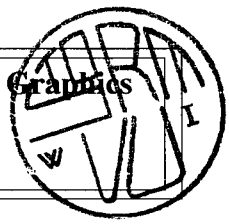
Department of Computer Science	Exam **Computer Graphics**
Vrije Universiteit	21 – 06 – 2004



This is a *"closed book"* exam.
No printed materials or electronic devices are admitted for use during the exam.
You are supposed to answer the questions **in English**.

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	a	b	a	b		a	b	c	a	b
P	3	3	4	4	4	9	10	4	4	3	7	14	3	3	3	6	6

Total: 90 (+10 bonus) = 100

1. Color

a) Explain how different colors are composed in the RGB model! Why can the RGB model create colors that are useful for the human observer?

b) Explain how the CMY color model differs from RGB! What is the main application area for the CMY model?

2. 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?

c) Write a callback function in the C language using OpenGL (the GLUT library) that allows a user to select a range in the active window using the mouse! The user selects a range by moving the mouse to one corner of the range, pressing the left mouse button, moving the mouse (while keeping the button pressed) to the opposite corner of the range, and finally releasing the mouse button. The screen coordinates have to be filled into the following struct variable `range` (see below/next page). (It is not necessary that your callback function visualizes the range while selecting.)

```
typedef struct {
  int x_start, y_start; /* first corner */
  int x_end, y_end;     /* opposite corner */
} range_type;
range_type range;
```

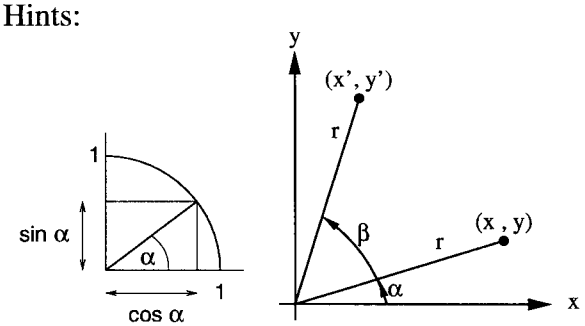
3. Affine Transformations

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 $[t_x, t_y, 0]^T$
- $S(s_x, s_y)$, for a *scaling* with the scalars s_x and s_y (and the fix point in the origin)
- $R(\beta)$, for a *rotation* around the origin by an angle β

Hints:



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

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

4. Hidden Surface Removal

a) 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 algortihm cannot handle by itself?

5. Graphics Objects

a) Explain the notion and use of "instance transformation" for graphics objects!

b) 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;
```

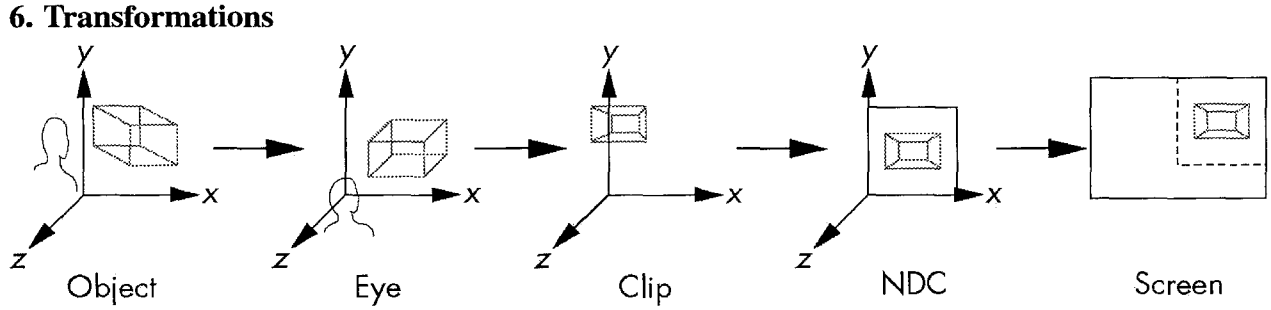
For animating a scene, the graphics object represented with this data structure can be displayed at various intermediate steps between two so-called *key frames*. Assume, you have a C function

```
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`.

6. Transformations



The image above (previous page) shows the transformations from objects to the screen. Explain the the diagram, focusing on the following terms:

1. clip coordinates
2. clipping
3. distortion
4. eye coordinates
5. frustum
6. model-view matrix
7. NDC coordinates
8. object coordinates
9. parallelepiped
10. perspective division
11. projection
12. screen coordinates
13. viewport
14. view volume

7. Polygon Shading

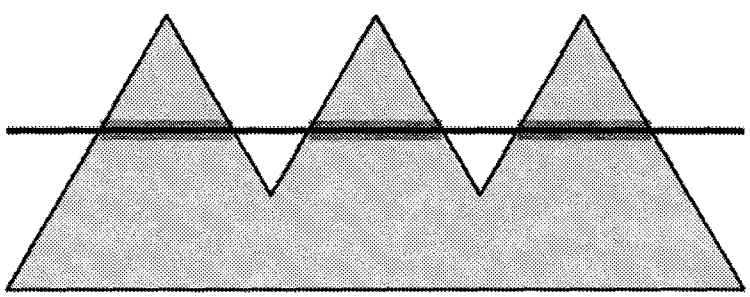
a) Explain the basic idea of the *Phong reflection* model! Draw a simple figure that shows the vectors involved in computing the shade of a given point on the surface of an object!

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. Polygon Filling

a) 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/operation do you use to modify the image?
(Hint: The edges are already drawn before the polygon will be filled.)



b) Implement the flood fill algorithm as a C function
`void flood_fill(int x, int y, color_t color);`
You can read the color of a pixel using `color_t read_pixel(int x, int y);`
and you can write a pixel using `void write_pixel(int x, int y, color_t color);`
Your function `flood_fill()` can assume that the polygon has been drawn as pixels of color `color` and that the pixel (x, y) lies inside the polygon.