Programming Assignment 2: User interaction
Computer Graphics, Fall 2016

Submission deadline: Thursday, October 20th, 11:59

You will have to submit your solution for this exercise before Thursday, October 20th, 11:59 to ILIAS. Additionally, you will have to present your solution to one of the assistants. Please register for a time slot on ILIAS. Please create separate Eclipse projects or packages for the individual tasks in order to be able to demonstrate them separately. You may for example copy and extend the project "simple." Use the additional resources that we provide in the file *material.zip*.

1 Frustum (1 Point)

1.1 Frustum

Extend the class *jrtr.Frustum* such that a projection matrix can be constructed out of the 4 parameters *Near Plane, Far Plane, Aspect Ratio* and *Vertical Field of View*. Use the matrix formula that has been discussed in the lecture. You must not use the *glu* library or other libraries to compute the camera and projection matrix.

1.2 Test

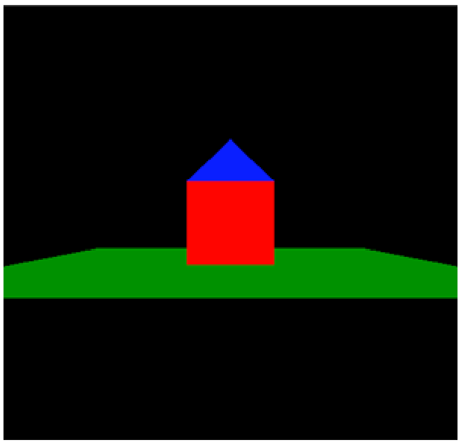
Use the test scene *house.txt* that we provide on ILIAS and render two images using the two parameter sets shown in the figure below.

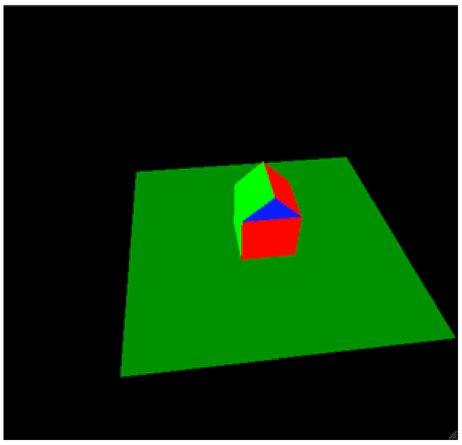
2 Virtual Trackball (3 Points)

Implement a virtual trackball that you can use to rotate an object with the mouse. Your solution should translate mouse movements with clicked mouse button (dragging) into a rotation matrix which can then be used to transform the scene. Rotations around all three coordinate axes should be possible. The figure below shows how to extract a rotation axis and a rotation angle out of a mouse movement. The symbols m_0 and m_1 denote two consecutive 2D mouse positions. These positions define two 3D points v and w on a "virtual sphere" that fills the rendering window. Use the cross product $a = v \times w$ as a rotation axis and the angle between v and w as a rotation angle.

Parameters for image 1	
Aspect Ratio	1
Vertical Field of View	60 Grad
Near, far clip Planes	1, 100
Center of Projection	0,0,40
Look-at-Point	0,0,0
Up-Vector	0,1,0

Parameters for image 2	
Aspect Ratio	1
Vertical Field of View	60 Grad
Near, far clip Planes	1, 100
Center of Projection	-10,40,40
Look-at-Point	-5,0,0
Up-Vector	0,1,0


Output for image 1

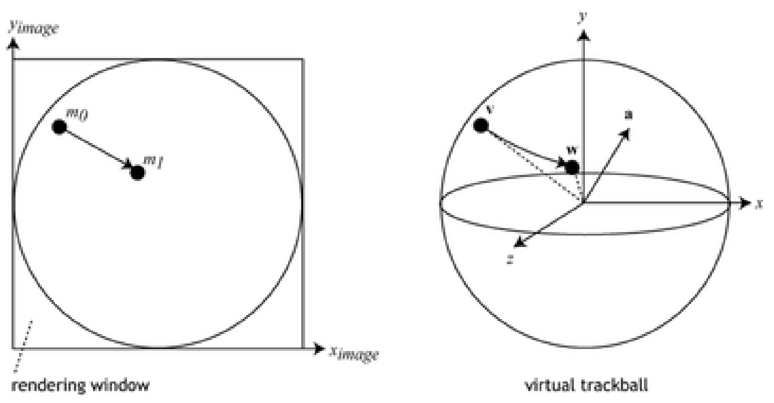

Output for image 2

Horizontal movements in the center of the window should lead to a rotation around the y-axis. Vertical movements in the center of the window should lead to a rotation around the x-axis. Movements at the boundary of the window (horizontal and vertical) should lead to a rotation around the z-axis. On ILIAS you'll find a detailed description of a virtual trackball implementation. Do not forget to handle the following special cases:

- The mouse position lies outside the virtual trackball.
- The rendering window is not quadratic.

2.1 Test

Test your implementation with triangle meshes that are read from files. You can use the class *ObjReader* which can be found in the *jrtr* project. The object reader reads the *.obj* file format, a simple text-based file format to store polygon meshes. What is basically stored is a list of vertices (rows starting with v, one vertex per row) and a list of indices of the vertices of all polygons (rows starting with f, one polygon per row). Additionally, normals (rows starting with vn) and texture coordinates (rows starting with vt) can be stored. You can find more details concerning the *.obj* format on Wikipedia and in the obj format specification. The subfolder *obj* of the provided basecode contains some test files. It is recommended to test the trackball with the file *Teapot.obj*.



Visualization of the virtual trackball.

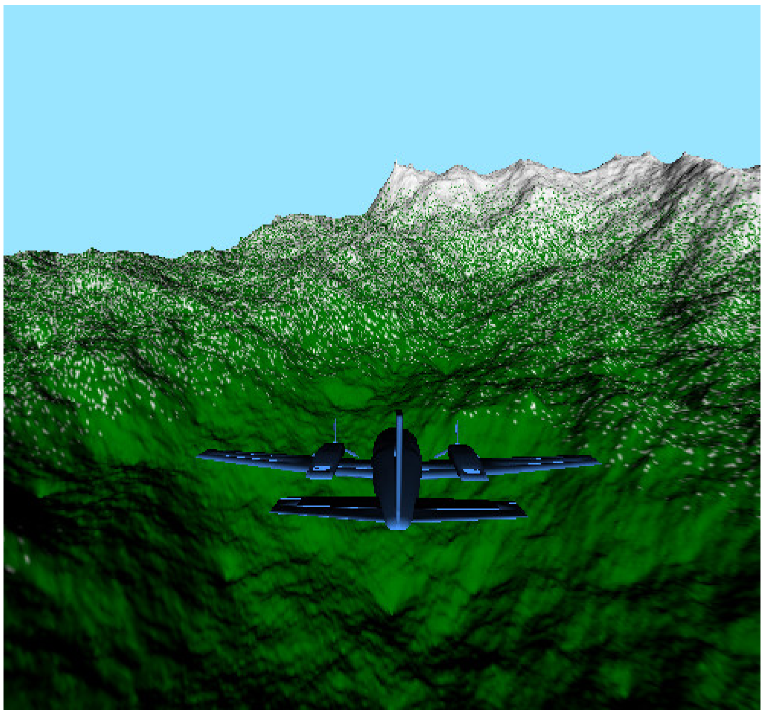
3 Fractal Landscape (3 Points)

Implement an algorithm to generate a fractal landscape. Follow the description from the exercise session. A detailed explanation can also be found here. Your method should also generate normal vectors by computing the cross product of neighboring triangle edges. Choose the coordinates such that the xy-plane corresponds to the ground plane (floor) and z corresponds to the height. Develop an easy method to assign colors to the vertices depending on their height.

4 Interactive camera movement (3 Points)

Implement a method to move the camera interactively using a combination of keyboard and mouse inputs. Keyboard inputs should determine the translation of the camera and mouse inputs should determine the rotation. Use for example the WASD keys to move forward, left, backward and right. If a key is pressed, the camera should move towards the corresponding direction. The mouse determines the rotation of the camera. There are different possibilities to choose the rotation axes for an intuitive navigation. We assume that the scene consists of a world where the xy-plane corresponds to the floor and z points upwards. We propose that vertical mouse movements correspond to a rotation of the camera around the x-axis of the camera coordinate system. Horizontal mouse movements could correspond to a rotation around the z-axis of the world coordinate system. Use the navigation with mouse and keyboard to "fly" through your landscape from subtask 3.

Place an airplane in front of the camera (*airplane.obj*[1]) and transform the airplane in such a way that you can fly the airplane through the scene using mouse and keyboard (see Figure below). The camera should follow the movement of the airplane, i.e. the camera should not move relative to the airplane.



Interactive Camere Movement

[1]The obj file for the airplane can be found on github.