Documentation of transform.cpp

The file is a development sandbox for estimating head pose, and for testing related computations. All the necessary data structures etc. are stuffed into that single file, so it can be quite difficult to read without an advanced code editor. This document provides a somewhat higher-level description.

Formulation

Let us assume that head is a rigid and perfectly diffuse surface in uniform light. Basically, all of these assumptions are wrong, but the algorithm may work anyway.

We are given two pictures of a human head: one of them will be regarded as *reference* and the other as *view*. We wish to calculate the 3D congruence, i.e., rotation and translation, that relates them to each other. To make the task easier, we also assume to have reliable depth information about each pixel of the reference image.

A pixel p in the reference view should get transformed to position $q = CRC^{-1}p$, where C is a known 4×4 camera projection matrix and R is a 4×4 congruence matrix to be computed. In order for the formula to make sense, p is 4-dimensional: 2D pixel with depth information, expressed in homogeneous coordinates.

The energy functional to be minimized is just the sum of error in each pixel: $E = \sum_p \psi(\text{view}(q) - \text{reference}(p))$, where ψ is an arbitrary non-decreasing smooth function.

Update step

The algorithm iterates in the direction of $-\frac{\partial E}{\partial R}$, in the hope that the global minimum lies there. To make the convergence somewhat faster, a momentum is used – but that does not matter since the optimization algorithm needs to be replaced by a smarter one anyway.

The formulation ∂R actually encompasses two different transformations: translation and rotation. Derivatives wrt. translation are easy to interpret since translation in each coordinate corresponds to an element of R and there are no constraints on the values.

Derivative wrt. rotation is somewhat more difficult. Let us consider a 4×4 rotation matrix ΔR, that is, an orthonormal matrix with 1 in the

bottom-right corner. Ignoring translation for the moment, we can formulate $R = \Delta R \cdot R_{\text{old}}$. Rotation matrices for small angles are almost skew-symmetric, so in our case, we can describe ΔR by its three elements above the main diagonal, excluding the last column.

The transformed pixel position $q = C \cdot R \cdot C^{-1} \cdot p$ can now be stated instead as $q = C \cdot \Delta R \cdot R_{\text{old}} \cdot C^{-1} \cdot p$. The actual vector $\frac{\partial E}{\partial \Delta R}$ is computed by a thorough application of the chain rule. To ensure that R remains a congruence, its rotational part is orthonormalized using singular value decomposition after each step.

Scale layers

In order to make the error function E a bit smooth, we can average it over several smoothed and downscaled versions of the image. The rest of the algorithm remains intact. Contribution of each scale layer to the final $\frac{\partial E}{\partial R}$ is divided by the image area, so that each layer contributes with exactly the same weight.

Issues

As already stated, the initial assumptions are wrong. In classical computer graphics, sophisticated subsurface shaders are used to mimic human skin, and they are used for a reason. Unless the user is wearing glasses, actually no part of his head is rigid; if the user starts to make faces, any fitting algorithm is lost.

Most importantly, the lighting is never even in real scenarios. The error function used so far cannot handle that bias well and the optimization will often diverge from the correct solution.