Department of Computer Science	Exam **Computer Graphics**
Vrije Universiteit	21 – 01 – 2003

Language disclaimer:
You are kindly asked to answer the questions using the English language. However, if it helps clarifying your answers, you may also 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	c	a	b	c	d	a	b	a	b	c	a	b	a	b	a	b	a	b	c
P	3	4	4	1	4	2	4	5	5	4	6	4	3	3	4	8	3	10	4	5	4

Total: 90 (+10 bonus) = 100

1. 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 application (written with OpenGL) has a window with the traditional aspect ratio 4:3. This video player shall now display videos with the new aspect ratio 16:9. 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.

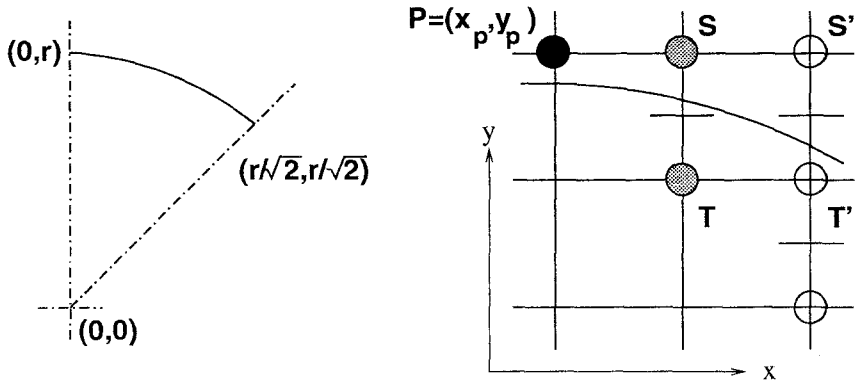
Draw a simple sketch that shows the window and the video image (maximal possible size within the window), centered in the window, keeping the aspect ratio 16:9!

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 window (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 two sides when the key `b` is pressed! (Your keyboard callback shall select a suitable viewport.)

The callback function can access the width of the window through the global variable `w`.

2. 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.

3. 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?

4. 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, for an animation program you need to have a complex graphical object represented by a recursive C data structure. The complex object shall be represented by a *child-sibling-tree*; each node in this tree is placed into the scene relative to its parent node by a translation, a scaling, and a rotation around a given axis.

a) Define two C data structures:

- `struct position`, contains the necessary information to place a node into the scene, relative to its parent node in the tree.
- `struct node` contains two `position` entries, one for the start position and one for the end position of a morphing sequence. `struct node` also contains two function pointers, one for drawing the object itself and one for setting attributes (like material properties). Of course, `struct node` contains the necessary pointers for forming the child-sibling-tree.

b) Assume, somebody implemented for you 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 p)`
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.)

c) For your `display` function from part b), explain how it works, especially why the order of the individual operations correctly displays the complex object while traversing the tree!

5. Human Visual System

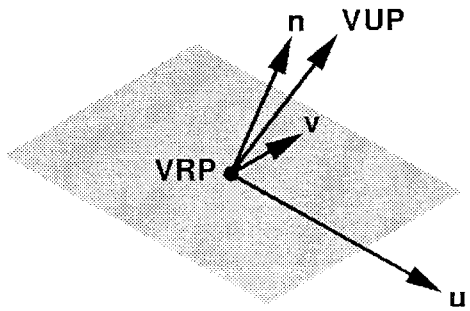
a) Explain the following properties of the human visual system as far as they are relevant for the perception of images generated by computer graphics!

- human color perception
- CIE standard observer curve
- lateral inhibition

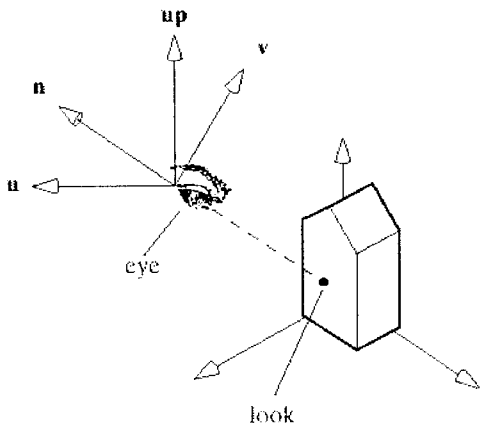
b) What can a computer-graphics application do to work around the properties of the human visual system, namely standard observer curve and lateral inhibition?

6. Viewing

a) Explain (briefly) how the transformation from object (world) coordinates to eye (camera) coordinates contributes to a viewing API! Explain the components of a u-v-n viewing coordinate system, as shown in the figure:



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



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

`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. Affine Transformations

In a 2D homogeneous coordinate system, each point P can be represented as $P = P_0 + xv_1 + yv_2$

a) For this coordinate system, identify the **matrices** for the following transformations:

- $T(t_x, t_y)$, for a *translation* by the vector $[t_x, t_y, 0]^{-1}$
- $S(s_x, s_y)$, for a *scaling* with the scalars s_x and s_y (and the fix point in the origin)
- $R(\theta)$, for a *rotation* around the origin by an angle θ

b) Let $T_1, T_2, S_1, S_2, R_1, R_2$ be translations, scalings, and rotations, as defined by the matrices from part a). Which of the following transformation pairs are commutative? Show why!

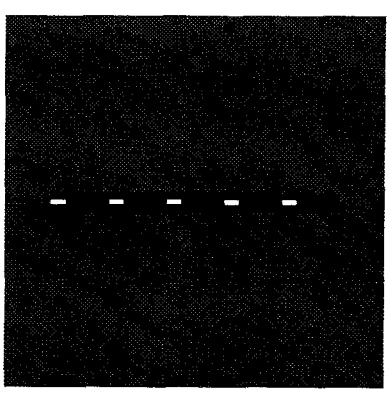
i) T_1, T_2
ii) S_1, S_2
iii) R_1, R_2
iv) T_1, S_1
v) T_1, R_1

Hint: $\sin(\alpha \pm \beta) = \sin\alpha \cdot \cos\beta \pm \cos\alpha \cdot \sin\beta$
$\cos(\alpha \pm \beta) = \cos\alpha \cdot \cos\beta \mp \sin\alpha \cdot \sin\beta$

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!