

This is a “closed book” exam.

Unlike previous years, no printed materials or electronic devices are admitted for use during the exam.

Wishing you lots of success with the exam!

Points per question (Normering)

Q	1	2	3	4	5	6	7	8
	a b	a b c	a b c d	a b	a b	a b	a b c	a b c
P	3 3	3 5 5	4 4 5 4	4 4	6 7	4 8	3 3 3	6 3 3

Total: 90 (+10 bonus) = 100

1. Human Visual System

- Explain the following properties of the human visual system as far as they are relevant for the perception of images generated by computer graphics!
 - human color perception
 - CIE standard observer curve
 - lateral inhibition
- What can a computer-graphics application do to work around the following properties of the human visual system
 - standard observer curve,
 - lateral inhibition ?

2. Viewports

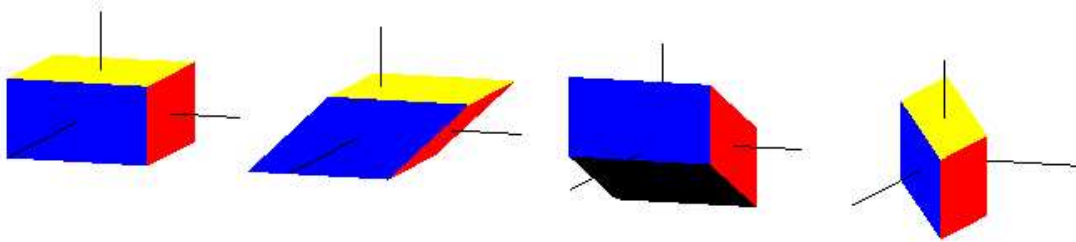
- Explain the terms *viewport* and *aspect ratio*! Give a formula that expresses the aspect ratio for a given viewport!
- Assume, an OpenGL application shall maintain the aspect ratio of its output a_v , even when a user resizes the window. In that case, the application shall use the maximal possible viewport that maintains a_v and that still fits into the reshaped window with its aspect ratio a_w . The viewport shall be centered in the window.

Given a_v and a_w , how many different cases have to be distinguished for finding such a maximal viewport? For each case, draw a simple sketch that shows the window, the viewport, and their respective width and height!

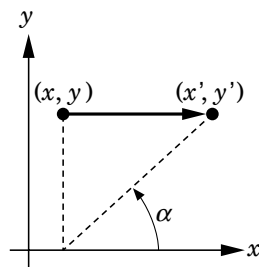
- c) Write a callback function in the C language using OpenGL (the GLUT library) that selects the viewport according to part b! Which (GLUT) function has to be used to register this callback?

3. Shear

The following pictures show (from left to right) a cube, and the same cube sheared along the X axis, the Y axis, and the Z axis.



The following drawing (left) shows how the shear along the X axis can be represented depending on a shearing angle α . H_x is the respective transformation matrix.



$$H_x(\alpha) = \begin{bmatrix} 1 & \cot \alpha & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Make similar drawings, one for the shear along the Y axis (angle β), and one for the shear along the Z axis (angle γ), according to the above picture!
- Determine the shear matrices $H_y(\beta)$, and $H_z(\gamma)$, corresponding to part a!
- Implement a C function `void glShearX(GLfloat alpha)` that works like `glTranslate` or `glRotate` (affecting the currently active transformation matrix by multiplying $H_x(\alpha)$ to it! You can assume that a function `cot()` is available.
- Determine a shear matrix $H(\alpha, \beta, \gamma)$ that combines the effects of $H_x(\alpha)$, $H_y(\beta)$, and $H_z(\gamma)$! Assume you have implemented the function `glShearX`, and also (analogously) `glShearY` and `glShearZ`. Is it possible to implement a function `glShearXYZ(alpha, beta, gamma)` by a combination of calls to `glShearX`, `glShearY`, and `glShearZ`? Give arguments for your answer!

4. Hidden Surface Removal

- Explain *briefly* the painter's algorithm! In which cases does the algorithm fail?

- b) Explain *briefly* the *z*-buffer algorithm! Which issues does the application programmer have to deal with that the algorithm cannot handle by itself?

5. Morphing

Transformation of object shapes from one form to another is called morphing. For morphing, object shapes are defined by so-called *key frames* between which several interpolative steps are generated for a smooth morphing transition. Suppose, you have a complex graphical object represented by the following, recursive C data structure.

```
typedef struct position {
    /* relative to the parent node in the tree: */
    GLfloat tx,ty,tz; /* translation */
    GLfloat sx,sy,sz; /* scaling */
    GLfloat rx,ry,rz; /* rotation axis */
    GLfloat angle;    /* angle of rotation */
} position;

typedef struct node {
    struct position *p1,*p2; /* key frames 1 and 2 */
    void (*attrib_function)(); /* setting attributes, */
                                /* if not NULL */
    void (*drawing_function)(); /* drawing the object */
    struct node *sibling; /* list of sibling nodes */
    struct node *child; /* list child nodes */
} node;
```

- a) Implement a C function:

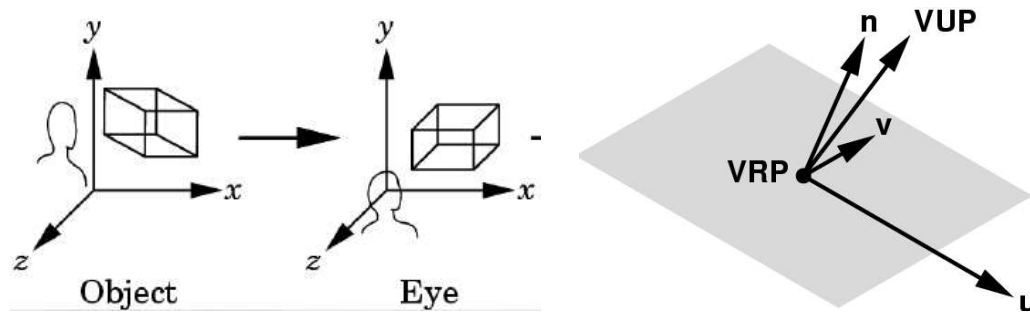
```
void interpolate(struct position *p1, struct position *p2,
                struct position *p3, GLfloat percentage){
```

that computes in p3 the interpolation between p1 and p2 for a position of percentage!
The valid range for percentage is 0..1, where 0 corresponds to p1 and 1 corresponds to p2.

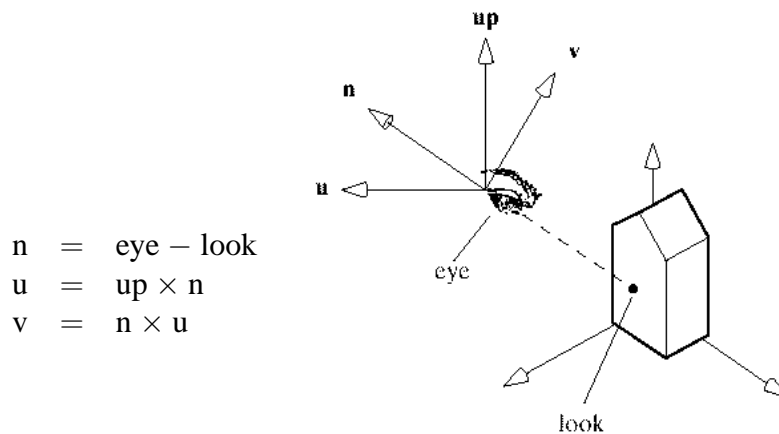
- b) Using your function `interpolate()`, now implement a C function (based on OpenGL) `void morph(struct node *tree, GLfloat percentage)`, that displays the graphical object given in tree! Your function shall produce a single image of the object that corresponds to a frame that lies percentage% between p1 and p2. (Your function has to traverse all nodes of the tree data structure.) The function `attrib_function` allows to set attributes like color or material properties; it shall only be called if the pointer is not NULL.

6. Viewing

- a) Explain (briefly) how the transformation from object (world) coordinates to eye (camera) coordinates contributes to a viewing API! What are the components of a u-v-n viewing coordinate system?



- b) The function `gluLookAt(eyex, eyey, eyez, lookx, looky, lookz, upx, upy, upz)` internally uses a u-v-n coordinate system:



$$\begin{aligned} n &= \text{eye} - \text{look} \\ u &= \text{up} \times n \\ v &= n \times u \end{aligned}$$

`gluLookAt` first normalizes n, u, v to unit length and then uses the normalized vectors to build up the viewing matrix:

$$V = \begin{pmatrix} u_x & u_y & u_z & d_x \\ v_x & v_y & v_z & d_y \\ n_x & n_y & n_z & d_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (d_x, d_y, d_z) = (-\text{eye} \cdot u, -\text{eye} \cdot v, -\text{eye} \cdot n)$$

Show that u, v, n are mutually perpendicular (orthogonal)!

Show that the matrix V properly converts object coordinates to eye coordinates by demonstrating that it maps eye to the origin $(0, 0, 0, 1)^T$, u to $(1, 0, 0, 0)^T$, v to $(0, 1, 0, 0)^T$, and n to $(0, 0, 1, 0)^T$!

7. Polygon Shading

- a) Explain the basic idea of the *Phong reflection* model! Draw a simple figure that shows the vectors involved in computing the shade of a given point on the surface of an object!
- b) Explain how *flat shading* works, for example for a polygonal mesh! What are the advantage and the disadvantage of flat shading?
- c) Explain *Phong shading* and how it improves over the disadvantage of flat shading!

8. Antialiasing

- a) Although an ideal pixel is a square of 1 unit per side, most CRT systems generate round pixels that can be approximated as circles of uniform intensity. If a completely full unit square has intensity 1.0, and an empty square has intensity 0.0, how does the intensity of a displayed pixel vary with the radius of the circle? Compute the intensity of a pixel with radius 0.5! Also compute the radius r_1 that corresponds to the intensity 1.0! Which side effect has the use of pixels with radius r_1 ?

(Hints: approximate your calculations with 1 or 2 decimal digits, $\sqrt{\pi} \approx 1.8$)

If the intensity of the whole screen is the percentage of pixels set, multiplied by the intensity of the individual pixel: does the intensity of the screen also equal 1.0 when using pixels of radius r_1 and setting all pixels?

- b) Defocusing the beam of a CRT is sometimes called “the poor man’s antialiasing.” Can you explain why?
- c) A standard antialiasing technique used in ray tracing is to cast rays not only through the center of each pixel, but also through the four corners of the pixel. What is the increase in necessary computation compared to using only a single ray per pixel?