Computer Graphics SoSe16
Exercise 2

Tobias Knopp & Martin Hofmann
Institut for Biomedical Imaging

Next exercises:
02.05.2016, 14:15-15:45
09.05.2016, 14:45-15:45

The aim of the first few exercises is to write a basic software rendering module for the visualization of 3D data using the scientific programming language `julia`.

1. In the last exercise basic transformations were introduced. These transformations say T_1 and T_2 can be composed to form more complex transformations. I.e. any vector $\vec{v}$ is first mapped to $\vec{v} \mapsto T_1\vec{v}$ and the result is then transformed by T_2 to $T_2T_1\vec{v}$.

   If many vectors are transformed it is more efficient to compose the transformations using matrix multiplication to calculate the effective transformation $T = T_2T_1$ and map $\vec{v} \mapsto T\vec{v}$.

   Your task is to implement `*(M1::Mat4f,M2::Mat4f)` and `*(T1::Transformation ,T2::Transformation)`.

   Hint: In Julia `import Base: *` has to be called prior to defining new functions that implement * for new types.

2. Write a function `rotateObject` that stepwise rotates the `houseOfSantaClaus` around the origin in the xy-plane using the appropriate basic rotation. Use a for loop and render the rotated Object for 60 equidistant angles using the `sleep` function to adjust the rendering speed.

   In a second step adjust the function to rotate the `houseOfSantaClaus` around the point $(1, 1, 0)$ by composing appropriate translations and rotations.

3. Composition of the basic rotation can be used to form any rotation. The Swiss mathematician Leonhard Euler found that three basic rotations are sufficient to perform any possible rotation. One of these so called Euler transformations is given by

   $$E(\alpha, \beta, \delta) = R_z(\delta)R_x(\beta)R_y(\alpha), \tag{1}$$

   were R_x, R_y and R_z are the basic rotations around standard axis. The corresponding angles α, β and δ are called Euler angles.

   Use the definition given in the last sheet to calculate an explicit expression for (1) and implement a function `euler(alpha,beta,delta)` which returns this more complex rotation as `Transformation`. Check if your implementation is correct by comparing `euler(alpha,beta,delta)` and `rotz(delta)*rotx(beta)*roty(alpha)` for random Euler angles.

Rotate and render the `houseOfSantaClaus` by $\alpha = \beta = \delta = \pi/4$.

4. In computer graphics one usually has more than one coordinate system. To be able to change the coordinate system forth and back a method to compute the inverse transformation is necessary.

   Implement the functions `inv(M::Mat4f)` and `inv(T::Transformation)`. You may use the build in `inv` function which operates on 2 dimensional arrays.

   Hint: In Julia `import Base: inv` has to be called prior to the new definitions.

5. Often it is convenient to describe Objects in their own local coordinate system. As an example consider an orthographic camera. Such a camera can in principle be described as an plane array of sensor pixels, where each pixel is lit up by light which enters the pixel parallel to the normal of the array.

   In this case it is very convenient to describe the pixel locations in a local coordinate system with axis $\vec{e}_u$, $\vec{e}_v$ and $\vec{e}_w$, where the origin coincides with the center of the array and the pixel array covers the area $-1 < u, v < 1$ and $w = 1$. In these systems the viewing direction is always in $-\vec{e}_w$ and the viewing distance is 2. Hence in this coordinate system it is always the canonical view volume with minimum corner $(-1, -1, -1)$ and maximum corner $(1, 1, 1)$ which is imaged by the pixel array.

   Usually the world coordinate system, which has the standard basis as axis and the camera coordinate system will not be aligned. In this case a translation $\vec{r}_c$ of the center of the pixel array and the normalized axis directions $\vec{e}_u$, $\vec{e}_v$ and $\vec{e}_w$ fully describe the camera coordinate system. Given these informations, which transformation (homogeneous space) $T : \mathbb{R}^4 \to \mathbb{R}^4$ transforms camera coordinates $(u, v, w, 1)$ into world coordinates $(x, y, z, 1)$ and vice versa?

   In computer graphics the camera is usually described by the position of the center $\vec{r}_c \in \mathbb{R}^3$ of the pixel array as well as the viewing and upwards direction $\vec{r}_v \in \mathbb{R}^3$ and $\vec{r}_u \in \mathbb{R}^3$ of the screen as these uniquely determine T.

   Generate T from $\vec{r}_c$, $\vec{r}_v$ and $\vec{r}_u$. Write a new `type OrthoCamera` which has `camToWorld` and its inverse `worldToCam` as fields and can be constructed from $\vec{r}_c$, r_v and r_u which are given as `Vector{Float32}`.

   Hint: The cross product can be used to complete two orthonormal vectors to a basis in $\mathbb{R}^3$.

6. Write a function `render(object::Object,camera::OrthoCamera;figNum=1)` that allows to render the same scene from different camera positions.

   Use this function to render the `houseofSantaClaus` from diffent perspectives.

```
# scale down the houseOfSantaClaus to fit into the canonical view volume
scaledHouseOfSantaClaus = scaling(0.5,0.5,0.5)*houseOfSantaClaus

# canonical view direction
camera = OrthoCamera(Float32[0,0,1],Float32[0,0,-1],Float32[0,1,0])
render(scaledHouseOfSantaClaus,camera;figNum=4)

# screen moved backwards 9 unit length
```

```
camera = OrthoCamera(Float32[0,0,10],Float32[0,0,-1],Float32[0,1,0])
render(scaledHouseOfSantaClaus,camera;figNum=5)

# rotate screen clockwise
for t=0:60
    camera = OrthoCamera(Float32[0,0,1],Float32[0,0,-1],Float32[sin(2*pi*t
       /60),cos(2*pi*t/60),0])
    render(scaledHouseOfSantaClaus,camera;figNum=6)
    sleep(0.01)
end
```