Augmented Reality

Question 1  Error Models in Estimation Problems

For estimation of a homography, different errors with different properties can be considered. The error triangle shows the three different errors that occur in estimation problems.

Figure 1: Error triangle

(1) Explain the estimation problem for homography estimation.

(2) Referring to the error triangle, what error is minimized in estimation? Explain the different error models that can be used to describe this error.

(3) Explain the difference of algebraic error and geometric error (in one image). What is the reason for this difference?

(4) When estimating a 2D homography, we usually have measurement errors in both images, not only in one. Considering this fact, explain symmetric transfer and reprojection error and their difference.

Question 2  Changing Camera Models

Up to now, only one camera model was discussed, the camera model based on focal length. This is the one used by the Computer Vision society to calculate with and describe cameras. However, there are more camera models that are used for different fields of applications. In Computer Graphics one will eventually be confronted with a camera model based on viewing angles (see figure 2). Here, the camera is described via a height and a width angle spanning the image plane and the aspect ratio determining the camera viewing width and height and a far and near clipping plane that define the spatial area where objects are actually captured and projected onto the image plane. The camera center is called viewpoint and it’s pose (position and orientation) is also described, i.e. the extrinsic parameters are the same in both camera models. The model based on viewing angles is for instance used by OPEN
INVENTOR, a library used to create interactive 3D graphics applications to describe virtual cameras of a scene. If a camera has been described by a projection matrix and shall be used to view a scene in OPEN INVENTOR, naturally, the camera models have to be changed.

![Figure 2: Camera Model based on Viewing Angle, from the Inventor Mentor book](image)

(1) A good assumption to make for implementations of this camera model is to make the line through viewpoint and image center perpendicular to the image plane. Is the model then still unique?

(2) From now on, we assume the line from viewpoint to image center perpendicular to the image plane. Regarding units in space and on the image plane, what adjustments have to be made in order to change camera models?

(3) Develop a formula that expresses the height angle with the parameters of the focal length camera model.

**Question 3 Getting to Know OpenGL and GLFW**

Now that our tracker is working, we will shift our focus to rendering. For cross-platform 3D graphics, the Open Graphics Library (OpenGL), originally developed by SGI, is widely used. As OpenGL doesn’t contain code for window handling, user interaction etc., GLFW will be used, too. Please review the help-slides provided with the assignment to guide you along the way. Also, the provided main.cpp already implements most of the start-up code that you need.

(1) Start a new program which should use GLFW to open a rendering window. Use the functions `glfwInit`, `glfwCreateWindow`, and `glfwMakeContextCurrent`. Your window should support double buffering (by default), RGBA colors, and a depth buffer. In your main function, create a rendering loop with `glfwWindowShouldClose`.

(2) Your new rendering window will so far not contain anything. Implement your own rendering function which clears the window using `glClear`, and call it in the rendering loop. As you should be using a double-buffered window, call `glfwSwapBuffers` at the end of the loop to exchange the front and back buffers. After swapping the buffer make sure to call `glfwPollEvents` to processes all pending events;
(3) In order to allow resizing the window, you also need to register a resize callback via `glfwSetFramebufferSizeCallback`. This callback is passed the new window dimensions by GLFW. Call `glViewport` to set the rendering area to the new size.

(4) For three-dimensional rendering, you need to set two matrices, the modelview and the projection matrix. The active matrix is set by `glMatrixMode`. Clear the modelview matrix at the beginning of your display callback by using the `glLoadIdentity()` function and set the projection matrix using `gluPerspective(30, ((double)width / (double)height), 0.01, 100)` in your resize callback.

(5) Finally, it's time to render something. Keep in mind that the camera is by default located at the origin and looks down the negative z axis, so use `glTranslatef` to move your world in negative z direction and, at long last, draw a solid sphere using `drawSphere` we provide.

**Question 4 OpenGL Primitives**

Now that we know how to draw spheres, we will now build a snowman. Use `drawSphere` commands to build the body and `drawCone` for the nose. Remember to change the modelview matrix between calls - the transformations affecting a primitive are all transformations called before, in reverse order (when thinking in terms of a single world coordinate system).

*Note:* it may be useful to save and restore your current matrix with `glPushMatrix` and `glPopMatrix`.

**Question 5 OpenGL Lighting**

To make the snowman look more realistic, add some lighting to the scene. Use `glLightfv` to set the three lighting parameters `GL_POSITION`, `GL_DIFFUSE` and `GL_AMBIENT` for the first light source. Don’t forget to enable the light source and lighting in general with `glEnable`.

**Question 6 GLFW: Animation**

Finally, let’s make the snowman spin. In order to increase the rotation angle periodically, use `glfwGetTime()` to get a timestamp and convert it into dynamic angle.

Example of how it should look in the end:

![Image of a snowman rendered with OpenGL](image-url)