and the objects are initialized in the function `void Initialize()`. At the beginning of this function, the cubes are built using the function `void createCubeMesh(GLuint* VBO, GLuint* IBO, GLuint* CBO)`. This function automatically builds a cube mesh and fills in the parameter buffers.

To add more cubes in the scenes, you can simply call the function createCubeMesh as many time as necessary, with the appropriate buffers as parameters.

In the function `Initialize()`, the model matrices are also set to identity.

You must initialize all the model matrices in this function.

The function `void OnIdle()` is called at each frame of the animation. This is where the model matrices are updated to modify the position of the objects. For instance, the model matrix of the first object `ModelMatrix[0]` is updated with a rotation on the Y axis, which makes the cube rotate at the center of the scene. The model matrix of the second object `ModelMatrix[1]` is the product of the same rotation with a translation, which makes the second cube rotating around the first one. The radius of the orbit corresponds to the translation.

In this function, you must update the model matrices of all the objects in your scene.

Two functions are available for interactions: `void scrollCallback(…)` and `void keyCallback(…)`. You can modify the content of these function to add interactions.

The function `scrollCallback` is automatically called when a mouse scrolling is used. The value of the scrolling offset is given by parameter `yoffset`.

The function `keyCallback` is automatically called when a keyboard button (corresponding to parameter `key`) is pressed. You can execute different operation depending on which key is pressed. In the example given, the idle rotation is activated/deactivated when the key “Q” is pressed.

Example: Modify the example code to add one more cube in the scene

1. **Global variables**

   Modify the number of objects:

```
int nbObjects = 3;
```

   Modify the size of the arrays of buffer and model matrix accordingly

```
GLuint VBOs[3];
GLuint CBOs[3];
GLuint IBOs[3];
GLuint VAOs[3];
//…
float ModelMatrix[3][16];
```

   Add a translation and scaling matrices for the new object

```
float NewTranslate[16]; /* translation matrix */
float NewScale[16];     /* scale matrix */
```

2. **Function Initialize()**

   Create a new cube

```
createCubeMesh(&VBOs[2], &IBOs[2], &CBOs[2], &VAOs[2]);
```

   Define the new scale matrix (scaling of 0.5 along the X-axis)

```
SetScaleMatrix(0.5f,1.0f,1.0f, NewScale);
```

   Define the new translation matrix (translation of 2.0 along the Y-axis)

```
SetTranslation(0.0, 3.0, 0.0, NewTranslate);
```

3. **Function OnIdle()**

   Use these transformation matrices to assemble the Model matrix of the new object.

   Multiply the RotationMatrixAnim by the translation matrix and write the result in the model matrix (Note: RotationMatrixAnim is a rotation matrix automatically updated)

```
MultiplyMatrix(RotationMatrixAnim, NewTranslate, ModelMatrix[2]);
```

   Multiply the result by the scaling matrix, and overwrite the final result in the model matrix.

```
MultiplyMatrix(ModelMatrix[2], NewScale, ModelMatrix[2]);
```

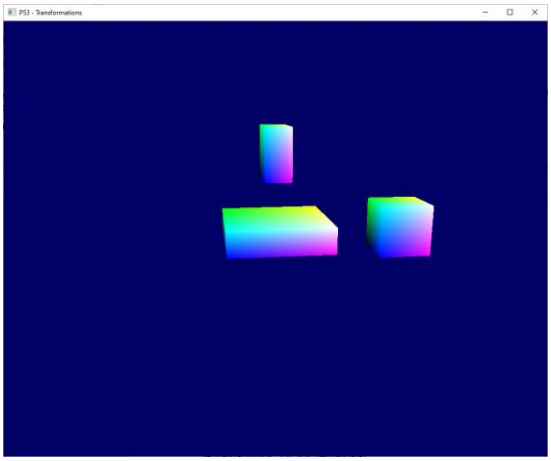

Figure 2: new scene obtained after modifications