

CS 175: Assignment 3

Quaternions and Arcball

Due:

Tuesday, October 13th, 11:59pm

Assignment Objectives

In this project you will implement the code necessary to apply rigid body transformations decomposed into a translation and a rotation (quaternion), as well as the Arcball interface for rotations. It is based off of the previous assignment and builds this functionality on top of it. You can use as base your solution for assignment 2, or the solution code provided for it.

Task 1: Rigid Body Transformation

Your first task is to define the Rigid Body Transformation structure, and implement the necessary manipulations of it. The rigid body transformation has two variables, a translation vector and a rotation quaternion. It represents the composition of the translation and the rotation. Pay attention to the order, translation on the left, rotation on the right, TR . You need to implement inversion, multiplication, conversion to matrix form (that you can later feed to OpenGL), conversion from a translation vector, conversion from a quaternion, and multiplication with a vector. Except for conversion to matrix form, all remaining operations should be implemented without converting the quaternion to matrix form. We will provide you with a Quaternion data type. The code below shows the overall structure of the Rbt type. The code will be labeled with a TODO comment in the parts that you must implement.

Example prototypes:

```
class Rbt
{
    Vector3 translation;    // translation
    Quaternion rotation;   // quaternion rotation

public:
    Rbt();
    Rbt(const Vector3& t, const Quaternion& r);
    Rbt(const Vector3& t);
    Rbt(const Quaternion& r);

    Matrix4 getMatrix() const;
    Rbt getInverse() const;
    Vector3 operator * (const Vector3 &a) const;
    Rbt operator * (const Rbt& a) const;
};
```

You need to alter your assignment 2 solution (or solution set of the course) so that it stores the position and orientation of cubes and sky camera as a Rigid Body Transformation structure, instead of a Matrix4.

At this point, you should not be using matrix-matrix multiplication, or matrix-coordinate multiplication for manipulating the sky camera and cube coordinate systems. These coordinate systems should now be

represented with an Rbt. Instead you will be using the equivalent rigid body transformation functions. During rendering you will convert rigid body transformation into a matrix, and then load these into the vertex shader appropriately as matrix data.

Task 2: Arcball

Your second task is to implement the arcball interface. The arcball interface will only come into play if:

1. You are manipulating the sky camera with respect to the world-sky coordinate system
2. You are manipulating a cube, and it is not with respect to itself.

In case 1, center the arcball about the world's origin. In case 2, center the arcball about the cube's center. In each case, you should also draw a sphere (wireframe or translucent) to view the arcball.

We provide some helper code. When manipulating an object using arcball, you know the center and radius in 3D of the arcball in world coordinates. However, the user clicks somewhere on the screen, and you must use these screen coordinates (measured in pixel units) to figure out where exactly on the arcball the user has clicked.

To facilitate this, the helper code figures out for you where exactly on the screen your 3D arcball projects. You give it the center and radius of the arcball (in eye coordinates), the current camera-projection matrix you are using, as well as the screen resolution, and it will compute for you the center and radius of the projected arcball, in screen (pixel) coordinates that are compatible with the screen coordinates that the mouse function receives.