Umeå University
Department of Computing Science
Computer Graphics and Visualization

Pedher Johansson
pedher@cs.umu.se

Assignment 3

Walking in the air

Objectives

This assignment calls for you to write a limited 3D studio software. This should make you familiar with 3D transformations and shading and how this is implemented in OpenGL. Further, parametric curves and 3D models are covered.

Requirements

This assignment must be done individually.

The user of your software should be able to do the following.

- Load 3D objects stored as OFF model files.
- Loaded objects should be normalized to fit in a unit cube. This may can be done by updating vertex data, merge with the view matrix (MV = V·T) or by using a model matrix (T).
- Move the camera freely using keyboard (or mouse).
- Implement Phong (or Blinn-Phong) lighting model in the vertex shader.
- Change default materials of objects, properties of the light source and direction of your normals by a GUI.
- Activate smooth movement of the camera using parametric curves.
- Get feedback of current camera position and look-at-point in the terminal.

Any extension or alternation of the functionality is encouraged as long as it does not limit the usability or intention of the assignment.

Bonus

The following extensions gives bonus points on the first written exam. Bonus points are only given if the basic functionality works and can only be used to get a hight grade (not to pass) the exam.

- Several light sources, including one that moves (2 points)
- Control the camera rotation and field of view with the mouse (1 point)
- Phong shading (1 point)
- Mirror effect (2 points). Add a flat surface that mirrors the object.

Moving the camera

We should be able to move the camera using the keyboard and get some feedback of its position. The latter can be done in the console.

- When moving the camera's position left (A), right (D), forward (W), and backward (S), up (Z) and down (X), the look-at-point should follow accordingly.
- When rotating the camera up (Up/Mouse) or down (Down/Mouse), the look-at-point should be rotated around the camera's x-axis. When rotating up (Left/Mouse) and (Right/Mouse), the look-at-point should be rotated around the camera's y-axis.
- Each time the camera moves, a new look-at-point should be computed such that the look-at-point always is at certain distance from the camera. Notice the following.

$$V = M_{wc} T(-p_0)$$
$$V^{-1} = T(-p_0)^{-1} M_{wc}^{-1} = T(p_0) M_{wc}^T$$

$$p^{(camera)} = V p^{(world)}$$
$$p^{(world)} = V^{-1} p^{(camera)}$$

$$V^{-1} p_0^{(camera)} = p^{(world)}$$
$$\Rightarrow V^{-1}[0\,0\,0\,1]^T = p_0^{(world)}$$
$$\Rightarrow V^{-1}[0\,0\,-d\,1]^T = p_{ref}^{(world)}$$
$$\Rightarrow V^{-1}[0\,1\,0\,0]^T = V^{(world)}$$

That is, an update of the camera position, reference point, or up-vector is easy to express in camera coordinates, which can be multiplied with V^{-1} to get them back to world coordinates. For example, a move of the camera along it's positive y (up) can be expressed as follows, resulting in an updated camera position and reference point, that can be used to update V.

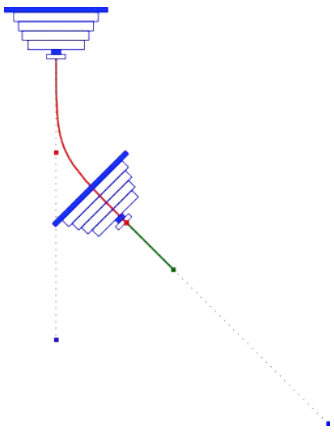
$$p'^{(world)}_0 \leftarrow V^{-1}[0\,d\,0\,1]^T$$
$$p'^{(world)}_{ref} \leftarrow V^{-1}[0\,d\,-1\,1]^T$$
$$Up' \leftarrow Up$$
$$V' \leftarrow view(p'_0, p'_{ref}, Up')$$

Smooth movement of the camera

By activating smooth movement of the camera, the controls starts acting differently. For example, then

- (A/Mouse) starts to turn the camera to the left
- (D/Mouse) starts to turn the camera to the right
- (W/SHIFT) increases the speed of the camera
- (S/Ctrl) decreases the speed (not negative)

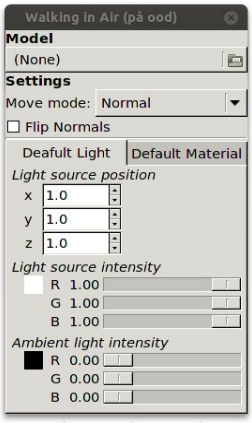
Speed should be units/time and not units/frame rate and be reasonable corresponding to the size of world coordinate system.



Paths of the camera should be calculated with a *quadratic* beziér curve, and line segments. The camera travels by default along the line-of-sight. Each time we gets a turn signal, a new beziér curve should be computed. The size of the curve should be such that the camera travels along the curve and reaches the end approximately at the same time independent of speed. As the camera reaches the end of the curve, it should continue in the tangent direction.

User interface

Your GUI should allow easy adjustments of the parameters for shading (material and light properties), rendering mode, back face mode and move mode. Also a possibility to flip normals ans to show them. A given GUI is available for you to use and it looks as follows. Code for this is to be found in the file collection.



Flip normals

The OFF-files do not specifies the order of the vertices. That results in both counter-clockwise and clockwise ordered vertices in our objects. Fortunately the order is the same in each OFF-files. The direction of the normal is however important, and if calculating the normal in the same way, results in some objects with normals pointing like a Sea Urchin (or hedgehog), and some objects with normals pointing like a mediaeval Iron maiden. For the latter objects we like to change the direction of the normals.

Instructions

This is an individual assignment and must not be implemented in pair or groups. Post your finished solution as an zip-file in Cambro no later than the date given in the course syllabus. The code should follow good programming practice and be compliant and executable on CS Linux environment. No report is required.

Some tips

- During development, auto-load an object (e.g., the cone), when the program starts.
- Set some default values for the light and material so you don't have to do that every time.
- Don't move the camera too fast, and have a short cut to reset the camera.