Divisie Wiskunde en Informatica	Tentamen **Computer Graphics**
Vrije Universiteit	29 – 01 – 2001

Solutions

(solutions in sans-serif font)

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

Allowed material:
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!

Normering

Vraag	1		2			3						4			5			6		7		8
	a	b	a	b	c	a	b	c	d	e	f	a	b	c	a	b	c	a	b	a	b	
Punten	4	4	4	4	2	4	3	3	4	6	2	3	6	2	3	3	3	6	6	2	10	6

Totaal: 90 (+10) = 100

1. Synthetic Camera Model

a) Describe how an image is constructed using the synthetic camera model!

Solution:
The synthetic camera model imitates the image construction in a real photo camera. A camera is described by its position in the coordinate system of the scene to be displayed, by its direction, and angle of view (based on size of its image plane, and the distance to the lens). The objects in the scene to be displayed are described by their properties (position, size, orientation, color, etc.) The image is formed by projecting the objects to the image plane; all projectors (straight lines) go from points on the surface of an object through the center of projection (camera lens) to the image plane (or are clipped if they miss the image plane).

To avoid the image being upside-down, the image plane of the synthetic camera is moved in front of the camera lens (center of projection).

b) What is the most important advantage of the synthetic camera model, compared to the pen-plotter model?

Solution:
With the synthetic camera model, the objects and the camera are described independently of each other. This allows to describe the objects as such and the camera (position, clipping area, etc.) separately, relative to the objects.

With the pen-plotter model, the 2D image of the 3D objects is described directly. Thus, when the objects are to be shown, for example, from a different angle, all operations for drawing the objects have to be changed.

2. Callback Functions

a) For which purpose do interactive graphics environments (like OpenGL) use the callback function mechanism? Name four examples of callback functions used by OpenGL!

Solution:
In interactive graphics environments, applications have to react to several possible types of events. To avoid having programmers to check for events explicitly, the control flow is taken over by the graphics system (e.g. OpenGL). The application programmer then simply provides callback functions containing the code to be executed whenever a certain event occurs.

Examples of OpenGL callback functions: display callback, mouse callback, keyboard callback, reshape callback, idle callback

b) 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.)

Solution:

```
void mouse_callback(int button, int state, int x, int y){
      if ( button==GLUT_LEFT_BUTTON && state == GLUT_DOWN){
            range.x_start = x;
            range.y_start = y;
      }
      if ( button==GLUT_LEFT_BUTTON && state == GLUT_UP){
            range.x_end = x;
            range.y_end = y;
      }
}
```

c) How can your OpenGL application ensure that the results of the mouse callback have any effect on the displayed image?

Solution:
Finally, the *display* callback has to be activated (called). This can happen either from inside the mouse callback (via `glutPostRedisplay()`) or regularly from inside the *idle* callback.

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

3. 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?

Solution:
In the RGB model, colors are composed from the three basic colors red, green, and blue, like adding the light of three spots on a black background.

The human eye has three kinds of color receptors that together send their "tri-stimulus" to the brain. Different colors that result in the same tri-stimulus can not be distinguished by the human brain, allowing for example to compose arbitrary colors from three basic colors, like red, green, blue.

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

Solution:
In the CMY model, colors are mixed from the three basic colors cyan, magenta, and yellow. Unlike RGB, CMY assumes a white background. By adding color to the white background, reflectivity in the related frequencies is *subtracted* rather than *added* as with RGB. CMY is used for color printers (using white paper).

c) Sketch a color cube for the RGB model in a homogeneous coordinate system defined by the frame (v_1, v_2, v_3, P_0) with:

$$v_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad v_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \quad v_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \quad P_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

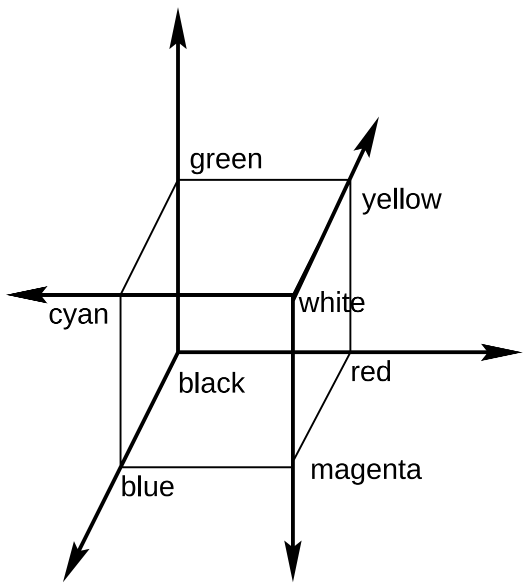
Here, a color c is denoted by its representation $c = P_0 + r \cdot v_1 + g \cdot v_2 + b \cdot v_3, 0 \leq r, g, b \leq 1$ Mark the eight corners of the cube with the respective colors red, green, blue, black, white, cyan, magenta, yellow!

d) Add to your drawing from part c) the three vectors of a second frame

$$u_1 = \begin{bmatrix} -1 \\ 0 \\ 0 \end{bmatrix} \quad u_2 = \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix} \quad u_3 = \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix} \quad Q_0 = P_0 + v_1 + v_2 + v_3$$

(This frame denotes the corresponding color cube for the CMY model.)

Solution to c) and d):



e) Identify the matrix M with

$$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ P_0 \end{bmatrix} = M \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ Q_0 \end{bmatrix}$$

Compute M^T and then $A = \left(M^T\right)^{-1}$!
Hint: A can be computed easily because of its very simple structure; keep in mind that:

$$M^T \left(M^T\right)^{-1} = I = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Using A, compute the RGB representation c' of a color from its CMY representation as

$$c' = Ac = A \begin{bmatrix} c \\ m \\ y \\ 1 \end{bmatrix}$$

Solution:

$$M = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \qquad M^T = \begin{bmatrix} -1 & 0 & 0 & 1 \\ 0 & -1 & 0 & 1 \\ 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad A = \left(M^T\right)^{-1} = M^T$$

$$c' = \begin{bmatrix} r \\ g \\ b \\ 1 \end{bmatrix} = \begin{bmatrix} 1-c \\ 1-m \\ 1-y \\ 1 \end{bmatrix}$$

f) Using the result from part e), write a C function
`void cmyColor3f(GLfloat c, GLfloat m, GLfloat y)`
that takes a color in the CMY system as parameter, computes the corresponding RGB values, and calls `glColor3f` to set the color in an OpenGL system!

Solution:

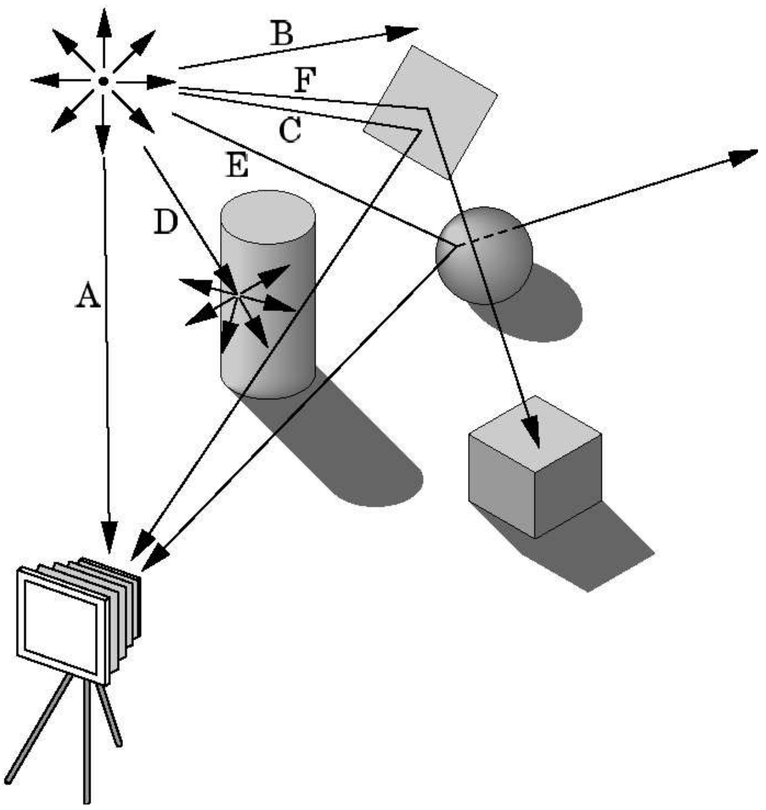
```
void cmyColor3f(GLfloat c, GLfloat m, GLfloat y){
    GLfloat r,g,b;
    r = 1.0-c;
    g = 1.0-m;
    b = 1.0-y;
    glColor3f(r,g,b);
}
```

4. Ray Tracing

a) Explain how images are created using the ray tracing technique!

Solution:
With ray tracing, the rays of light emitted from the light sources in a scene are followed (traced) until they enter the virtual camera or disappear from the scene. When light rays hit an object, they are specularly reflected, refracted, and/or diffusely reflected, possibly all three effects at the same time. This depends on the material properties of the objects. For reducing the necessary computational effort, rays are traced in the reverse direction, from the camera via the scene to the light sources, leaving out many rays emanating from the light sources that immediately disapear and do not contribute to the final image.

b) Look at the rays denoted *A* to *F* in the following picture! What happens to each ray and how does it contribute to the image created in the camera?



Solution:

- A This ray directly enters the camera and shows the light source in the image.
- B This ray disappears from the scene and does not contribute to the image.
- C This ray is specularly reflected by an object and shows the object's color in the image.
- D This ray is diffusely reflected by an object, creating many new rays that have to be traced individually.
- E This ray is partially reflected and refracted. The reflected part shows the object in the image, the refracted part disappears and does not contribute to the image.
- F This ray is first reflected and then absorbed (not or very weakly reflected) by another object; it does not contribute to the image.

c) What is the most important disadvantage of ray tracing because of which it is not used, for example, by OpenGL?

Solution:
The computation for ray tracing is very complex and takes a lot of time. Therefore, interactive graphics as with OpenGL can not use this technique.

5. Antialiasing

a) 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?

Solution:
Although each pixel uses five rays, the total number of rays to be computed has only doubled; think of a second grid that is offset half a pixel in both directions. In addition, at one side each in both directions, an additional row of rays has to be computed, which can be almost neglected compared to the inner pixels of the image.

b) 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 corresponds to the intensity 1.0! Which side effect has the use of pixels with radius r_1?
(Hint: approximate your calculations with up to 2 decimal digits.)

If the intensity of the 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?

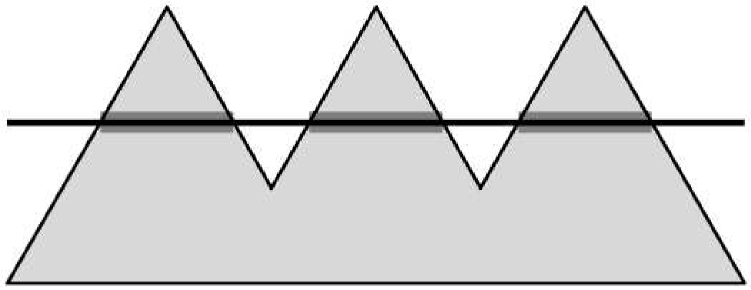
Solution:
The intensity of a round pixel corresponds to the area of a circle with radius r: $I = \pi \cdot r^2$
With $r = 0.5$, we get: $I = \pi \cdot 0.5^2 \approx 3.14 \cdot 0.25 = 0.785$
$1 = \pi \cdot r_1^2 \longrightarrow r_1 = \frac{1}{\sqrt{\pi}}$ With $\sqrt{\pi} \approx 1.8$ we get $r_1 \approx 0.55$.
The side effect of using pixels with radius 0.55 is that they slightly overlap. Because the pixels do overlap, the intensity of a screen with all pixels set will be less than 1.0; some space in the corners of the rectangular pixel areas will still not be illuminated.

c) Why is defocusing the beam of a CRT sometimes called "the poor person's antialiasing"?

Solution:
With antialiasing, the staircase effect shall be softened that is caused by binary "on/off"decisions while approximating lines by individual pixels. Using antialiasing, neighboring pixels also become slightly illuminated. By defocusing a CRT beam, pixels slightly overlap, producing a similar effect.

6. XOR

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



Solution:
Suppose that we move across a scanline left to right starting on the outside of a polygon. As we move across the scanline we replace each point by the exclusive or of its value with the value of the point on the left (which has already been processed). If we start on the outside of any polygon, we can assume the first point is a 0. Thus as long as we remain outside the first polygon we generate $0 \otimes 0 = 0$. When we encounter the first edge, a 1, we compute $1 \otimes 0 = 1$ (we have a 0 from the last point XORed with the edge). As we proceed inside the polygon we compute $1 \otimes 0 = 1$ because we have a 1 from the last XOR and a 0 from the unfilled point. When we encounter the second edge, we compute $1 \otimes 1 = 0$, thus returning to our initial situation.
As described so-far, the XOR operation allows to compute which pixels have to be set by the algorithm. However, if we directly use the XOR for writing into the image, then every second edge gets deleted. Instead, the image and the result from the XOR operation have to be combined by a OR writing operation.

b) Reconsider the range selection from question 2.b)
Suppose you wish to visualize the currently selected rectangle, while the mouse button is being pressed. Assume that the mouse callback may not only get invoked when a mouse button is pressed or released, but also when the mouse is moved while a button is being held down. In that case, assume the button state to be GLUT_HOLD. Also assume that you have a function that allows you to draw a rectangle in XOR writing mode. How would you implement the visualization of the range selection?

Solution:
For the visualization, an additional data structure is needed that stores the two opposite corners of the previously visualized rectangle. While processing the GLUT_DOWN event, initialize this data structure with both corners having the coordinates of the current mouse position; also draw (XOR write) the corresponding rectangle, consisting of only a single pixel. While processing the GLUT_HOLD events, first re-draw

the old rectangle (thus restore the initial state). Then modify the new data structure such that the rectangle now is defined by the mouse position of the GLUT_DOWN event and the current mouse position, and XOR-draw the according rectangle. While processing the final GLUT_UP event, just re-draw (restore) the most recently drawn rectangle.

7. Graphics Objects

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

Solution:
Instance transformation is used for objects stored in graphics libraries. In the library, for each class of object only the information (like a display list) is stored that is needed to draw the object. However, several instances of the same object class can be displayed in an image; they can be distinguished by their instance transformation. The instance transformation consists of translation, rotation, and scaling of the generic object description into the desired place and shape of the instance.

b) Suppose you want to animate the movement of the parts of a complex machine. For the individual parts, you can use graphical objects from a library. Your animation shall show the transition of the machine from one position to a second position, expressed via the rotation angles between the parts of the machine. Assume the description of the machine already exists in form of a tree of nodes of type node, representing the hierarchical structure of the library objects forming the machine.

Write a C function void animate(node *tree, int step, int total_steps) that produces one image of the machine! (Your function shall use OpenGL to produce the image.) The transition from the first to the second position of the machine shall happen in total_steps steps; your function should just display the step step $\in 0 \ldots$ total_steps $- 1$. The struct element positive gives the direction of the movement; if positive == TRUE, then the angle shall be increased from angle1 to angle2, and decreased otherwise. The function attrib_function allows to set atributes like color or material properties; it is optional and shall only be called if the pointer is not NULL;

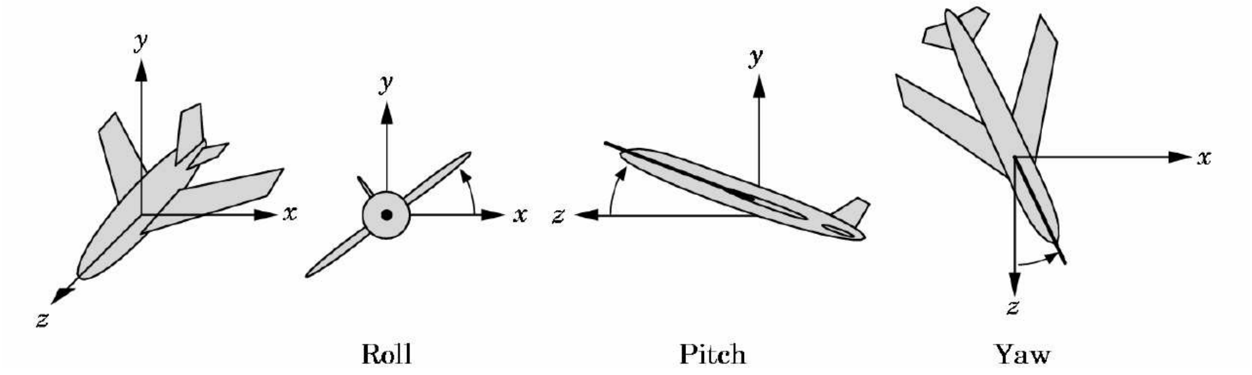
```
#define FALSE 0
#define TRUE  1
typedef struct node {
  /* relative to the parent node in the tree:          */
  GLfloat tx,ty,tz;           /* translation           */
  GLfloat sx,sy,sz;           /* scaling               */
  GLfloat rx,ry,rz;           /* rotation axis         */
  GLfloat angle1,angle2;      /* angle of rotation,    */
                              /* position 1 and 2      */
  int     positive;           /* direction of movement */
                              /* TRUE or FALSE         */
  void (*attrib_function)();  /* setting attributes,   */
                              /* if not NULL           */
  void (*drawing_function)(); /* draw the object       */
  struct node *sibling;       /* list of sibling nodes */
  struct node *child;         /* list child nodes      */
} node;
```

Solution:

```
void animate(node *tree, int step, int total_steps){
    GLfloat a1,a2,a;
    if ( tree == NULL) return;
    glPushMatrix();
    glPushAttrib(GL_ALL_ATTRIB_BITS);
    glTranslatef(tree->tx,tree->ty,tree->tz);
    glScalef(tree->sx,tree->sy,tree->sz);
    a1 = tree->angle1;
    a2 = tree->angle2;
    if ( tree->positive ){
      if ( a2 < a1 ) a2 += 360.0;
      a = a1 + (a2-a1)*step/total_steps;
    }else{
      if ( a1 < a2 ) a1 += 360.0;
      a = a1 - (a1-a2)*step/total_steps;
    }
    if ( a > 360.0 ) a-= 360.0;
    glRotate(a,tree->rx,tree->ry,tree->rz);
    if ( tree->attrib_function != NULL )
      tree->attrib_function();
    tree->drawing_function();
    animate(tree->child,step,total_steps);
    glPopAttrib();
    glPopMatrix();
    animate(tree->sibling,step,total_steps);
}
```

8. Viewing

Consider an airplane whose position is specified by the roll, pitch, and yaw, and by the distance from an object. Find a model-view matrix in terms of these parameters!
(Hint: It is not required to manually multiply individual matrix elements!)



Solution:
A model-view matrix can be composed from multiple affine transformations, such as rotations and translations. The distance to an object contributes a translation; roll, pitch, and yaw denote rotations around the z, x, and y axes, respectively. The model-view matrix can thus be constructed as

$$M = T(dx, dy, dz) R_x(pitch) R_y(yaw) R_z(roll)$$

Here, (dx,dy,dz) denotes the distance of the airplane from the object.
$T, R_x, R_y, \text{and} R_z$ denote the translation and rotation matrices as introduced in the book by Angel (Chapter 4.7).