Department of Computer Science
Vrije Universiteit
Dr. T. Kielmann, Dr. F. Seinstra

Exam **Computer Graphics**
Exam code: 4001061
15 - 02 - 2010

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 (maximum)

Q	1			2			3					4		5			6			7		8	
	a	b	c	a	b	c	a	b	c	d	e	a	b	a	b	c	a	b	c	a	b	a	b
P	2	3	3	3	2	3	4	4	4	3	3	5	5	3	4	4	3	6	2	5	8	3	8

Total: 90 To pass the exam, it is sufficient to get at least 45 points.

1. Pipeline Architectures

a) The workings of OpenGL and other graphics systems is based on a so-called *"pipeline architecture"*. The pipeline converts vertices into pixels that appear on screen. Name the 4 major parts of the pipeline architecture.

b) A vertex defined using OpenGL typically will go through 5 transformations. These transformations constitute changes in frames (or: coordinate systems). Name the 6 consecutive frames (coordinate systems) used in the pipeline.

c) What is the most important benefit of the pipeline architecture, and for what specific reason does this benefit take effect?

2. Texture Mapping

a) Roughly speaking, texture mapping associates a discrete texel with each point on a geometric object. Name at least 3 problems that can occur with mapping textures, and explain each of the problems in reasonable detail.

b) Mapping of 2-dimensional textures to non-flat 3-dimensional surfaces can be quite complex. Describe a two-step solution that can simplify the mapping.

c) In step two of the solution of part b) above there are many choices possible for the final mapping of texture values onto the target surface. Describe 3 possible approaches.

3. Shearing

Figure 1 shows a cube, as well as a version of the same cube sheared along the x-axis (relative to the y-axis). Figure 2 shows how this shear along the x-axis is characterized, given a single angle θ. It also presents the related shearing matrix.

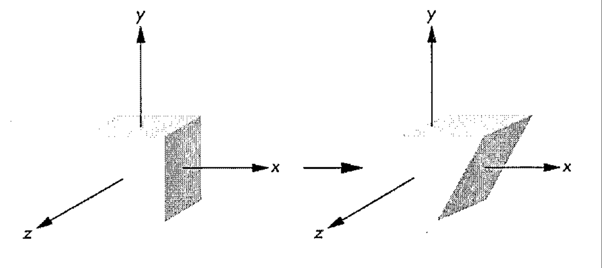
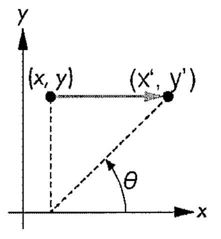


Figure 1



$$H_x(\theta) = \begin{bmatrix} 1 & \cot\theta & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 2

a) Implement a Java method that manipulates OpenGL's Current Transformation Matrix by way of the shearing matrix $H_x(\theta)$:

```
public void shearX_relativetoY(float theta) {
        // Your implementation should follow here.
        // Pure Java is preferred; Java-like or C-like pseudo code is fine, too.
}
```

b) Give pictorial characterizations similar to Figure 2 (left) for shearing along the y-axis (relative to the z-axis), and shearing along the z-axis (relative to the x-axis), using angles α and β respectively.

c) Write down the transformation matrices for the two transformations of part b) above.

d) Write down a single shearing matrix that combines the effects of the three shearings of Figure 2 and question part b) above.

e) Suppose we have implementations available for methods *shearX_relativetoY*, *shearY_relativetoZ*, and *shearZ_relativetoX*: is it possible to implement a single method *shearXYZ_relativetoYZX(theta, alpha, beta)* by a combination of calls to the first three methods? If not: why not? If so: how?

4. Scene Graphs

In a program, scene graphs shall be built from objects of a class `Node`; all specific classes of nodes (geometric objects, transformations, lights, material properties, etc.) are supposed to be subclasses of `Node`. The scene graph shall be organized as a left-child, right-sibling tree. Here is a (partial) implementation of a Java class `Node`:

```
abstract public class Node {
  private Node LeftChild;
  private Node RightSibling;

  public Node(){ setLeftChild(null); setRightSibling(null); }
  public void addChild(Node child){
     child.setRightSibling(getLeftChild())
     setLeftChild(child);
  }
  abstract public void render(GL gl);
  public void traverse(GL gl){ // not shown here
  }
  public void setLeftChild(Node leftChild) { LeftChild = leftChild; }
  public Node getLeftChild() { return LeftChild; }
  public void setRightSibling(Node rightSibling) { RightSibling = rightSibling; }
  public Node getRightSibling() { return RightSibling; }
}
```

a) Implement the method `Traverse` of class `Node`.

b) Here is a (partial) implementation of a Java class `Square`:

```
public class Square extends Node{
  double radius;
  public Square(double radius){ this.radius = radius; }

  public void render(GL gl){
    // renders a square centered at the origin, in the plane z=0
}
```

Implement its `render` method such that the square will be centered at the origin, being in the plane $z = 0$.

Where necessary, use OpenGL calls. Please use Java syntax, or pseudo code close to Java.

5. 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!

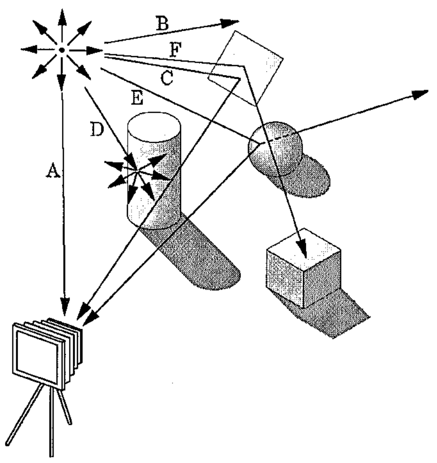
c) Write an implementation in Java-like syntax of the `reshape` method (using OpenGL) that selects the viewport according to part b) above.

```
public void reshape(GLAutoDrawable d, int ww, int wh){
   // Your implementation should follow here.
   // Pure Java is preferred; Java-like or C-like pseudo code is fine, too.
   // Note: 'ww' and 'wh' represent width and height of the display window.
}
```

6. Ray Tracing

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

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?



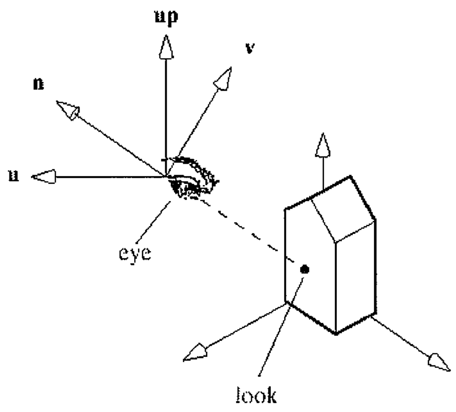
c) What is the most important disadvantage of ray tracing because of which it is not used, for example, by OpenGL?

7. Viewing

a) In the OpenGL rendering pipeline, the Current Transformation Matrix (CTM) consists of two parts. Which are they? What is their respective role for the rendering process? To which of the two should the function `gluLookAt()` be applied to? (say why!)

b) The function `gluLookAt(eyex,eyey,eyez,lookx,looky,lookz,upx,upy,upz)` internally uses a u-v-n viewing 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` first normalizes n, u, v to unit length and then 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 $(0,0,1,0)^T$!

Hint: $\cos 0 = 1$

8. Bresenham's Algorithm

a) What is the fundamental idea behind Bresenham's line drawing algorithm that reduces the necessary computation per pixel?

b) Using Bresenham's algorithm, implement a function `draw_line(x0,y0,x1,y1)` that draws a line from individual pixels, where the starting point is `(x0,y0)` and the end point is `(x1,y1)`. Your function should work for all lines with slope m for which holds $0 \leq m \leq 1$. For drawing an individual pixel `(x,y)`, use the function `void plot(int x, int y)`.

Hint: It is OK to use floating point arithmetic for your function.