Divisie Wiskunde en Informatica	Tentamen **Computer Graphics**
Vrije Universiteit	27 – 03 – 2002

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

Total: 90 (+10 bonus) = 100

1. Input Devices

a) What are the *measure* and *trigger* of an input device? What are the *measure* and *trigger* of a *mouse*? How can this information be derived from the physical mouse device?

b) Explain the different input modes: request mode, sample mode, event mode! Give examples for each mode! Which input mode is implemented in OpenGL's callback functions? And what is the advantage of doing so?

2. Aspect Ratio

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

b) Assume, a video player (written with OpenGL) has a window with the aspect ratio 16:9. This video player shall display videos with the traditional aspect ratio 4:3. This can be done either by distorting the image and using the entire window, or by keeping the aspect ratio and not using parts of the window.

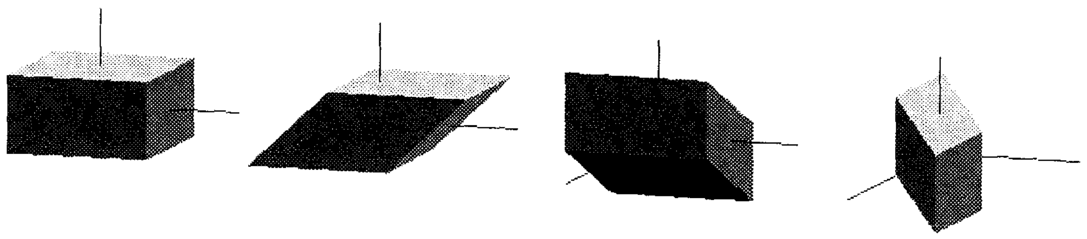
Draw a simple sketch that shows the window and the video image (maximal possible size within the window), centered in the window, keeping the aspect ratio 4:3!

c) The user of the video player program shall be able to select between the two viewing modes. Write a keyboard callback function that selects using the full screen (with distortion of the image) when the key `f` is pressed, and that selects showing the video with the original aspect ratio and black bars at the left and right side when the key `b` is pressed! (Your keyboard callback shall select a suitable viewport.)

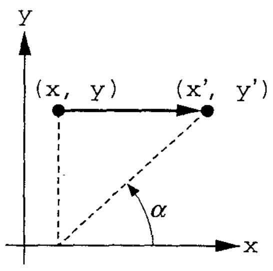
The callback function can access the height of the window through the global variable `h`.

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!

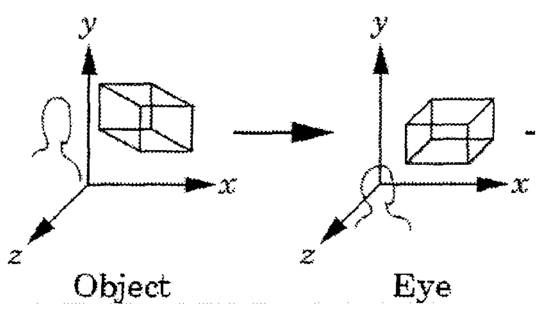
c) Implement a C function `void glShearY(GLfloat beta)` that multiplies $H_y(\beta)$ to the currently active transformation matrix! (Just as glTranslate or glRotate works.) You can assume that a function cot() is available.

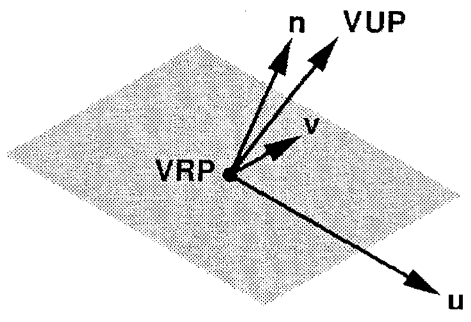
d) Assume you have implemented the function glShearY, and also (analogously) glShearX. Are these two functions commutative?

Implement a function `void glShearXY(GLfloat alpha, GLfloat beta)` that performs both individual shear opearations!

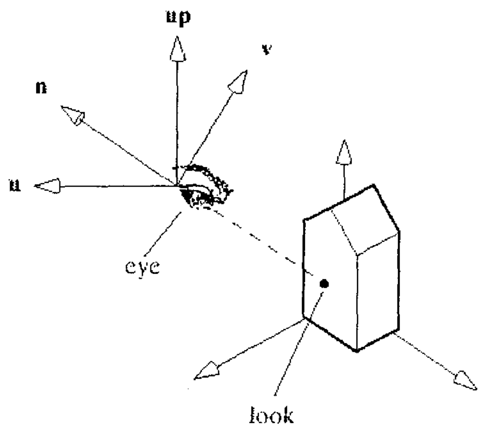
6. Viewing

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` 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. Phong Reflection and 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!

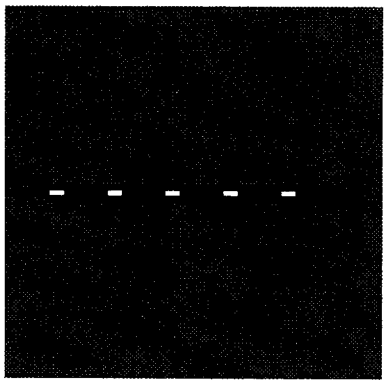
b) Explain how *flat shading* works, for example for a polygonal mesh! What are the advantage and the disadvantage of flat shading?

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 20 meters long and 1 meter wide. Every 4 meters, there is a white dash painted on the asphalt. Each dash is 150 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(-13.0,13.0,-13.0,13.0);
 glMatrixMode(GL_MODELVIEW);
 glLoadIdentity();
 Init_Texture();
 glutDisplayFunc(Display);
 glutMainLoop();
 return 0;
}
```



a) Define the data structure for your 8 × 8 texture and initialize it (assign data values)!

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!

5. Animating Graphics Objects

Suppose, for an animation program 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;
```

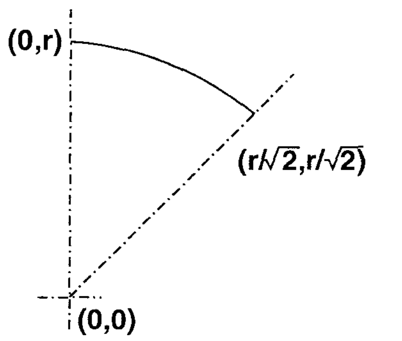
For animating a scene, the graphics object represented with this data structure can be displayed at various intermediate steps between two so-called *key frames*. Assume, you have a C function

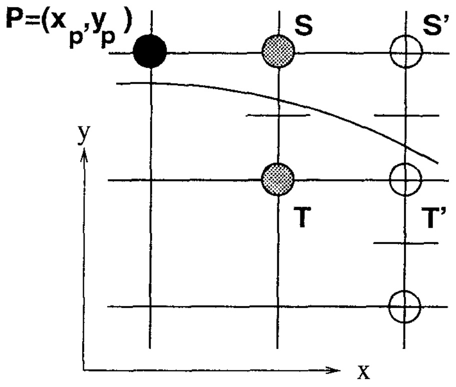
```
void interpolate(struct position *p1, struct position *p2,
                 struct position *p3, GLfloat percentage);
```

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`.
Implement a C function (based on OpenGL):
`void display(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 p % 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`.

4. Bresenham's Algorithm

Mr. Bresenham also developed an algorithm for drawing circles. It works similar to his famous line drawing algorithm. The following picture shows at its left side the second octant of a circle with radius r and origin $(0,0)$. In this octant, the arc that is part of the circle can be drawn, starting from the 12 o'clock position, by incrementing x and choosing the y coordinate closest to the circle. (Points (x,y) lie on the circle iff $x^2 + y^2 = r^2$)





For any point (x,y), we can use the decision variable $d = x^2 + y^2 - r^2$.
$d = 0$ if the point lies on the circle. $d > 0$ if the point lies outside, and $d < 0$ if the point lies inside the circle.

a) Assume the last point that has been drawn is $P = (x_p, y_p)$ (see the picture above, on the right side). The next point for drawing the circle is chosen from S and T. Give the formula for d for the midpoint between S and T!

b) Bresenham's algorithm works incrementally. The decision variable d' shall be used for choosing the next point, after one of S or T has been chosen. For both cases, give the respective formula for d', relative to d!

c) For starting at the 12 o'clock position, give the initial value for d! Also give an initialization for an integer-valued decision variable d_i that most closely approximates d!

d) Implement a C function `void DrawArc(int Radius)` that draws the octant of a circle with the center at the origin $(0,0)$ that is shown in the above picture on the left side! Assume, you have a function `void setPixel(int x, int y)` available. Your function shall not use any floating point arithmetic.