```
void storleksbyte(int width, int height) {
    glViewport(0, 0, width, height);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-0.5,3.0,-0.5,3.0,-1.0,1.0);
}

void tangenthanterare(unsigned char key, int x, int y) {
   if (key == 'm') A = A + 3.0;
   if (key == 'n') A = A - 3.0;
   if (key == 'k') B1 = B1 - 3.0;
   if (key == 'l') B1 = B1 + 3.0;
   ...
   if (key == 'q') exit(0);
   glutPostRedisplay();
}

void myMenu(int item) {
   switch(item) {
       case 1: tangenthanterare('L',0,0); break;
       case 2: tangenthanterare('F',0,0); break;
       case 9: exit(0); break;
   }
}

int main(int argc, char** argv)  {
    // Dubbla bildminnen vid all rörelse
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(400,400);
    glutCreateWindow("Rotera med m,n,k,l,o,p,1,2,3,4");
    printf("Växla ritsätt med F, L och P\n");
    glutReshapeFunc (storleksbyte);
    glutDisplayFunc(rita);
    glutKeyboardFunc(tangenthanterare);
    glutCreateMenu(myMenu);
        glutAddMenuEntry("Avsluta",9);
    glutAttachMenu(GLUT_RIGHT_BUTTON);
    glutMainLoop();
}
```


Ett mellanspel - polygonmoder

Normalt ritas ytor fyllda. Men med anrop av typen

`glPolygonMode(GL_FRONT_AND_BACK, ritsätt);`

där ritsätt är `GL_POINT` (hörnpunkterna), `GL_LINE` (kanterna) eller `GL_FILL` (standardbeteendet) kan vi få ritning av enbart hörnpunkterna, enbart kanterna respektive fyllnad. Och det går att kombinera (vilket för rätt effekt kräver vissa extra åtgärder).

I förra programmet finns i (den fullständiga) tangenthanteraren:

```
   if (key == 'F')
         glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
   if (key == 'L')
         glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
   if (key == 'P') {
         glPolygonMode(GL_FRONT_AND_BACK, GL_POINT);
         glPointSize(4.0);
   }
```

I menyuppbyggnaden finns

```
glutAddMenuEntry("Linjemod",1);
glutAddMenuEntry("Fyllmod",2);
```

varför vi i menyhanteraren kan ta till

```
void myMenu(int item) {
   switch(item) {
       case 1: tangenthanterare('L',0,0); break;
       case 2: tangenthanterare('F',0,0); break;
       case 9: exit(0); break;
   }
}
```

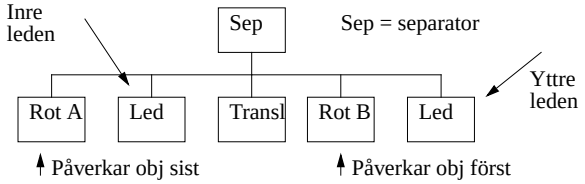

Scengrafer 1(3)

Scengrafer har huvudsakligen följande användningar:

- **Strukturering av bl a hierarkiska modeller**
- **Strukturering av modeller för optimering och selektering**

En scengraf är ett allmänt träd (eller en graf om vi återutnyttjar vissa objekt) med alla scenens objekt. Till objekten räknas de geometriska objekten men även transformationer och material. Och mycket annat som vi i varje fall inte just nu går in på. Man skapar objekten och lägger successivt in dem i grafen (objekten kallas därefter noder) utan att rita något. Ritning görs i ett separat steg utifrån scengrafen. Man kan när som helst ändra en egenskap hos ett objekt/nod. Om vi t ex ändrar en vinkel för en rotationsnod kan vi efter ny uppritning få rörelse.

Vi belyser nu scengrafer i anslutning till förra exemplet. Först bara två leder. Objektet Led är i det allmänna fallet i sig ett sammansatt objekt.



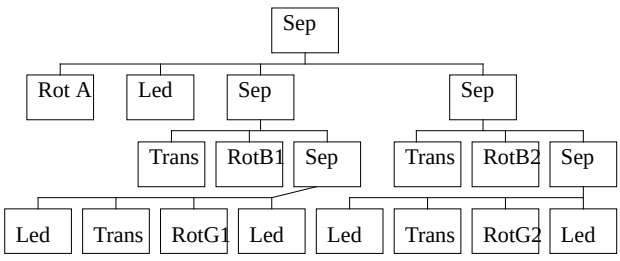
Transformationer ackumuleras precis som i OpenGL. Tillstånd ersätter tidigare gällande tillstånd. Scengrafen genomlöpes enligt "djupet först" och från vänster till höger. Inplaceringsordningen är således betydelsefull.

Scengrafer 2(3)

OpenGL har inte scengrafer men vi kan simulera dem på låg nivå (med hjälp av bl a *glPushMatrix*/*glPopMatrix*) eller på högre nivå genom att införa egna verktyg (träd eller grafer och listor). Open Inventor och Java 3D har scengrafer och hjälpmedel för dem inbyggda. VRML (Virtual Reality Modelling Language) och dess efterföljare har scengrafer som en ingrediens.

Separationsnoder (Sep i figurerna) gör att alla förändringar under noden förblir lokala (kodas i OpenGL med *glPushMatrix*/*glPopMatrix*). Tillstånds- och transformationsändringar till vänster om och på samma nivå som separationsnoden ärvs. **Gruppnoder** - som vi inte exemplifierat - innebär att det som görs under noden påverkar till höger om noden.

Till slut en scengraf för fallet med 5 leder fördelade enligt tidigare figur:



Scengrafer 3(3)

Open Inventor är en produkt som ursprungligen togs fram av Silicon Graphics. SGI släppte för ett par år sedan källkoden vilket lett till att det finns ett par fria versioner. Se kursens webbsida för adresser.

Linux: En av dessa (fr o m ht 2005 FLTK:s) har jag fått till att fungera tillsammans med g++ under Linux. Används så här. Säg att källkodsfilen heter `test.c++`. Då kan man kompilera och länka med (som vanligt har jag gjort en kommandoprocedur, som den intresserade kan inspektera)
`INV_COMPILE test`
varvid det bildas en **stor** körbar fil test, som enklast körs med
`INV_RUN test`
I mappen `/users/course/TDA360/LINUX/FLINV` finns ett par exempel. I `/users/course/TDA360/LINUX/FLINV/fl-inventor/install/bin/unix` många fler.
Windows: En av mig installerad version går att använda direkt ihop med MSVC++. Man måste se till `Z:\DGKURSEN\PC\INVPC\VC98` räknas som inkluderingsmapp och att `Z:\DGKURSEN\PC\glut32.lib` samt `Z:\DGKURSEN\PC\INVPC\inventor.lib` länkas med. Vidare måste `inventor.dll` nås via PATH (t ex i aktuell mapp).

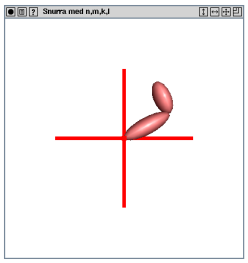
Open Inventor utvecklas kommersiellt av företaget TGS (nu uppköpt av Mercury).

Andra scengraf-system ovanpå OpenGL, som finns (i princip) för åtminstone Windows och Linux (se webbsidan för adresser):

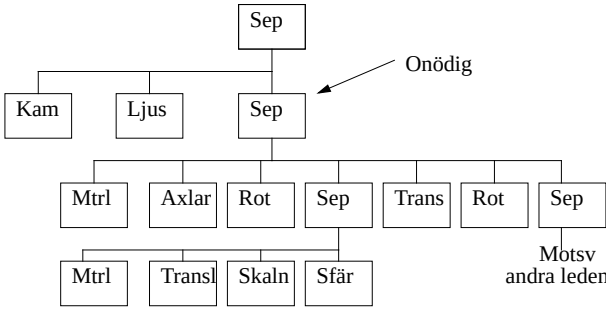
- **OpenScenegraph**
- **NVIDIA SG**
- **Gizmo (från Saab Training Systems AB)**

Open Inventor 1(6)

Vi belyser nu scengrafer med hjälp av Open Inventor. Materialet här enbart med för illustrationens skull. Tyvärr blir även det enklaste program rätt skrymmande. Men jag hoppas att när nu de flesta läst ett objektorienterat språk som Java, skall helhetsideerna framgå trots de många detaljerna. Vi nöjer oss med fallet med två leder. Vi behöver då



bygga en scengraf med följande struktur (vissa extra finesser):



Open Inventor 2(6)

Kommentarerna till programkoden får bli huvudsakligen muntliga.

```
> cat INV_2003.c++

// Först måste man inkludera filer för varje klass
#include <Inventor/SoDB.h>
#include <Inventor/SoSceneManager.h>
#include <Inventor/nodes/SoDirectionalLight.h>
#include <Inventor/nodes/SoPerspectiveCamera.h>
#include <Inventor/nodes/SoSeparator.h>
#include <Inventor/nodes/SoMaterial.h>
#include <Inventor/nodes/SoSphere.h>
#include <Inventor/nodes/SoRotationXYZ.h>
#include <Inventor/nodes/SoTransform.h>
#include <Inventor/actions/SoWriteAction.h>
...
#include <GL/glut.h>
#include <iostream>
using namespace std;

// Globala objekt m m
SoSceneManager *scenemanager;
SoSeparator *root ;
SoPerspectiveCamera *perspCamera;
SoRotationXYZ *rotation = new SoRotationXYZ;
SoRotationXYZ *rotationB = new SoRotationXYZ;
double A=3.0, B=1.5; //vinklar vid start
double Z = 16.0;

// Redraw on scenegraph changes.
void redraw(void * user, SoSceneManager * manager) {
     glEnable(GL_DEPTH_TEST); glEnable(GL_LIGHTING);
     manager->render();
     glutSwapBuffers();
}
```


Open Inventor 3(6)

```
// Vanliga omritaren
void rita(void) {
     glEnable(GL_DEPTH_TEST);  glEnable(GL_LIGHTING);
     scenemanager->render();
     glutSwapBuffers();
}

void storleksbyte(int w, int h) {
    scenemanager->setWindowSize(SbVec2s(w, h));
    scenemanager->setSize(SbVec2s(w, h));
    scenemanager->setViewportRegion(SbViewportRegion(w, h));
    scenemanager->scheduleRedraw();
}

// Måste vara med?
void idle(void) {
     SoDB::getSensorManager()->processTimerQueue();
     SoDB::getSensorManager()->processDelayQueue(TRUE);
}

SoSeparator* enLed(double length) {
    SoSeparator *led = new SoSeparator;
    SoTransform *translation = new SoTransform;
    translation ->translation.setValue(length/2, 0, 0);
    SoTransform *skalning = new SoTransform;
    skalning ->scaleFactor.setValue(length/2, 0.5, 1);
    led->addChild(translation);
    led->addChild(skalning);
    SoMaterial *mtrl = new SoMaterial;
    mtrl->ambientColor.setValue(1.0, .2, .2);
    mtrl->diffuseColor.setValue(1.0, .6, .6);
    mtrl->specularColor.setValue(1.0, .5, .5);
    mtrl->shininess = .5;
    led->addChild(mtrl);
    SoSphere *s = new SoSphere;
    led->addChild(s);
    return led;
}
```

Open Inventor 4(6)

```
SoCube* axel(double xL, double yL, double zL) {
        SoCube *x = new SoCube;
        x->width = xL; x->height = yL; x->depth = zL;
        return x;
}

SoSeparator* roboten() {
    // En robot med två leder och koordinataxlar
    SoMaterial *axelmtrl = new SoMaterial;
    axelmtrl->ambientColor.setValue(1.0, 0, 0.0);
    axelmtrl->diffuseColor.setValue(1.0, 0.0, 0.0);
    axelmtrl->specularColor.setValue(1.0, 0.0, 0.0);
    axelmtrl->shininess = .5;

    SoSeparator *robot = new SoSeparator;
    rotation->axis = SoRotationXYZ::Z;
    rotation->angle = A;
    robot->addChild(axelmtrl);
    robot->addChild(axel(8.0, 0.2, 0.2));
    robot->addChild(axel(0.2,8.0,0.2));
    robot->addChild(axel(0.2,0.2,8.0));
    robot->addChild(rotation);
    robot->addChild(enLed(3.0));
    SoTransform *translationB = new SoTransform;
    translationB->translation.setValue(3, 0, 0);
    robot->addChild(translationB);
    rotationB->axis = SoRotationXYZ::Z;
    rotationB->angle = B;
    robot->addChild(rotationB);
    robot->addChild(enLed(2.0));
    return robot;
}
```


Open Inventor 5(6)

```
void tangentbord(unsigned char key, int x, int y) {
        SoWriteAction writeAction;
        switch(key) {
            case 'm': // Radianer (grader i OpenGL)
                      A=A+0.03; rotation->angle = A; break;
            case 'n':A=A-0.03; rotation->angle = A; break;
            case 'k':B=B+0.03; rotationB->angle = B; break;
            case 'l':B=B-0.03; rotationB->angle = B; break;
            case 'p': writeAction.apply(root); break;
            case '+': Z=Z+1;
                      perspCamera->position.setValue(0,0,Z);
                      break;
            case '-': Z=Z-1;
                      perspCamera->position.setValue(0,0,Z);
                      break;
            case 27: exit(0); // ESC
            default: break ;
        }
        scenemanager->scheduleRedraw();
}

void menu(int value) {
  // Vi samlar allt i tangentbord i stället
  tangentbord((unsigned char) value, 0, 0);
}

void initMenu() {
   glutCreateMenu(menu);
      glutAddMenuEntry("Skriv ut scengrafen",'p');
      glutAddMenuEntry("Avsluta", 27);
   glutAttachMenu(GLUT_RIGHT_BUTTON);
}
```


Open Inventor 6(6)

```
int main(int argc, char ** argv) {
  SoDB::init();
  printf("Open Inventor version: %s\n",SoDB::getVersion()) ;
  // Scengrafens rotnod
  root = new SoSeparator; root->ref();
  // En kamera
  perspCamera = new SoPerspectiveCamera;
  root->addChild(perspCamera);
  // Ljus
  root->addChild(new SoDirectionalLight);
  // Roboten
  SoSeparator *temp = roboten();
  if (temp) root->addChild(temp) ;
  // Initiera en scenhanterare
  scenemanager = new SoSceneManager;
  scenemanager->setRenderCallback(redraw, NULL);
  scenemanager->setBackgroundColor(SbColor(1.0, 1.0, 1.0));
  scenemanager->activate();
  scenemanager->setSceneGraph(root);
  // Vi vill se hela scenen
  perspCamera->viewAll(root,
                scenemanager->getViewportRegion());
  // GLUT
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA | GLUT_DEPTH);
  glutInitWindowSize(400, 400);
  glutCreateWindow("Snurra med n,m,k,l");
  glutDisplayFunc(rita);
  glutReshapeFunc(storleksbyte);
  glutIdleFunc(idle); //Verkar nödvändig
  glutKeyboardFunc(tangentbord); initMenu();
  glutMainLoop();
  // Snarare på annan plats
  root->unref();
  delete scenemanager;
  return 0;
}
```


Open Inventor - Scengraf utskriven 1(2)

Scengrafen kan skrivas ut enkelt (och även läsas in). Värden skrivs inte ut om de är standard.

```
#Inventor V2.1 ascii
Separator {
    PerspectiveCamera {
        position        0 0 16.7262
        nearDistance    9.79796
        farDistance     23.6544
        focalDistance   16.7262
    }
    DirectionalLight {
    }
    Separator {
        # Här kommer koordinataxlarna; jag hoppar över en del
        Material {
            ambientColor        1 0 0
            ...
        }
        Cube {
            width       8
            height      0.2
            depth       0.2
        }
        Cube {...}
        Cube {...}
        RotationXYZ {
            axis        Z
            angle       6.78
        }
```

Open Inventor - Scengraf utskriven 2(2)

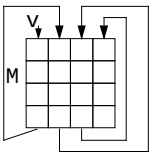
```
        # Här kommer ellipsoiden
        Separator {
            Transform {
                translation     1.5 0 0
            }
            Transform {
                scaleFactor     1.5 0.5 1
            }
            Material {
                ambientColor    1 0.2 0.2
                diffuseColor    1 0.6 0.6
                specularColor   1 0.5 0.5
                shininess       0.5
            }
            Sphere {
            }
        }
        Transform {
            translation 3 0 0
        }
        RotationXYZ {
            axis        Z
            angle       1.5
        }
        # Den andra ellipsoiden
        Separator {...}
    }
}
```


OpenGL och 3D-grafik: Avläs matriserna 1(2)
(se även "Från värld till skärm", avsnitt 5)

Man kan avläsa modellvy- och projektionsmatriserna!

```
> cat Matriskoll.c
// Diverse include
void CheckMatrix(GLenum vilken_matris) {
    GLfloat v[16]; // double duger även
    int i;
    glGetFloatv(vilken_matris,v);
    // glGetDoublev finns också

    if (vilken_matris==GL_PROJECTION_MATRIX) {
       printf("Utskrift av projektionsmatrisen\n");
    } else printf("Utskrift av modellvy-matrisen\n");
    for (i=0; i<4; i++) {
      printf("%f %f %f %f\n",v[i], v[i+4], v[i+8], v[i+12]);
    }
}
void myReshape(int width, int height) {
    glViewport(0, 0, width, height);
    glMatrixMode(GL_PROJECTION); glLoadIdentity();
    glOrtho(0,width,0,width,-1,1);
    glMatrixMode(GL_MODELVIEW);glLoadIdentity();
}
void display(void) {
    glRotated(45,0,0,1); // Ackumuleras!!!!!!!!!!!!!!!!
    CheckMatrix(GL_MODELVIEW_MATRIX);
    CheckMatrix(GL_PROJECTION_MATRIX);
}
int main(int argc, char** argv) {
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(400,400);
    glutCreateWindow("Matriskontroll");
    glutReshapeFunc (myReshape); glutDisplayFunc(display);
    glutMainLoop();
}
```



OpenGL och 3D-grafik: Avläs matriserna 2(2)

Kompilering och körning på PC.

```
>gcc -o Matriskoll.exe Matriskoll.c -lglut32 -lglu32
     -lopengl32 -luser32 -lgdi32
>Matriskoll
Uppdatering                          // första gången
Utskrift av modellvy-matrisen
0.707107 -0.707107 0.000000 0.000000     // 45°
0.707107 0.707107 0.000000 0.000000
0.000000 0.000000 1.000000 0.000000
0.000000 0.000000 0.000000 1.000000
Utskrift av projektionsmatrisen
0.005000 0.000000 0.000000 -1.000000     // Ortografisk
0.000000 0.005000 0.000000 -1.000000
0.000000 0.000000 -1.000000 0.000000
0.000000 0.000000 0.000000 1.000000
Uppdatering                          // framtvingad andra gång
Utskrift av modellvy-matrisen
0.000000 -1.000000 0.000000 0.000000     // Nu 90°
1.000000 0.000000 0.000000 0.000000
0.000000 0.000000 1.000000 0.000000
0.000000 0.000000 0.000000 1.000000
Utskrift av projektionsmatrisen
0.005000 0.000000 0.000000 -1.000000     // Oförändrad
0.000000 0.005000 0.000000 -1.000000
0.000000 0.000000 -1.000000 0.000000
0.000000 0.000000 0.000000 1.000000
```

Stämmer med teorin!

Animering (OpenGL-häftet avsnitt 9)

Vi vill visa upp en scen - förmodligen successivt förändrad - gång på gång.

- **Dubbelbuffra**, dvs använd båda bildminnena.
  Rita i det som inte visas. Begärs enligt modellen
  `glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA);`
  Växla bildminne i slutet av omritningsproceduren med
  `glutSwapBuffers();`
  JOGL: Automatiskt.
- Låt **all ritning ske i omritningsproceduren**
  Anropa den aldrig direkt utan skapa en omritningshändelse med
  `glutPostRedisplay();`
  JOGL: Automatiskt med den programvara vi utgår från.
- Gärna **tidsstyrd omritningshastighe**t
  Se avsnitt 17.
  JOGL: Borde kunna styras med metod i klassen Animator, men jag hittar ingen sådan. Ett sätt är att lägga tidskontrollen i *display*.
- **Tangentbordet är sällan fullgott för styrning** på en dator med goda grafikprestanda
  P g a låg repetitionshastighet (som nog kan ändras) hos tangenterna. I stället kan man använda dem som enbart tillståndsändrare, t ex framåt <-> bakåt.

Hur styrningen av förändringarna skall gå till tar vi upp i samband med navigering efter att vi "löst" problemet med dolda ytor.