OpenGL - Lab Session 4

Hierarchichal modeling

1 Introduction

In this session, you will work on hierarchical modeling which is a way of composing complex objects with multiple primitives.

1.1 The Frame

In the 3D world, every position is described in a coordinate frame, linked to the world, an object in the world, or part of such an object. A frame consists of a reference position P and of three orthogonal vectors x, y and z. We usually use the following frame as reference :

$$P = (0, 0, 0), \quad x = (1, 0, 0), \quad y = (0, 1, 0), \quad z = (0, 0, 1)$$

Most of the time, when dealing with a position, people do not pay attention to the frame used to describe it, assuming that the frame is the usual one. To be able to reuse objects, it is good practice to describe geometry in a frame centered on the object itself rather than absolute (world) coordinates that may be valid for only one instance of the object.

One may want to describe the equation of the sphere of radius one and of center $(0.5, 2, 55)$. Rather than computing coordinates for a sphere centered on $(0.5, 2, 55)$, a better solution will be to describe the sphere of radius 1 and of center $(0, 0, 0)$ and "place" it the following frame when drawing it:

$$P = (0.5, 2, 55), \quad x = (1, 0, 0), \quad y = (0, 1, 0), \quad z = (0, 0, 1)$$

1.2 From one Frame to Another

It is possible to mathematically describe the change from a frame to another. For this, we use **homogeneous coordinates**. Homogeneous coordinates add a fourth component to a vector to allow translations to be modeled as a matrix operator. Thus, we are able to model frame changes as multiplications of matrices.

For example, imagine we want to build the change from the reference frame to the frame described in the previous section and scale the sphere in direction y. We will model this change by the following matrix:

$$M = S_y \times T$$

Where S_y is the scaling matrix on the y component and T is the translation matrix of vector $(0.5, 2, 55)$.

Note that the order of the matrices is important as matrix multiplication is not commutative !

1.3 Frames in OpenGL

The `MODEL_VIEW` Matrix

In the Graphical pipeline, we need to switch from local coordinates (the object) to the eye coordinates (the camera). This is done in OpenGL with the `MODEL_VIEW` matrix.

This matrix is defined by all transformations from the eye to the objects local coordinates.

$$MV = M_{eye \to world} * M_{world \to object}$$

To modify this matrix, we must set OpenGL into the `MODEL_VIEW` state with the following function call:

```
glMatrixMode(GL_MODELVIEW);
```

From the Eye to the World

In OpenGL one usually starts by describing the transform between the world frame and the camera frame. Fortunately, because we are using QGLViewer, this transform is already taken care of by QGLViewer, in the form of the trackball used to move around the world frame. Keep in mind that when in the `draw()` method, QGLViewer already provides this given state of the Modelview matrix.

From the World to the Object

Given that the Modelview matrix is already loaded with world to camera transform, one can describe transforms from the world coordinate frame to a local object frame. Typically this is done by right-multiplying the current given state of the Modelview matrix by additional transforms. This is exactly the role of the following commands:

```
// Translation in the current frame
void glTranslate{fd}(TYPE x, TYPE y, TYPE z);

// Rotation in the current frame
// (x,y,z) is the axis of the rotation in the current frame
void glRotate{fd}(TYPE angle, TYPE x, TYPE y, TYPE z);

// Scaling in the current frame
void glScale{fd}(TYPE x, TYPE y, TYPE z);
```

These commands take the current Modelview state, and right-multiplies it with a 4x4 transformation matrix describing respectively a translation, rotation, or scale.

Saving a frame

Because you may have several objects in your given scene, and because all transform commands change and overwrite the current state of the Modelview matrix, you will need to save the current frame you're using to use another one instead. You typically need to use this as soon as you draw two objects in different local frames than the world frame. To do this, we can save the current `MODEL_VIEW` matrix and restore it afterwards.

```
// Save the current matrix
void glPushMatrix();

// Restore the last saved matrix
void glPopMatrix();
```

Example: if you need to draw two instances of a cube, one at a frame centered at $(10, 0, 0)$, the other at a frame centered at $(0, 0, 20)$, you first need to call `glPushMatrix()` to save the world frame. Then you translate the frame to $(10, 0, 0)$ and draw cube 1. Now you want to go to cube 2's frame but Modelview is overwritten with cube 1's local frame: a call to `glPopMatrix()` will restore the world frame previously saved. Now you can translate to $(0, 0, 20)$ and then draw cube 2.

Note that you can push and pop the Modelview matrix multiple times: this is especially useful to draw hierarchically defined objects, such as a human body.

2 Project-related assignment : model a human body

Your project will include a sea diver. Try to model him as a human body based on drawing primitives.

You may start with a simple model with a sphere for the head, cylinder for torso, arm, forearm, leg and foreleg, and just two joint levels : shoulder and elbow. You can refine the appearance of the model later. Think about what variables you need to parameterize joint angles with respect to the parent element in the skeletal hierarchy, and how to use frames, `glPushMatrix()` and `glPopMatrix()` to draw each body part.