Universitetet i Agder
Fakultet for fakultet for Teknologi og realfag

EKSAMEN

Emnekode:	**DAT200**
Emnenavn:	**Grafisk Databehandling**
Dato:	10. mai 2013
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)	"Raster Scan" skjermen er den mest utbredte skjermtypen i dag grunnet sin gode evne til å tegne fylte områder.		
b)	En skalering og en rotasjon i planet (2D) er alltid kommutative (Det vil si at rekkefølgen er uvesentlig).		
c)	Sutherland Hodgman sin klippingsalgoritme kan kun anvendes for konvekse polygoner.		
d)	To translasjoner i rommet (3D) er alltid kommutative.		
e)	Et mindre vindu vil medføre at detaljer trer klarere frem i en viewport (skjermport) med fast størrelse.		
f)	Perspektiviske transformasjoner etterligner virkemåten til det menneskelige øyet.		
g)	Et matt materiale har speilende ("Specular") refleksjon over at større område enn et blankt materiale.		
h)	En isometrisk projeksjon har projeksjonsstråler som står normalt på projeksjonsplanet.		
i)	Ved "bump-mapping" så vil ikke et objekts silhuett påvirkes.		
j)	En Bezier-spline oppfyller det vi på engelsk kaller "convex hull property"		
k)	"Antialiasing" er en teknikk for å gjengi fylte flater på en mest mulig virkelighetsnær måte.		
l)	Painters algoritme (Malerens algoritme) er en algoritme for fylling av polygoner.		

OPPGAVE 2. TRANSFORMASJONER (14%)

a) De tre matrisene nedenfor representerer en transformasjon av punkter når vi bruker homogene koordinater:

$$\begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} S & 0 & 0 & 0 \\ 0 & S & 0 & 0 \\ 0 & 0 & S & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(1) (2) (3)

Beskriv de transformasjonene som hver matrise(1-3) ovenfor representerer.

b) Hvorfor benyttes homogene transformasjonsmatriser?

c) Vis transformasjonssekvensen av 3x3 homogene transformasjonsmatriser (2D transformasjoner) som transformerer et linjestykke med endepunkter (8,1) og (12,1) til et linjestykke med endepunkter (2,2) og (2,4). Det er ikke nødvendig å multiplisere ut matrisene, men sekvensen skal settes opp i riktig rekkefølge.

OPPGAVE 3. FARGER, SKULTE FLATER OG INTENSITETER (12%)

a) Hva er halvtoning og hvor ser vi den daglige anvendelsen av denne teknikken?

b) Angi kort prinsippet for z-buffer algoritmen for fjerning av flater. Er dette en objekt-presisjons algoritme og/eller en bilde-presisjons algoritme?

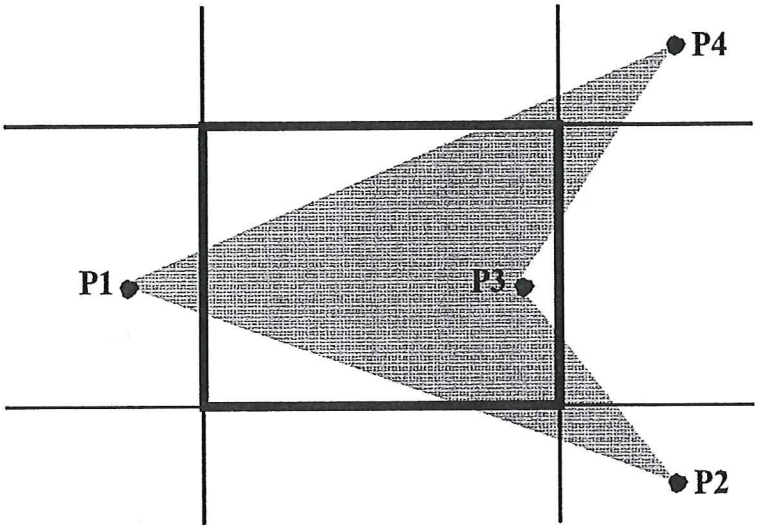
c) Hvorfor gjengir PHONG-"shading" speilende refleksjon("specular reflection") på en bedre måte enn Gouraud-"shading"?

OPPGAVE 4. KLIPPING, PROJEKSJONER, KURVER OG SOLIDER (24%)

Sutherland og Hodgman's polygonklippingsalgoritme benytter seg av en "splitt og hersk" strategi.

a) Forklar kort grunnprinsippet bak denne algoritmen.

Gitt følgende polygon:



b) Vis hvordan Sutherland og Hodgman's polygonklippingsalgoritme benyttes for å klippe polygonet mot angitt klippevindu. Gjør spesielt rede for hvor og hvordan skjæringspunkt mellom polygonet og klippekantene oppstår.

c) Forklar i hvilke sammenhenger du vil anvende henholdsvis parallell og perspektivisk projeksjon.

d) Bezier-kurver har den egenskap at en hver del av et kurvesegment ligger innenfor det konvekse omhyllingslegemet definert av de fire styrepunktene til kurvesegmentet. Hvorfor er dette tilfelle? Finnes det andre typer kubiske parametriske kurver som også har denne egenskapen? Hvordan kan denne egenskapen utnyttes i en linjeklippingsalgoritme for å gjøre den mer effektiv?

e) Naturlige kubiske splines har C^0, C^1 og C^2 kontinuitet, noe som gjør dem glatter enn hermite-kurver og kubiske bezier-kurver. Dessuten interpolerer de kontrollpunktene sine, noe som gjør dem enklere å styre enn f.eks. B-Splines. Likevel brukes disse i liten grad i f.eks. profesjonelle DAK-systemer. Hvorfor?

f) Hvordan defineres et "Binary Space-Partitioning Tree"? Kan du nevne noen fordeler/ulemper med denne representasjonsformen for solidmodeller.

OPPGAVE 5. 3D-Studio (22%)

Vi skal modellere deler av en snøfrese-maskin (Se figur til høyre). Selve freseren fungerer ved at skovlene foran i huset roterer om sin akse og dytter snøen inn til en utkaster som dytter snøen videre ut gjennom toppen av maskinen. Fresen har to hjul og diverse håndtak etc. for å styre motor, maskin og utkastretning som kan varieres.

a) Forklar kort og prinsipielt hvordan du vil gå frem i 3D-Studio for å modellere ett av hjulene på snøfreseren. Ta dine egne antagelser dersom du er usikker på formen på objektene.

b) Angi hva slags materiale du ville definert i 3D Studio for hjulet (dekk og felg).

c) For å få mønster på hjulet kan vi bruke eksakt modellering aller "bump mapping". Angi fordeler og ulemper ved de to metodene.

OPPGAVE 6. Java (16%)

Vi skal nå laste inn (deloppgave a nedenfor) og deretter animere (deloppgave b nedenfor) snøfresmaskinen fra oppgave 5 i Java 3D. Den består av fem deler: skovlene, karosseriet (som inkluderer motor og hus med tilbehør), to hjul og en utkaster (som er dreibar). Fresen skal kunne bevege seg forover, bakover og kunne endre retning. I tillegg skal hele maskinen også kunne roteres slik at hus med skovler kommmer opp i fra bakken. Utkasteren skal også kunne roteres slik at kasteretning kan justeres. I vår oppgave skal vi se bort fra spaker etc. Ta utgangspunkt i at objektene Karosseri, Hjul, Skovler og Utkaster er modellert i 3Dstudio og lagret som filer med samme navn.

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

b) Vi vil nå ha muligheten for å bruke fresemaskinen og den skal kunne bevege seg , skovlene skal kunne rotere og utkastretning skal kunne justeres. 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
```