INF3320 - OBLIGATORY EXERCISE 2

INTERACTIVE VIEWER WITH TRACKBALL

CHRISTOPHER DYKEN, REVISED BY ANDRÉ R. BRODTKORB, ATGEIRR RASMUSSEN AND BART SIWEK

Revision: 15/09/2014

We will play with transformations and quaternions in this exercise. We will make an interactive viewer, that is a program we use to view and investigate geometric models using a virtual trackball. A virtual trackball is a manipulator which mimics a regular trackball, and in this way gives the user the ability to rotate the scene in an intuitive way. Virtual trackballs are used extensively in 3D-applications.

To get an idea of how this manipulator works, try to think that the scene is contained inside a glass sphere floating in space. The center of the sphere is fixed. The user can grab the sphere at one point and drag this point in a direction. This produces a rotation around and axis orthogonal to this direction. Usually, one also has the ability to move around (translation) and zoom in and out. However, we will just look at rotation in this exercise, however, creating a full-featured virtual trackball is a small challenge to inquisitive minds.

THE APPLICATION

The application should show a cube, using perspective view, where each vertex is colored in a different color. One should inspect the cube from different angles using the mouse, i.e., it should be manipulated by a trackball manipulator.

DETAILS AND HINTS

The GLM library contains a `glm/gtc/quaternion.hpp` file which houses an implementation of quaternions including quaternion multiplication and creating an rotation matrix from a quaternion (`mat4_cast`). The files `ColoredCube.hpp` and `ColoredCube.cpp` contain code that renders a cube with different colours on vertex.

Converting screen coordinates to points on the unit sphere. First, we scale the screen coordinates to the square $[-\frac{1}{2}, \frac{1}{2}] \times [-\frac{1}{2}, \frac{1}{2}]$. We let (x', y') be the scaled counterparts of (x, y),

$$(x', y') = \left(\frac{x}{w} - \frac{1}{2}, \frac{1}{2} - \frac{y}{h}\right),$$

where w is the width of the window and h is the height. A tiny detail: We switch the sign of the y-coordinate. The reason is that the y axis points downwards in screen coordinates, while it points upwards in the coordinate system of OpenGL.

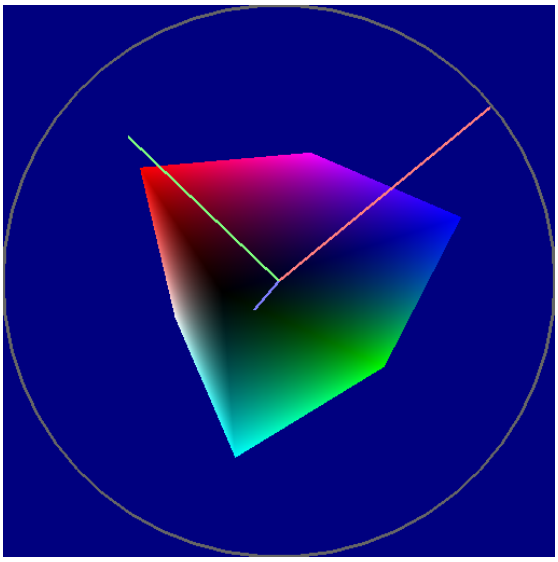

Figure 1: The sphere of the trackball is outlined in grey, the vector corresponding to the mouse click is green, the vector corresponding to the current mouse position is blue, and the axis of rotation is red.

Then, we apply a parallel projection of this square onto a sphere (actually it is an ellipsiod unless the window is a square) centered in $[0, 0, 0]$ with radius $\frac{1}{2}$. We define $r = \sqrt{x'x' + y'y'}$, the distance from the point to the center of the screen.

If $r < 1/2$ then the point hits *on* the sphere, and the vector corresponding to the point is given by

$$(2x', 2y', \sqrt{1 - 4r^2}).$$

However, if $r \geq 1/2$ then the point hits *outside* the sphere, and we choose to use the closest point on the sphere instead. The corresponding vector is then

$$(x'/r, y'/r, 0).$$

This vector is on the unit sphere, and this, the length of this vector is 1.

Finding the axis and angle of rotation The shortest path between two points on a sphere is called *the great circle*. We let the rotation that takes the first point to the second point along the great circle be our rotation. It can be found in the following manner.

Let $\mathbf{v}_1$ and $\mathbf{v}_2$ be the vectors on the unit sphere corresponding to the position where the mouse button was clicked and the current mouse position respectively.

The axis of rotation is defined by the normalized cross product of these two vectors,

$$\mathbf{a} = \frac{\mathbf{v}_1 \times \mathbf{v}_2}{\|\mathbf{v}_1 \times \mathbf{v}_2\|}$$

and the angle of the rotation is defined by

$$\theta = \cos^{-1}\left(\mathbf{v}_1 \cdot \mathbf{v}_2\right).$$

Note that, when $\mathbf{v}_1$ and $\mathbf{v}_2$ are equal, the cross product is the zero vector. This must be handled somehow.

Design suggestions. The exercise can be solved both using unit quaternions or with concatenated rotation matrices, however, we recommend using unit quaternions. We recommend using a class along the lines of

```
class SimpleViewer {
 public:
  SimpleViewer();

  void setWindowSize(int w, int h);

  void rotationBegin(int x, int y);
  void resetState(int x, int y);
  void motion(int x, int y);

  glm::mat4x4 getProjectionMatrix();
  glm::mat4x4 getMovelViewMatrix();

 private:
  glm::vec2 getNormalizedCoords(int x, int y);
  glm::vec3 getPointOnUnitSphere(GfxMath::Vec2f p);

  enum { NONE, ROTATING } m_state;
  int   m_window_width;  //< Width of window
  int   m_window_height; //< Height of window
  float m_window_aspect; //< Aspect ratio of window.

  glm::fquat m_camera_orientation;     ///< Current orientation
  glm::fquat m_rotation_orientation_i; ///< Orientation when mouse was pressed
  glm::vec3  m_rotation_mousepos_i;    ///< Mouseposition when mousebutton was pressed.
};
```

with the following functions:

- `rotationBegin()` copies `m_camera_orientation` into `m_rotation_orientation_i` and sets `m_state` to `ROTATING`.
- `resetState()` simply sets `m_state` to `NONE`.
- `motion()` if `m_state` is `ROTATING`, then do the following:

  Define `m_rotation_mousepos_c` as the vector on the unit sphere corresponding to the current mouse position.

  Define a temporary quaternion `t` describing the great circle rotation defined by `m_rotation_mousepos_i` and `m_rotation_mousepos_c`.

Set `m_camera_orientation` to be the product of `t` and `m_rotation_orientation_i`.

- `getProjectionMatrix()` computes a suitable projection matrix using the `glm::frustum` function and returns it.
- `getModelViewMatrix()` does the following:

  Start with the `glm::mat4` which corresponds to identity. Apply a translation to move the origin into the center of the view frustum.

  Then, apply the current rotation by multiplying the current transformation matrix by a result of using the `glm::mat4_cast` with the current rotation quaternion.

The `SimpleViewer` is used by a `glut`-application in the following manner,

```
ColoredCube  cube;
SimpleViewer viewer;

void myMouseFunc(int b, int s, int x, int y) {
    if(s==GLUT_DOWN && b == GLUT_LEFT_BUTTON)
        viewer.rotationBegin(x, y);
    else
        viewer.resetState(x,y);
    glutPostRedisplay();
}

void myMotionFunc(int x, int y) {
    viewer.motion(x, y);
    glutPostRedisplay();
}

void myReshape(int w, int h) {
    glViewport(0, 0, (GLsizei)w, (GLsizei)h);
    viewer.setWindowSize(w,h);
}
```

HANDING IN THE EXERCISE

The assignment is individual and everyone shall create their own program. If you choose to use code or derivations of code that is not your own, its source and author shall be explicitly cited. We will normally accept some degree of unoriginal work, but we may require that parts of the unoriginal code must be rewritten.

We reserve the right to do an oral examination of the student for each submitted exercise. The failing or passing of this examination will decide whether you pass this exercise, and this decision is final.

The program should preferably be a C++ program, and the visualisation shall be done using OpenGL. The choice of windowing toolkit is the student's. Document your efforts by commenting the code.

All source code as well as a Makefile shall be included. The Makefile should have at least a target to build the program as well as a clean-target that deletes all object-files as well as the executable. Assume that your user name is `foo` (exchange "foo" with your user name). Put all files in a directory called `foo-2`.
Handing in the assignment is performed by creating a private repository at either GitHub (sign up as student for five private repositories on `https://github.com/edu`) or Bitbucket, and then sending an email to `mariuek@ifi.uio.no` with the clone URL and a short description detailing the files involved in your hand-in. Make sure the program compiles and runs cleanly on the computers at IFI, and that it fulfils the requirements outlined in the evaluation guidelines.

Good luck!