Image Processing and Computer Graphics
Prof. Dr.-Ing. M. Teschner

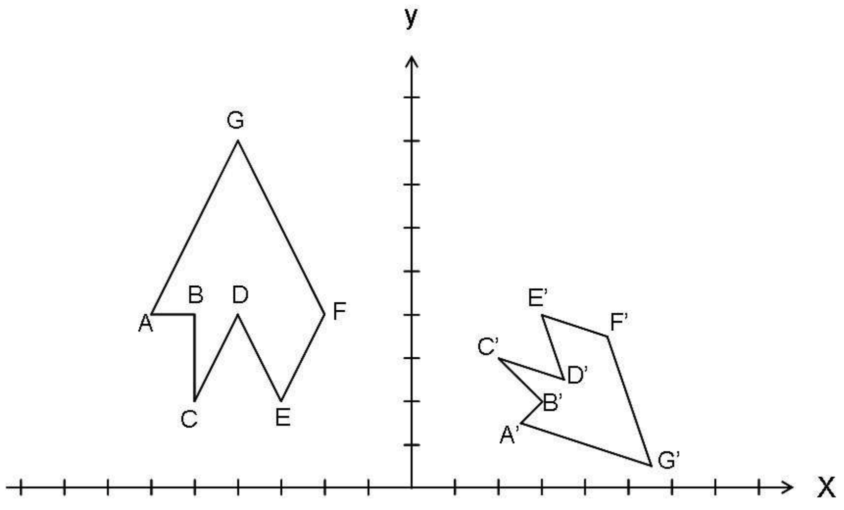
Exercise 1 - Transformations

1 Transformation of objects

1. Transform the object ABCDEF given below to the object A'B'C'D'E'F'. Give matrices in homogenous notation for each step, show in which order they have to be applied and compute the final model transform.
2. If the camera position and viewing direction is given by

$$\mathbf{V} = \begin{pmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 2 \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & -3 \\ 0 & 0 & 1 \end{pmatrix},$$

what is the corresponding ModelView transform?



The coordinates of the points are $A(-6/4)$, $B(-5/4)$, $C(-5/2)$, $D(-4/4)$, $E(-3/2)$, $F(-2/4)$, $G(-4/8)$ and $A'(2,5/1,5)$, $B'(3/2)$, $C'(2/3)$, $D'(3,5/2,5)$, $E'(3/4)$, $F'(4,5/3,5)$, $G'(5,5/0,5)$.

2 Transformation basics in OpenGL

In the following exercises, we get familiar with transformations in OpenGL. Use the commands *glRotated()*, *glTranslated()* and *glMultMatrixd()* for passing transformations to OpenGL. Note that 4×4-matrices are generally stored in linear arrays with 16 values. While C/C++ uses row major order, OpenGL uses column major order for these arrays. Thus, *glMultMatrixd()* expects a 16-value array which denotes a matrix in column major order. There is also the command *glMultTransposeMatrixd()*, which expects matrices in row major order, but it is probably better to get used to the column major notation as it is the standard OpenGL interpretation, and e. g. querying a matrix with *glGetDoublev()* also returns column major matrices. Therefore, we recommend using the *glMultMatrixd()* command.

F.Y.I. We usually think of vectors as column vectors, and therefore, we multiply matrices from the left. However, we could equally interpret vectors as row vectors and multiply matrices from the right. Then, transposing the column vector $\mathbf{Ax}$ yields with $\mathbf{x}^T\mathbf{A}^T$, so multiplying from the left is the same like multiplying the transpose from the right. Second, interpreting a row major matrix as a column major matrix obviously corresponds to taking the transposed matrix. Thus, in the resulting vector there is no difference between interpreting the matrix as column major and multiplying it from the left or interpreting it as row major and multiplying it from the right, and it's up to you to choose your favorite interpretation. However, we would recommend the first version.

a) Use the command *glTranslated()* to translate an object by 2 units in x-direction. Then, use the command *glMultMatrixd()* with an appropriate matrix $\mathbf{T}$ that yields the same result.

b) Use the command *glRotated()* to rotate an object by $45°$ around the y-axis. Again, use *glMultMatrixd()* with a 4×4-matrix $\mathbf{R}$ to get the same result.

c) An object should first be rotated by $\mathbf{R}$ and then translated by $\mathbf{T}$ given in (a) and (b), respectively. Write down the resulting model transform.

d) In which order do you have to perform the *glTranslated()* and *glRotated()* commands in order to get the transformation demanded in (c)? If you are not sure, compute the resulting transformation by hand and use *glMultMatrixd()* to compare the visual result.

e) If the camera is transformed by

$$\mathbf{V} = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & -1 & 0 & 2 \\ 0 & 0 & -1 & 3 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \qquad (1)$$

which is the resulting ModelView matrix?

3 View Transform

In this exercise, we develop the *gluLookAt()* function, which performs the view transform in OpenGL for a given camera position, a point the camera looks at and a given up-vector, i. e. it puts the inverse view transform onto the matrix stack. In exercise 4, this function is re-implemented.

At the beginning, the camera is located at the origin with viewing direction $\mathbf{view}_0 = (0, 0, -1)^T$ and up-vector $\mathbf{up}_0 = (0, 1, 0)^T$. $\mathbf{view}_0$ and $\mathbf{up}_0$ are completed in a natural way by $\mathbf{s}_0 = \mathbf{view}_0 \times \mathbf{up}_0 = (1, 0, 0)^T$ to form an orthonormal basis (an orthonormal basis is a basis where all basis vectors are orthogonal to each other and have unit length).

a) Let $\mathbf{eye} = (0, 3, -4)^T$ be the position of the camera, $\mathbf{center} = (0, 0, 0)^T$ the center point where the camera looks at, and $\mathbf{up} = (1, 0, 0)^T$ be the up-vector of the camera. What is the viewing direction $\mathbf{view}$, which vector $\mathbf{s}$ completes an orthonormal basis in a natural way, and how does the basis look like?

b) The ordering of the standard orthonormal basis is $\mathbf{B}_0 = (\mathbf{s}_0, \mathbf{up}_0, \mathbf{view}_0)$. Which matrix transforms $\mathbf{B}_0$ to $\mathbf{B} = (\mathbf{s}, \mathbf{up}, \mathbf{view})$? Obviously, this is the rotational part of the view transform.

c) Additionally, the camera has to be translated to $\mathbf{eye}$. Which operation should be performed first, translation or rotation?

d) Hence, which matrix has to be put onto the ModelView matrix stack? Write down the 16-value array and keep exercise 2 in mind.

e) If one chooses $\mathbf{up}' = (1, 0.3, -0.4)$ to be the up vector, it wouldn't be orthogonal to the viewing direction, and therefore, the view transform wouldn't be a rotation any more. In order to correct this drawback, one has to orthogonalize $\mathbf{up}'$ manually before continuing.

   Show that $\mathbf{up} := \mathbf{up}' - \mathbf{view}(\mathbf{up}' \cdot \mathbf{view})$ is orthogonal to $\mathbf{view}$.

   Alternatively, let $\mathbf{s} = \mathbf{view} \times \mathbf{up}'$ and $\mathbf{up}^{\times} := \mathbf{view} \times \mathbf{s}$. Then, by construction $\mathbf{view}$, $\mathbf{s}$ and $\mathbf{up}^{\times}$ are orthogonal to each other. Show that $\mathbf{up} = \mathbf{up}^{\times}$.

4 Implementation

In this exercise, we want to become familiar with implementing transformations in OpenGL. Use the TransformationBasics - framework from the course web page. Each part where something should be implemented is marked with a //TODO. In the accompanying executable files, you can see one possible solution, but you are free to implement different scenarios.

The files are prepared for Visual Studio 2008. You are free to use any platform, but your submission has to be compilable with Visual Studio 2008 (i. e. you must not use compiler-specific syntax in the header and code files).

a) Draw some basic objects into the scene (at least two), using e.g. *gluSolidSphere(), gluSolidTorus()* etc. Move them to different positions, and preferably use different colors for the objects. Insert your code in *CViewer::draw()* in viewer.cpp.

b) Implement the *setLookAt()* function in the class CViewer in viewer.cpp, which should reflect the *gluLookAt()* function, and replace the call of *gluLookAt()* in *CViewer::initialize()* in viewer.cpp by *setLookAt()*. Check the correctness of you implementation by comparing either the matrices or the visual result.

   Instead of establishing the inverse view transformation matrix as done in exercise 3, it is easier to perform rotation and translation separately. Thus, establish the rotation matrix according to exercise 3 (a)-(e) and use *glMultMatrixd()* to pass it to OpenGL. Use *glTranslated()* to perform the appropriate translation. Be careful in which order the commands have to be performed (see exercise 2).

c) Use the PreciseTimer-class to introduce different time-dependent relative movements for the objects, e.g. one object could be rotating, one object could be moving back and forth. . .

d) Implement a time-dependent camera rotation around the y-axis. Establish the additional view transform explicitly as a matrix and use the *glMultMatrixd()* command.

5 Resources

- Course homepage: http://cg.informatik.uni-freiburg.de/teaching.htm
- C++ introduction: e. g. http://www.cplusplus.com/doc/tutorial/
- OpenGL web page: http://www.opengl.org/
- OpenGL introduction: e. g. http://www.codeworx.org/opengl_tuts.php
- OpenGL Wiki: http://wiki.delphigl.com
- VisualStudio introduction: http://cg.informatik.uni-freiburg.de/teaching/VisualStudio8.pdf