Department of Computer Science	Exam **Computer Graphics**
Vrije Universiteit	29 – 04 – 2003

Language disclaimer:
You are kindly asked to answer the questions using the English language. However, if it helps clarifying your answers, you may use *some* Dutch here and there. Doing so, *will not affect* your result.

Allowed material:
This is an *"open book"* exam. For answering the questions, you are allowed to use all kinds of written material like textbooks, printouts of the lecture slides, your own notes, etc.
However, it is **not allowed** to use any electronic equipment or any means of communication.

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	c	a	b	c
P	3	3	3	5	5	4	5	5	4	7	6	7	4	8	2	3	3	4	5	4

Total: 90 (+10 bonus) = 100

1. Human Visual System

a) 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

b) What can a computer-graphics application do to work around the properties of the human visual system, namely standard observer curve and lateral inhibition?

2. Viewports

a) Explain the terms *viewport* and *aspect ratio*! Give a formula that expresses the aspect ratio for a given viewport!

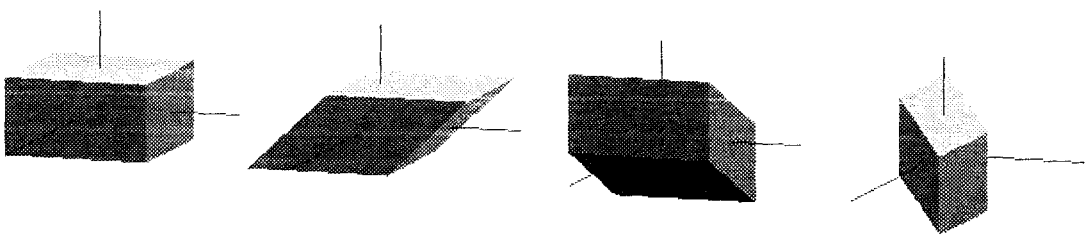
b) Assume, an OpenGL application shall maintain the aspect ratio of its output a_v, even when a user resizes the window. In that case, the application shall use the maximal possible viewport that maintains a_v and that still fits into the reshaped window with its aspect ratio a_w. The viewport shall 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!

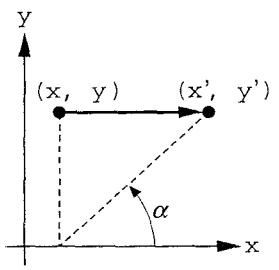
c) Write a callback function in the C language using OpenGL (the GLUT library) that selects the viewport according to part b! Which (GLUT) function has to be used to register this callback?

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.



a) The following drawing shows how the shear along the X axis can be represented depending on a shearing angle α. 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!



b) Determine the shear matrices $H_x(\alpha)$, $H_y(\beta)$, and $H_z(\gamma)$, corresponding to part a!

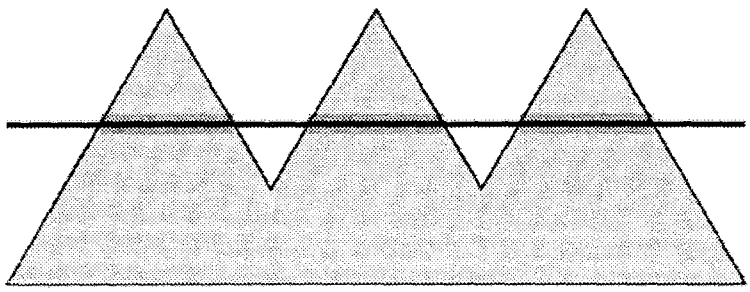
c) Implement a C function `void glShearX(GLfloat alpha)` that works like glTranslate or glRotate (affecting the currently active transformation matrix by multiplying $H_x(\alpha)$ to it! You can assume that a function cot() is available.

d) Assume you have implemented the function glShearX, and also (analogously) glShearY and glShearZ. Determine a shear matrix $H(\alpha, \beta, \gamma)$ that combines the effects of $H_x(\alpha)$, $H_y(\beta)$, and $H_z(\gamma)$!

Is it possible to implement a function `glShearXYZ(alpha,beta,gamma)` by a combination of calls to glShearX, glShearY, and glShearZ? Explain why!

4. Polygon Filling

Show how you can use the XOR operation to implement the odd-even polygon fill algorithm! Assume the simple case in which 0 is the background color and 1 is the edge and fill color. Which writing mode do you use to modify the image?
(Hint: The edges are already drawn before the polygon will be filled.)



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:

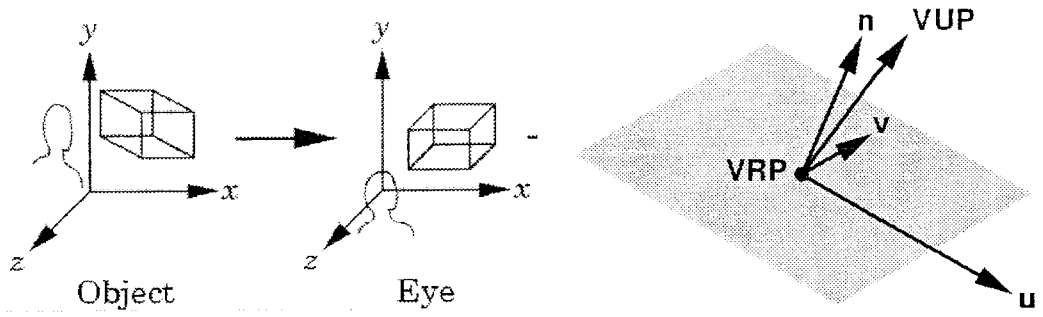
```
void interpolate(struct position *p1, struct position *p2,
                 struct position *p3, GLfloat percentage)}
```

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`.

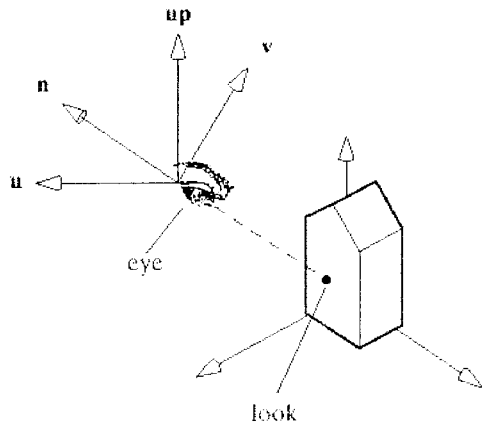
b) Using your function `interpolate()`, now implement a C function (based on OpenGL) `void morph(struct node *tree, GLfloat percentage)`, that displays the graphical object given in `tree`! Your function shall produce a single image of the object that corresponds to a frame that lies `percentage`% between `p1` and `p2`. (Your function has to traverse all nodes of the tree data structure.) The function `attrib_function` allows to set atributes 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:



n = eye − look
u = up × n
v = n × u

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

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

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!

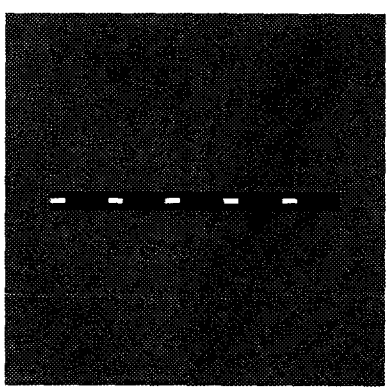
b) Explain how *flat shading* works, for example for a polygonal mesh! What are the advantage and the disadvantage of flat shading?

c) Explain *Phong shading* and how it improves over the disadvantage of flat shading!

8. Texture

Write a program that shows a road: The road is 15 meters long and 1 meter wide. Every 3 meters, there is a white dash painted on the asphalt. Each dash is 75 cm long and 25 cm wide. The pattern of the road shall be determined by an 8×8 texture pattern that is mapped onto a quad. Start with the following main program. The desired output of your program is shown on the right side.

```
int main (int argc, char** argv)
{
 glutInit(&argc, argv);
 glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE );
 glutInitWindowSize (500, 500);
 glutCreateWindow (argv[0]);
 glClearColor(0.0,0.0,0.7,1.0);
 glMatrixMode(GL_PROJECTION);
 glLoadIdentity();
 gluOrtho2D(-10.0,10.0,-10.0,10.0);
 glMatrixMode(GL_MODELVIEW);
 glLoadIdentity();
 Init_Texture();
 glutDisplayFunc(Display);
 glutMainLoop();
 return 0;
}
```



a) Define and initialize the data structure for your 8×8 texture!
(The texture shall **NOT** be read from a file.)

b) Implement the function `void Init_Texture(void)` that does all initialization necessary for the texturing!

c) Implement the function `void Display(void)` that actually displays the road!