Universitetet i Agder
Fakultet for teknologi og realfag

E K S A M E N

Emnekode:	**DAT200**
Emnenavn:	**Grafisk Databehandling**
Dato:	11. desember 2008
Varighet:	0900 - 1300
Antall sider inkl. forside	8
Tillatte hjelpemidler:	Skrivesaker
Merknader:	Oppgavenes vektning er angitt i overskrift på hver oppgave.

OPPGAVE 1. (12%)

NB: Du får + 1 poeng for riktig svar, - ½ poeng for feil svar.
(Skriv besvarelsen inn på eget ark sammen med resten av besvarelsen.)
Angi om du er enig(Ja) eller uenig(Nei) i følgende utsagn:

		Ja	Nei
a)	To rotasjoner i rommet (3D) er alltid kommutative (Det vil si at de er uavhengig av rekkefølgen de utføres i).		
b)	Når vi skal utføre flere transformasjoner etter hverandre er det viktig at vi setter matrisene opp fra venstre mot høyre slik at de blir utført i riktig rekkefølge.		
c)	To translasjoner er alltid kommutative.		
d)	Ved fylling av polygoner med scanlinjealgoritmen behandles horisontale kanter korrekt ved ikke å ta dem med.		
e)	Sutherland Hodgman sin klippingsalgoritme for polygoner anvendes vanligvis i forhold til et klipperektangel, men algoritmen kan også anvendes på en klipperegion i form av et konvekst polygon.		
f)	En perspektivisk projeksjon vil bevare alle parallelle linjer i rommet som parallelle etter projeksjon.		
g)	Ved "displacement-mapping" så vil ikke et objekts silhuett påvirkes.		
h)	En B-spline kjennetegnes ved lokal kontroll.		
i)	I Java kan vi definere våre egne hendelser og få disse lagt inn i hendelseskøen.		
j)	I Java3D forlanges det at transformasjoner over viewplattformen er kongruente (Stivt legeme transformasjoner)		
k)	I Java kan det kun være en lytter til en hendelse.		
l)	Ved bruk av et MouseAdapter objekt i Java kan vi begrense oss til å skrive inn koden for de metodene som skal gjøre noe i MouseListener-"interfacet".		

OPPGAVE 2. TRANSFORMASJONER (16%)

a) De fire matrisene nedenfor representerer en transformasjon av punkter når vi bruker homogene koordinater i 2D:

$$\begin{bmatrix} 3 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} -2 & 0 & 8 \\ 0 & 3 & 6 \\ 0 & 0 & 1 \end{bmatrix}$$

(1) (2) (3) (4)

Beskriv de transformasjonene som hver matrise(1-4) ovenfor representerer. (En transformasjon kan være sammensatt av flere enkle transformasjoner)

b) Hvorfor benyttes homogene transformasjonsmatriser?

c) Vis transformasjonssekvensen av 3x3 homogene transformasjonsmatriser (2D transformasjoner) som transformerer et linjestykke med endepunkter (4,1) og (6,1) til et linjestykke med endepunkter (1,1) og (1,4). Det er ikke nødvendig å multiplisere ut matrisene, men sekvensen skal settes opp i riktig rekkefølge.

OPPGAVE 3. KURVER OG LYSMODELLER (14%)

Gitt følgende kontrollpunkt for en Bezier-kurve:

P1 = (10,10)
P2 = (20,20)
P3 = (30,0)
P4 = (40,10)

a) Tegn den resulterende Bezier-kurve samt kurven sitt tilhørende konvekse hull.

b) Phong sin lysmodell for ett lys er uttrykt ved likningen:

$$I = k_a I_a + k_d I(N \cdot L) + k_s I(V \cdot R)^{ns}$$

1) Forklar hva likningen uttrykker.
2) Forklar hva de tre forskjellige leddene i summasjonen betyr .
3) Skisser de forskjellige vektorene (**N,L,V,R**).

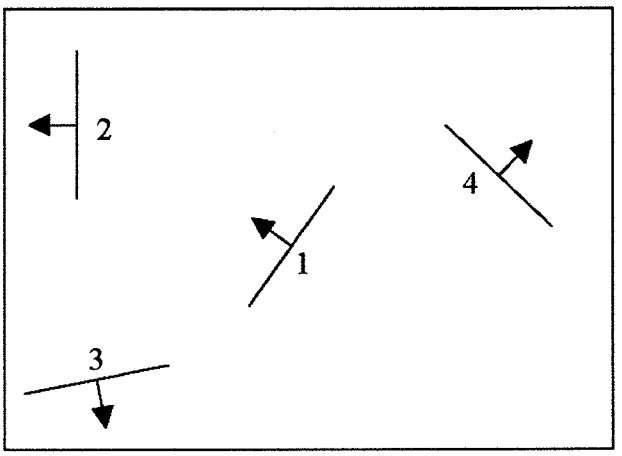
c) Forklar forskjellen på Phong og Gouraud sin lyssettingsmodell.

OPPGAVE 4. SYNLIGE FLATER OG PROJEKSJONER (20%)

a) Forklar hva som menes med "backface culling"(fjerning av bakflater) og hvordan denne algoritmen fungerer.

b) Forklar hvordan Z-buffer algoritmen fungerer.

c) Binary Space Partitioning (BSP-trær) er en algoritme som benyttes i forbindelse med bestemmelse av synlige flater. Tegn et komplett BSP-tre for figuren under(4 linjer nummerert fra 1 til 4). Benytt linje 1 som root.

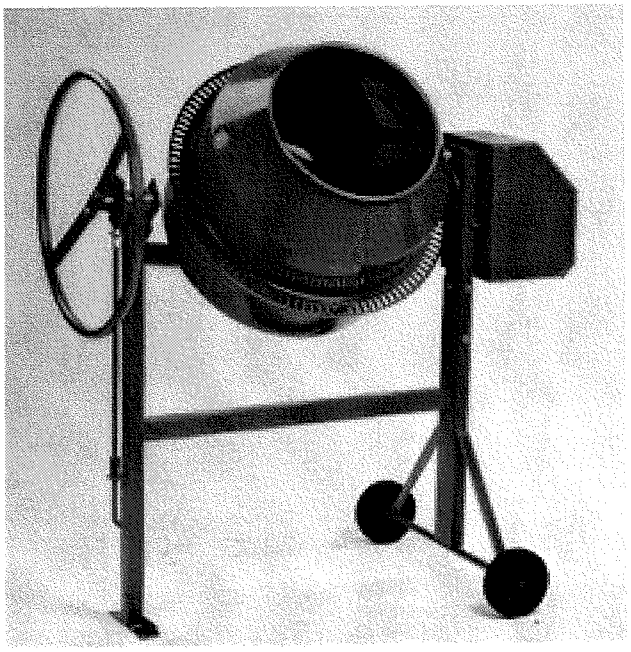


d) Til hvilken klasse projeksjoner hører isometrisk projeksjon og hva kjennetegner denne projeksjonstypen?

e) Hvorfor brukes de parallelle projeksjonene i stor grad innefor modellering?

OPPGAVE 5. 3D-Studio (22%)

Vi skal modellere deler av en blandemaskin som blander sand, vann og sement til betong (Se figur til høyre). Selve blanderen (trommelen) roterer ved hjelp av en motor (boksen til høyre for blanderen) som sørger for blandeeffekten. Et ratt sørger for at vi kan tømme betongen ut av blanderen f. eks oppi en trillebår.



a) Forklar kort og prinsipielt hvordan du vil gå frem i 3D-Studio for å modellere selve beholderen (Ikke bladene inne i beholderen) der betongen skal blandes. Ta dine egne antagelser dersom du er usikker på formen på objektene.

b) Angi hva slags materiale du ville definert i 3D Studio for blandemaskinen.

c) For å rendere en murvegg i en scene kan veggen modelleres som et enkelt teksturert polygon. Skriv kort om fordelene og ulempene med å benytte teksturering istedenfor å modellere murveggen med detaljert geometri.

d) Forklar kort om noen av ulempene fra oppgave c) kan reduseres ved i tillegg å benytte andre tekstureringsmetoder som er nevnt i pensum.

OPPGAVE 6. Java (16%)

Vi skal nå laste inn (deloppgave a nedenfor) og deretter animere (deloppgave b nedenfor) blandemaskinen fra oppgave 5 i Java 3D. Den består av fem deler: en blander (trommel), en ramme (som inkluderer motor), et ratt (for tømming) og to hjul. Blandemaskinen skal kunne rotere blanderen for å blande betongen og blanderen skal kunne tømmes ved å rotere rattet. Ta utgangspunkt i at objektene Blander, Ratt, Ramme og Hjul er modellert i 3Dstudio og lagret som filer med samme navn.

a) Skriv den rutinen i en ny klasse Blandemaskin som setter sammen selve objektet (Tilsvarende public BranchGroup createSceneGraph() i Vindmolle.java). Bruk egne antagelser vedrørende dimensjoner, akseretninger, avstander etc. Du trenger **ikke** å ta hensyn til at blandemaskinen skal bevege(rotere) seg.

b) Vi vil nå ha muligheten for å bruke blandemaskinen og den skal kunne blande betong, tømmes og forflyttes (Det siste forgår ved man løfter maskinen i enden uten hjul og trekker den etter seg). Tegn i diagramsform den BranchGroup, med nødvendige forklaringer, som rutinen (public BranchGroup createSceneGraph()) nå skal returnere.

VEDLEGG 1

```
package Vindmolle;

import java.awt.*;
import java.awt.event.*;
import javax.media.j3d.*;
import javax.vecmath.*;
import javax.swing.*;
import com.mnstarfire.loaders3d.Inspector3DS;


class VindmollePanel extends JPanel implements ActionListener
{
  Button minus = new Button("+");
  Button pluss = new Button("-");
  Tastaturtrykk t;
  Alpha rotationAlpha;

  public VindmollePanel()
  {
    setLayout(new BorderLayout());

    GraphicsConfigTemplate3D template = new GraphicsConfigTemplate3D();
    template.setSceneAntialiasing(GraphicsConfigTemplate3D.REQUIRED);

    // Get the GraphicsConfiguration that best fits our needs.
    GraphicsConfiguration gcfg =
    GraphicsEnvironment.getLocalGraphicsEnvironment().
    getDefaultScreenDevice().getBestConfiguration(template);

    Canvas3D c = new Canvas3D(gcfg);
    add("Center", c);
    Panel p =new Panel();

    p.add(minus);
    p.add(pluss);
    add("North",p);

    pluss.addActionListener(this);
    minus.addActionListener(this);

    // Create a simple scene and attach it to the virtual
    // universe

    BranchGroup scene = createSceneGraph();
    UniverseBuilder u = new UniverseBuilder(c);
    u.addBranchGraph(scene);
  }

public BranchGroup createSceneGraph() {

    // Create the root of the branch graph
    BranchGroup objRoot = new BranchGroup();


    // Create the TransformGroup node and initialize it to the
    // identity. Enable the TRANSFORM_WRITE capability so that
    // our behavior code can modify it at run time. Add it to
    // the root of the subgraph.

    TransformGroup TGBlad1   = new TransformGroup();
    TransformGroup TGBlad2   = new TransformGroup();
    TransformGroup TGRotator = new TransformGroup();


    TGRotator.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);

    objRoot.addChild(TGRotator);

    // Add the fundament and the base in the scene graph

    Inspector3DS loader = new Inspector3DS("c:/temp/Vindmolle/fundament.3ds"); // constructor
    loader.parseIt(); // process the file
    TransformGroup fundament = loader.getModel();
    objRoot.addChild(fundament);
    // get the resulting 3D model as a Transform Group with Shape3Ds as children


    // Create a new Behavior object that will perform the
    // desired operation on the specified transform and add
    // it into the scene graph.

    Transform3D zAxis = new Transform3D();
    zAxis.rotX(Math.PI/2);
    rotationAlpha = new Alpha( -1, Alpha.INCREASING_ENABLE,0, 0,
                    4000, 0, 0, 0, 0, 0);
    RotationInterpolator rotator = new RotationInterpolator(
      rotationAlpha, TGRotator, zAxis, 0.0f, (float) Math.PI*2.0f);
    BoundingSphere bounds = new BoundingSphere(new Point3d(0.0,0.0,0.0),200.0);
    rotator.setSchedulingBounds(bounds);
    TGRotator.addChild(rotator);

    // Add a Behavior that accepts keyboard input
    Tastaturtrykk t = new Tastaturtrykk(rotationAlpha);
    TGRotator.addChild(t);

    // Hent inn bladene
    Inspector3DS loader2 = new Inspector3DS("c:/temp/Vindmolle/blad.3ds"); // constructor
    loader2.parseIt(); // process the file
    TransformGroup blad1 = loader2.getModel();

    Inspector3DS loader3 = new Inspector3DS("c:/temp/Vindmolle/blad.3ds"); // constructor
    loader3.parseIt(); // process the file
```

```
    TransformGroup blad2 = loader3.getModel();

    Inspector3DS loader4 = new Inspector3DS("c:/temp/Vindmolle/blad.3ds"); // constructor
    loader4.parseIt(); // process the file
    TransformGroup blad3 = loader4.getModel();

    TGRotator.addChild(blad1);

    // Add the blades and rotate them
    Transform3D zAxis2 = new Transform3D();

    zAxis2.rotX(Math.PI/2);
    zAxis2.rotZ(2.0*Math.PI/3);

    TGBlad1.setTransform(zAxis2);
    TGBlad1.addChild(blad2);

    TGBlad2.setTransform(zAxis2);
    TGBlad2.addChild(blad3);

    TGBlad2.addChild(TGBlad1);
    TGRotator.addChild(TGBlad2);

    return objRoot;
  }

   public void actionPerformed(ActionEvent e )
   {
     long oldvalue= rotationAlpha.getIncreasingAlphaDuration();
     String kommando=e.getActionCommand();
     if (kommando=="+")
     {
       rotationAlpha.setIncreasingAlphaDuration(oldvalue/2);
     }
     else if (kommando=="-")
     {
       rotationAlpha.setIncreasingAlphaDuration(oldvalue*2);
     }
   } // End actionPerformed
}

class VindmolleFrame extends JFrame
{
  public VindmolleFrame()
  {
    addWindowListener(new WindowAdapter()
      {  public void windowClosing(WindowEvent e)
      { System.exit(0); }
    } );

    setSize(400, 400);
    setTitle(getClass().getName());

    Container contentPane = getContentPane();
    contentPane.add(new VindmollePanel());

  }
}

public class Vindmolle
{

  public static void main(String args[])
  {
    JFrame f = new VindmolleFrame();
    f.setSize(500,500);
    f.show();
  }
}

package Vindmolle;

import java.awt.*;
import java.awt.event.*;
import javax.media.j3d.*;
import javax.vecmath.*;

public class UniverseBuilder extends Object {

    // User-specified canvas

    Canvas3D canvas;

    // Scene graph elements to which the user may want access

    VirtualUniverse      universe;
    Locale               locale;
    TransformGroup       vpTrans;
    View                 view;

    public UniverseBuilder(Canvas3D c) {
      this.canvas = c;


      // Establish a virtual universe that has a single
      // hi-res Locale

      universe = new VirtualUniverse();
      locale = new Locale(universe);


      // Create a PhysicalBody and PhysicalEnvironment object

      PhysicalBody body = new PhysicalBody();
      PhysicalEnvironment environment =
                              new PhysicalEnvironment();
```

```
        // Create a View and attach the Canvas3D and the physical
        // body and environment to the view.

        view = new View();
        view.addCanvas3D(c);
        view.setPhysicalBody(body);
        view.setPhysicalEnvironment(environment);
        view.setBackClipDistance(500);


        // Create a BranchGroup node for the view platform

        BranchGroup vpRoot = new BranchGroup();

        // Create a ViewPlatform object, and its associated
        // TransformGroup object, and attach it to the root of the
        // subgraph. Attach the view to the view platform.

        Transform3D t = new Transform3D();
        Transform3D s = new Transform3D();

        t.rotY(Math.PI/4);
        s.set(new Vector3f(0.0f, 0.0f, 200.0f));
        t.mul(s);
        s.rotX(-Math.PI/32);
        t.mul(s);

        ViewPlatform vp = new ViewPlatform();
        vpTrans = new TransformGroup(t);
        vpTrans.addChild(vp);
        vpRoot.addChild(vpTrans);
        view.attachViewPlatform(vp);

        // Attach the branch graph to the universe, via the
        // Locale. The scene graph is now live!

       locale.addBranchGraph(vpRoot);
    }

    public void addBranchGraph(BranchGroup bg) {
       locale.addBranchGraph(bg);
    }
  }

package Vindmolle;

import java.awt.*;
import java.awt.event.*;
import java.util.*;
import javax.media.j3d.*;
import javax.vecmath.*;

public class Tastaturtrykk extends Behavior
{
 Alpha alpha;
 WakeupCriterion[] keyEvents;
 WakeupOr keyCriterion;

 public Tastaturtrykk(Alpha alpha)
 {
  this.alpha=alpha;
  BoundingSphere bounds = new BoundingSphere(new Point3d(0.0,0.0,0.0), 200.0);
  this.setSchedulingBounds(bounds);
 }

 public void initialize()
 {
  keyEvents = new WakeupCriterion[1];
  keyEvents[0]=new WakeupOnAWTEvent(KeyEvent.KEY_PRESSED);
  keyCriterion = new WakeupOr(keyEvents);
  wakeupOn (keyCriterion);
 }

 public void processStimulus (Enumeration criteria)
 {
  WakeupCriterion wakeup;
  AWTEvent[] event;
  int id;
  char k;

  while (criteria.hasMoreElements()) {
   wakeup = (WakeupCriterion) criteria.nextElement();
   if (wakeup instanceof WakeupOnAWTEvent) {
    event = ((WakeupOnAWTEvent)wakeup).getAWTEvent();
    for (int i=0; i<event.length; i++) {
     id = event[i].getID();
     if (id == KeyEvent.KEY_PRESSED) {
      k = ((KeyEvent)event[i]).getKeyChar();
      long oldvalue= alpha.getIncreasingAlphaDuration();
      if (k=='+')
       {
        alpha.setIncreasingAlphaDuration(oldvalue/2);
        System.out.println("+");
       }
      else if (k=='-')
       {
        alpha.setIncreasingAlphaDuration(oldvalue*2);
        System.out.println("-");
       }
     } // End if
    } // End for
   } // End if
  } // End while
  wakeupOn (keyCriterion);
 } // End processStimulus
} // End class Tastaturtrykk
```