Computer Graphics SoSe16
Exercise 1

Tobias Knopp & Martin Hofmann
Institut for Biomedical Imaging

Next exercises:
18.04.2016, 14:15-15:45
25.04.2016, 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 computer graphics 3D objects are often described as a collection of 3D vertices $v_i = (x_i, y_i, z_i)$. These 3D vertices however are usually represented in homogeneous notation as 4D vectors $\vec{w}_i = (x_i, y_i, z_i, 1)$.

   Use the following type definitions

```
type Vec4f
    e1::Float32
    e2::Float32
    e3::Float32
    e4::Float32
end

type Mat4f
    v1::Vec4f
    v2::Vec4f
    v3::Vec4f
    v4::Vec4f
end
```

   for the column vector `Vec4f` and the matrix `Mat4f` with columns `v1`, `v2`, `v3` and `v4` to implement

```
+(v1::Vec4f,v2::Vec4f)
*(a::Float32,v::Vec4f)
*(M::Mat4f,v::Vec4f)
```

   vector-vector addition scalar-vector multiplication and matrix-vector multiplication.

2. For our 3D software renderer we use an `object` type to store the vertices describing geometrical objects. Its definition is given by

```julia
type Object
    vertices::Vector{Vec4f}
    # Type constructorwhich allows to use Object(vec1,vec2,...)
    Object(x::Vector{Vec4f}) = new(x)
    Object(x...) = new(collect(Vec4f,x))
end
```

Write a function `render(object;figAxis=[-1,1,-1,1])` to draw a given object. To do so first apply an orthographical projection on the Object along the z-plane. This can be done at this stage by simply using the `e1` and `e2` coordinate of each vertex in `vertices`. Using these 2D coordinates draw the object by connecting neighbouring vertices in `vertices` using the `plot` function of the `PyPlot` module. To restrict plotting to a specified area call `axis(figAxis)` before `plot`.

Use this function to render a triangle and the house of Santa Clause, which are given by

```julia
# triangle
v1 = Vec4f(0,0,0,1)
v2 = Vec4f(1,0,0,1)
v3 = Vec4f(0,1,0,1)
triangle = Object(v1,v2,v3,v1)
render(triangle)

# houseofsantaclaus
v1 = Vec4f(-1,-1,0,1)
v2 = Vec4f(1,-1,0,1)
v3 = Vec4f(-1,1,0,1)
v4 = Vec4f(0,2,0,1)
v5 = Vec4f(1,1,0,1)
v6 = Vec4f(-1,-1,0,1)
v7 = Vec4f(-1,1,0,1)
v8 = Vec4f(1,1,0,1)
v9 = Vec4f(1,-1,0,1)
houseOfSantaClaus = Object(v1,v2,v3,v4,v5,v6,v7,v8,v9)
render(houseOfSantaClaus, figNum=2, figAxis=[-2,2,-2,2])
```

3. One important aspect in computer graphics is the ability to move and rotate objects or the camera position. In homogeneous space translation, rotation and scaling can be expressed as a linear transformation and hence as 4 by 4 matrix $T \in \mathbb{R}^{4\times 4}$. Any Vertex $\vec{w}_i = (x_i, y_i, z_i, 1)^T$ is transformed by application of the usual matrix-vector multiplication

$$\vec{w} \mapsto T\vec{w}.$$

Note that in usual 3D coordinates the translation would be a non-linear transformation.

Using the `Transformation` type

```julia
type Transformation
    M::Mat4f

    Transformation(v1,v2,v3,v4) = new(Mat4f(v1,v2,v3,v4))
end
```

implement the transformation of a vector and an Object

```
*(T::Transformation,v::Vec4f)
*(T::Transformation,O::Object)
```

Note that an Object is transformed by transforming all its vertices.

4. A change in location maps any $\vec{x} = (x, y, z) \in \mathbb{R}^3$ to $\vec{x} + \vec{t}$ for $t = (t_x, t_y, t_z) \in \mathbb{R}^3$. In homogeneous space this transformation is given by $\vec{w} \mapsto T\vec{w}$, where $\vec{w} = (x, y, z, 1)^T$ and

$$T = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{1}$$

Similar rotations of a vertex ϕ radians around the x-, y-, and z-axes can be written as linear transformation $\vec{w} \mapsto R_i(\phi)\vec{w}$, $i \in \{x, y, z\}$ with

$$R_x(\phi) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi & 0 \\ 0 & \sin\phi & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2}$$

$$R_y(\phi) = \begin{pmatrix} \cos\phi & 0 & \sin\phi & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\phi & 0 & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3}$$

$$R_z(\phi) = \begin{pmatrix} \cos\phi & -\sin\phi & 0 & 0 \\ \sin\phi & \cos\phi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{4}$$

As the last coordinate transform we consider the scaling along the x-, y-, and z-direction. This transform maps any $\vec{x} = (x, y, z) \in \mathbb{R}^3$ to $(s_x x, s_y y, s_z z)$ with the non-zero scaling factors s_x, s_y and s_z. In homogeneous space this transformation is given by $\vec{w} \mapsto T\vec{w}$, where $\vec{w} = (x, y, z, 1)^T$ and

$$T = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{5}$$

Implement all these transformations into the software renderer.

- Write a function `translation(x,y,z)`, which returns (1) as `Transformation`.
- Write functions `rotx(phi)`, `roty(phi)` and `rotz(phi)`, which return (2), (3), and (4) respectively.
- Write a function `scaling(sx,sy,sz)`, which returns (5) as `Transformation`.

Use these to render the following

```
# translate santas house to the right
T = translation(1,0,0)
TranslatedHouseOfSantaClaus = T*houseOfSantaClaus
render(TranslatedHouseOfSantaClaus, figAxis=[-2,2,-2,2])

# rotate santas house out of the xy-plane
T = rotx(pi/4)
RotatedHouseOfSantaClaus = T*houseOfSantaClaus
render(RotatedHouseOfSantaClaus, figAxis=[-2,2,-2,2])

# shrink santas house in y-direction and enlarge it in x-driection
T = scaling(1.25,0.75,1)
ScaledHouseOfSantaClaus = T*houseOfSantaClaus
render(ScaledHouseOfSantaClaus, figAxis=[-2,2,-2,2])
```

5. Write a function that stepwise translates Objects around the unit circle $(\cos 2\pi t, \sin 2\pi t, 0)$ in the xy-plane. Use a `for`-loop for rendering 180 equidistant positions on the circle and use the `sleep` function to adjust the rendering speed.