UNIVERSITATEA “POLITEHNICA” DIN TIMISOARA
Facultatea de Automatică şi Calculatoare
Departamentul de Automatică şi Informatică Industrială

Nr. Contract: 33501
Nr. Temă: 2
Cod CNCSIS: 46

STUDIUL, ANALIZA ŞI CONDUCEREA UNUI DISPOZITIV HAPTIC FOLOSIND UN MEDIU VIRTUAL BAZAT PE OPEN GL

Director de proiect:
S.l. ing. Florin Drăgan

-2003-

Cuprins

1. Introducere

Scopul acestei lucrari este sa prezinte proiectarea, implemetarea si utilizarea unei camere virtuale pentru studiul miscarii unui dispozitiv haptic. In principal acest proiect se incadreaza in directiile de cercetare aferente domeniului realitatii virtuale.

Termenul de realitate virtuala se refera la starea in care utilizatorul este complet captivat de o "lume" complet generata de calculator. De-a lungul timpului au fost utilizate mai multe dispozitive de simulare a mediului inconjurator.

Una dintre primele incercari de generare a unui mediu virtual a constat in utilizarea unei casti purtate pe cap de catre utilizator. O astfel de casca contine, in mod uzual, doua ecrane si un sistem optic ce prezinta o perspectiva stereo a lumii virtuale generate de calculator. Un dispozitiv de urmarire permite masurarea continua a pozitiei si a orientarii capului utilizatorului. Rezultatul este ca utlizatorul poate explora, vizual, mediul virtual inconjurator.

Dezavantajul major al acestei metode consta in disconfortul indus de casca motata pe capul utilizatorului. Un astfel de dispozitiv a fost testat inca din anul 1965 de Evans si Southerland.

Un alt proiect dezvltat in acest domeniu al realitatii virtuale este numit CAVE si a fost dezvltat de Universitatea Ilinois din Chicago. In acest caz sunt utilizate o serie de proiectoare ce genereaza o serie de imagini pe peretii, tavanul si podeaua unei camere cubice. Mai multe persoane ce utilizeaza ochelari stereoscopici pot intra si vizualiza imaginile din aceasta camera. Un sistem de urmarire a capului va ajusta imaginea proiectata asrfel incat sa ofere impresia unui mediu real. De asemenea au fost dezvoltate o serie de dispozitive, ca de exemplu manusi de date, *joystick*-uri, ce permit utilizatorilor o navigare mai facila prin mediul virtual. Caracteristicile unui astfel de mediu virtual pot fi rezunate astfel:

- Sistem de proiectie a imaginii bazat pe orientarea capului ce ofera o interfata naturala pentru navigarea iintr-un spatiu tridimensional virtual.
- Capailitati de vizionare stereoscopice ce ofera perceptia adncimii si senzatia de spatiu.

- Lumea virtuala este prezentata la scara normala si in proportii strans legate de dimensiunea corpului uman.
- Interactiuni realistice cu obiectele virtuale via unor dispozitive haptice gen manusi de date si alte dispozitive similare ce permit manipularea, operarea si controlul unor obiecte virtuale.
- Iluzia convingatoare a apartenentei utlizatorului la spatiul virtual poate fi imbunatatita prin elemente tactile, acustice si alte tehnici nonvizuale.
- Posibilitatea de a fi folosite in comun de mai multi utilizatori prin retea.

Acest tip de medii virtuale, ce pot fi folosite in comun de mai multi utilizatori, pot permite ca utilizatori din diferite colturi ale globului sa se intalneasca in acelasi mediu virtual, fiecare avand propria perspectiva asupra acestuia. In cadrul acestui mediu virtual fiecare utilizator va avea propria lui reprezentare (avatar) pentru alti participanti. Utilizatorii se pot vedea reciproc, pot comunica unii cu altii si pot interactiona, in mediul virtual ca si o echipa.

In prezent termenul de "Realitate Virtuala" are o aplicabilitate mult mai larga in care nu sunt incluse doar aplicatii care presupun interactiunea dintre subiectul uman si mediul virtual prin dispozitive specializate dar si medii virtuale ce permit navigarea cu ajutorul *mouse*-ului sau simularea unor procese pentru o mai buna intelegere, acesta fiind si cazul proiectului de fata.

Alte tehnologii legate de realitatea virtuala combina mediile virtuale cu cele reale. Exista, de asemenea tehnologii ce permit vizualizarea unor medii reale impreuna cu obiecte virtuale. Sistemele de teleprezenta permit unui utilizator sa fie transpus intr-un mediu real aflat la distanta, mediu ce este captat de camere video, si permite manipularea la distanta a obiectelor prin intermediul unor brate robotice si a unor manipulatoare.

Pe masura ce tehnologiile legate de realitatea virtuala evolueaza aplicatiile acestui domeniu devin practic nelimitate. Oricum un mediu virtual poate contine o reprezentare tridimensionala atat elemente reale cat si abstracte. Astfel sistemele abstracte includ sisteme precum cladiri, masini, elemente de anatomie umana etc, iar in cazul elementelor abstracte putem include campuri magnetice, modele molecurare, sisteme matematice etc.

Toate aceste lumi virtuale pot fi animate, distribuite, sau pot expune noi comportamente si functionalitati.

2. Tehnologii utilizate

Pricipalul pachet soft pe care se bazeaza apliactie este Open Inventor/ Coin. Acest capitol va descrie cateva elemente cheie ce fac parte din Open Inventor, de asemenea se va prezenta legatura dintre Open Invetor si alte instrumente de programare cum sunt OpenGL si X Windows.

Open Inventor este un set de blocuri ce permite scrierea de programe care sa profite de avantajele placilor grafice cu unefort de programare minim. Fiind bazat pe OpenGL acest pachet ofera o biblioteca de obiecte ce pot fi utilizate, modificate sau extinse pentru a intruni cele mai variate cerinte.

Obiectele din Open Inventor includ primitive ale bazei de datecum ar fi obiecte ale motorului de randare, forma, propritati si grup preum si manipulatori interactivi si componente cum sunt editor de materiale, editor de lumini directionale, fereastra de vizualizare.Open Inventor ofera pe langa un sistem de progamare orientat pe obiecte si capabilitati de transfer de date intre aplicatii pe baza unui format de fisier specific. Asa cum este prezentat si in figura 1. fundatia Open Inventor-ului este furnizata de OpenGL si Linux/Unix, pachetul, Open Inventor, oferind un model de programare si o interfata pentru programele OpenGL.Pachetul Open Inventor este independent de sistemul de interfata grafica, o biblioteca componenta facand legatura intre Inventor si o interfata grafica specifica.

Open inventor se concentreaza pe creearea de obiecte 3D. Toate informatiile despre aceste obiecte, forma, dimensiune, culoare, textura, localizare in spatiul 3D sunt memorate intr-o baza de date a scenei. Utilizarea obisnuita a unei astfel de baze de date consta in afisarea unei imagini 3D pe ecran.

Pentru multe pachete de grafica 3D, imaginea este scopul final, adica reprezentarea fotorealista a unei scene pe ecran. Probleme apar in momentul in care se incearca mutarea obiectului de afisat intr-o alta locatie, sau vizualizarea scenei dintr-o alta perspectiva.

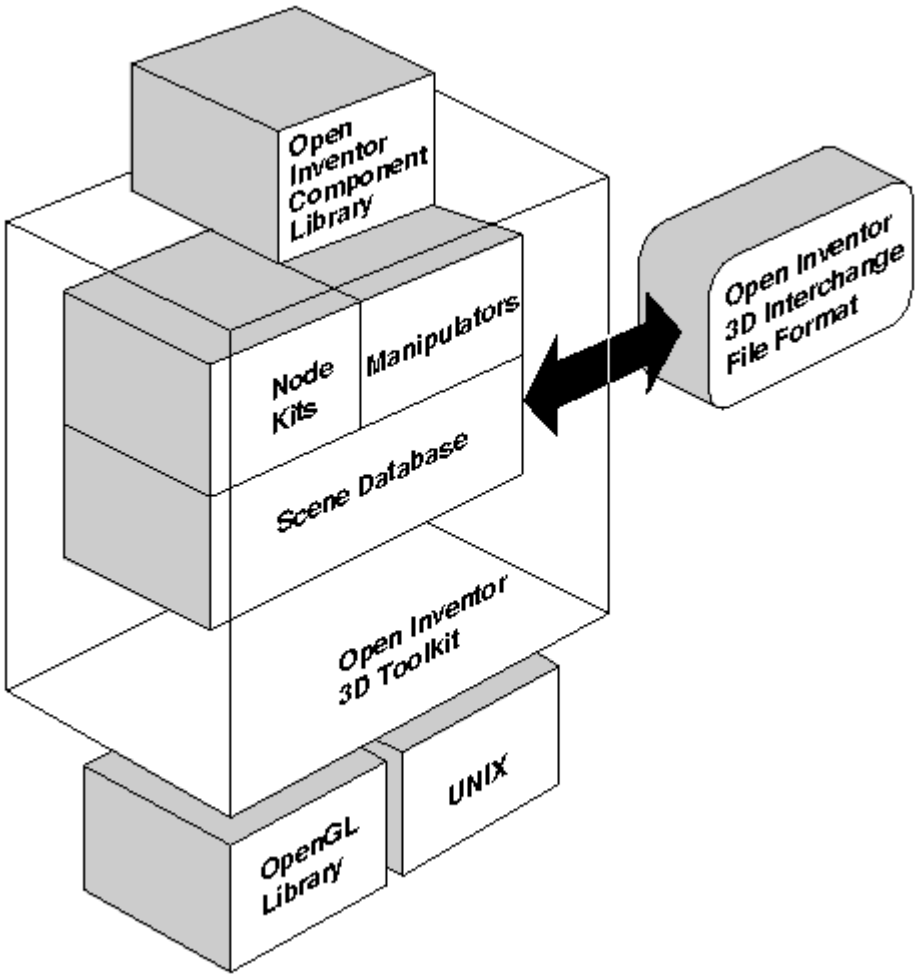


Figura 1 Arhitectura Open Inventor

Daca imaginea exista numai ca un desen pe ecran programatorul trebuie sa scrie parti complicate de cod pentru a realiza aceste obiective. De asemenea este nevoie de cod suplimentar pentru a anima aceste parti ale acestor scene. Cu Open Inventor posibilitatea de a realiza aceste modificari este integrata in modelul de programare. Modificarea obiectelor in scena, adaugarea unora noi la scena si interactiunea cu obiectele se realizeaza intr-un mod foarte simplu din cauza ca aceste modificari interfata Open Inventor, din cauza ca ele au fost anticipate in proiectarea Inventr-ului.

Din cauza ca baza de date Open Inventor contine informatii despre obiecte ca si cum ar exista in propria lume 3D si nu le reprezinta doar ca si o colectie de puncte

desenate pe ecran, alte operatii in plus fata de afisare pot fi efectuate direct asupra obiectelor. Obiectele dintr-o scena pot fi selectate, iluminate si manipulate ca si entitati discrete. Calculele pentru determinarea cubului inconjurator pot fi efectuate direct pe obiecte. Ele pot fi tiparite, asupra lor se pot efectua operatii de cautare, citite si scrise in fisire. Fiecare din aceste operatii incorporate ofera noi posibilati pentru un programator.

Obiectele Open Inventor pot, de asemenea, incapsula comportament in descrierea continuta in baza de date, ceea ce ofera posibilitatea de efectuare a diverse animatii.

Open Inventor foloseste OpenGL pentru afisare, oricum afisarea se realizeaza in mod explicit, iar in Inventor afisarea impreuna cu alte operatii cum ar fi selectarea, citirea, scrierea calcularea cubului inconjurator sunt incapsulate in obiectul propriu-zis.

OpenGL nu ofera moduri de acces imediat la memoria placii video. Asa cum s-a prezentat anterior Inventor este bazat pe un model de programare orientat pe obiecte care creeaza obiecte editabile de nivel inalt, ce pot fi memorate intr-o baza de date. Fiecare obiect incapsuleaza un set de operatii ce pot fi aplicate asupra lui: selectare, cautare in baza de date si calcularea cubului inconjurator. In Open Inventor, afisarea in memoria placii video intervine atunci cand operatia de afisare este invocata. Daca un program ce foloseste Open Inventor nu va apela aceasta comanda, fie direct fie indirect, atunci nici o scena nu va fi desenata.

Open Inventor ofera suport la diveres nivele de programare. La nivelul de interfata pentru utilizatorul final, Open Inventor ofera o imagine unificata pentru interfetele grafice 3D. La nivelul de programare ofera urmatoarele unelte:

- O baza de date a scenei 3D ce poate include: forme, proprietati, grupuri, si senzori, utilizate la creearea unei scene 3D ierarhice.
- Un set de noduri ce ofera un mecanism convenabil de grupare a nodurilor Open Inventor.
- Un set de manipulatori ce includ obiecte din baza de date a scenei cu care utilizatorii pot interactiona direct.
- Biblioteci componente pentru Xt sau Qt incluzand aria de afisare (fereastra utlizata pentru afisare), editorul de materiale, utilitare de afisare si functii utilitare ce pot fi folosite pentru unele sarcini de nivel inalt.

Un element important folosit in acest pachet este baza de date a scenei. In acest caz nodul este elementul de baza folosit pentru a crea baze de date pentru scene tridimensionale in Open Inventor. Fiecare nod contine parti de informatie, cum ar fi materilul folosit, descrierea formei, transformari geometrice, lumini, sau camere de vizualizare. Toate formele tridimensionale, atributele, camerele de vizualizare, si luminile prezente intr-o scena sunt reprezentate ca si noduri.

O colectie ordonata de noduri este denumita ca fiind graful scenei. Acest graf este memorat in baza de date a Open Inventorului, aceasta baza de date putand memora unul sau mai multe astfel de grafuri.

Dupa constructia unui graf, asupra lui se pot aplica mai multe operatii sau actiuni, printre care afisare, selectare, cautare, calcului cubului inconjurator sau scrierea intr-un fisier.

Clasele unei baze de date includ noduri ce descriu forma (ca de exemplu sfere, cub, cilindru), clase ce descriu proprietatile unui nod, (ca de exemplu materialul, modelul de iluminare, texturi, mediul) si nodurile ce descriu grupurile (ca de exemplu separatorul, nivelul de detaliu). Alte primiive speciale sunt motoare deanimatie si senzori. Motoarele de animatie sunt obiecte ce pot fi conectate cu alte obiecte si care au rolul de a anima parti ale scenei sau de a conditiona anumite parti ale scenei I relatie cu altele. Un senzor este un obiect ce detecteaza modificari ale bazei de date si apeleaza functia oferita de aplicatie. De asemenea senzorii pot raspunde la anumite specificatii de timp sau la modificari in graful scenei.

Seturile de noduri faciliteaza crearea unor baze de date consistente si structurate. Fiecare set de noduri este o colectie de noduri cu un aranjament specific. Un format asociat setului de noduri determina care noduri pot fi adaugate si unde trebuie ele plasate. De exemplu SoShapeKit este utilizate pentru orice forma de obiect utilizat in Open Inventor, acesta contine implicit un nod SoCube si permite ca propritatile legate material, a transformarilor geometrice si altor proprietati sa fie plasate la locul potrivit atunci cand este necesar.

O alta utilizare a seturilor de noduri consta in a defini obiecte si semnatici specifice aplicatiei. De exemplu o aplicatie de simulare a unei camere ar putea contine o serie de obiecte care sa reprezinte diverse elemnte de mobilier. Fiecare dintre aceste

elemnte vor avea cate un graf al scenei asemanator , sertare, tablii, similar vor fi si metode specifice – inchideSertar(), deschideSertar(). Un programator care foloseste acest pachet fiecare tip de mobilier poate fi tratat in acelasi mod. Crearea acestor obiecte si metode noi implica extinderea Open Inventor-ului prin subclasare. Este foarte importanta utilizarea unor tipuri de seturi de noduri pentru pastrarea ordinii in aplicatie.

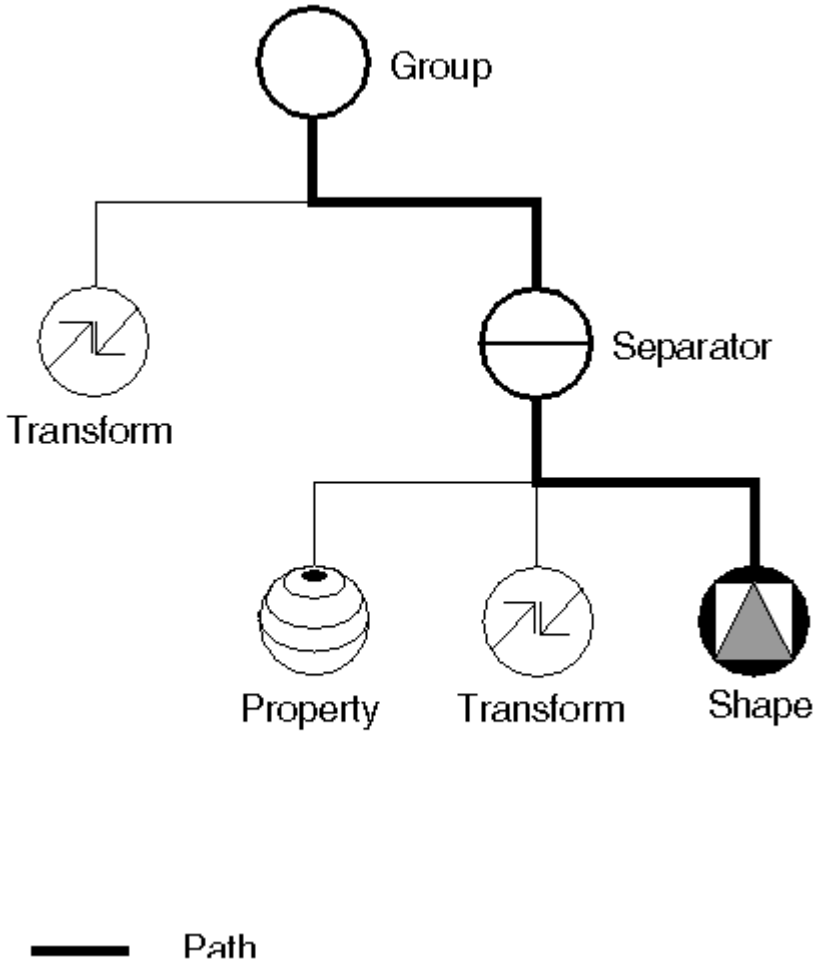


Fig. 2 Exemplu de graful unei scene

Exemplu:Cillindru

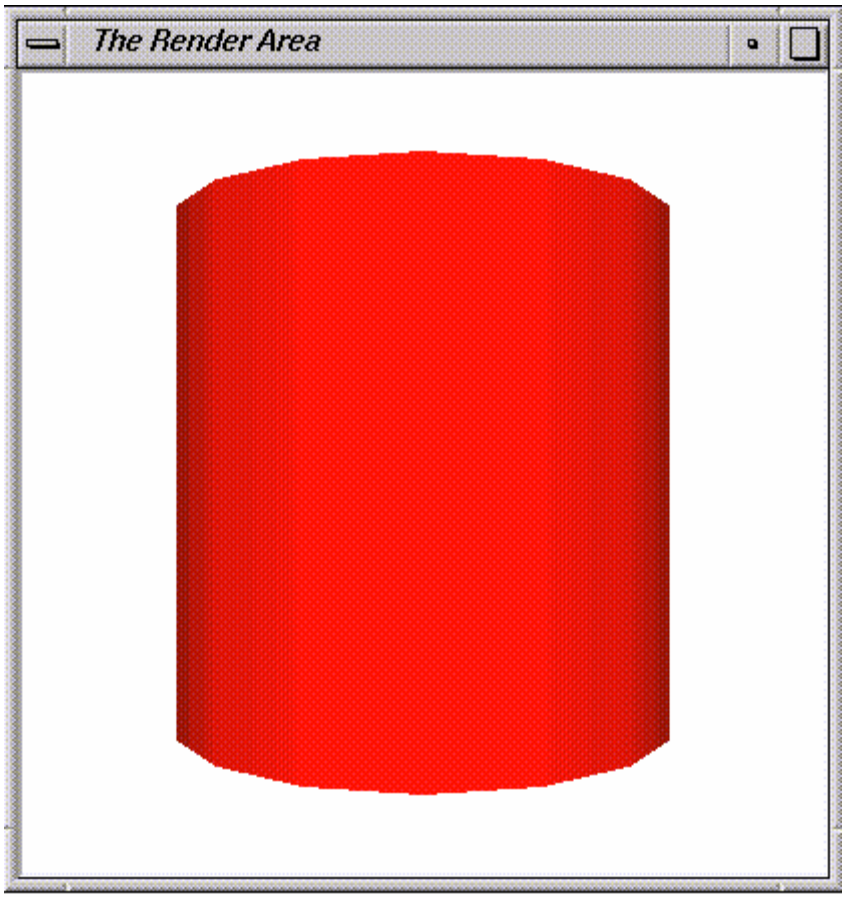


Fig. 3 Cilindru

Un exemplu simplu ce urmareste sa ilustreze modul de utilizare al pachetului Open Inventor este prezentat in cele ce urmeaza si prezinta pas cu pas toate etapele ce trebuie urmarite in cazul construirii unui cilindru.

Primul pas consta in importatrea tuturor fisierelor necesare, pachetul Open Inventor contine fisiere de definire pentru fiecrae clasa. De exemplu fisierul *SoQt.h* este este necesar daca aplicatia foloseste biblioteca Qt din cadrul Open Inventor.

```
#include < Inventor/Xt/SoXt.h >
#include < Inventor/Xt/SoXtRenderArea.h >
#include < Inventor/nodes/SoMaterial.h >
#include < Inventor/nodes/SoCylinder.h >
#include < Inventor/nodes/SoSeparator.h >
```

```
#include < Inventor/nodes/SoPerspectiveCamera.h >
#include < Inventor/nodes/SoDirectionalLight.h >
```

Pentru initializarea Open Inventor si a Qt se foloseste metoda *SoQt::init()*

```
Widget mainWindow = SoXt::init(argv[0]);
```

Aceasta metoda apeleaza *QtAppInitialize*si returneza o fereastra in care urmeaza sa se realizeze operatiile de afisare ale aplicatiei. De asemenea va realiza legatura intre Open Inventor si sistemul de evenimente Qt.

```
SoSeparator *root = new SoSeparator();
root->ref();
```

In continuare se va construi scena care va avea un nod separator ca si radacina. Nodul separator este folosit pentru a grupa noduri de alte tipuri. Rolul lui este de a salva starea traversarii grafului scenei inainte ca aceasta sa parcurga ramurile fiu si sa restaureze aceasta stare la terminarea parcurgerii fiilor. Astfel acesta va realiza o izoare a nodurilor din componenta ramurilor fiu fata de restul grafului. Nodul radacina a scenei unui graf este de obicei un nod separator pentru ca se doreste resetarea starii intre dou parcurgeri succesive a grafului unei scene.

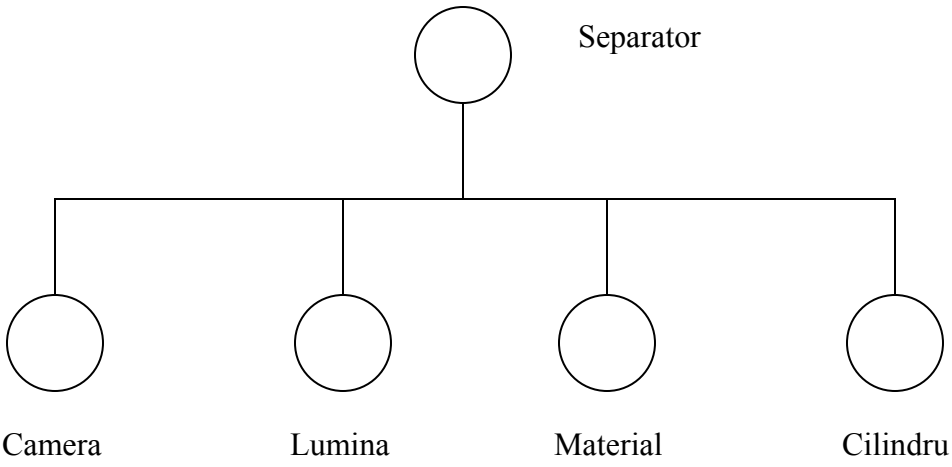
Un aspect important in orice graf al unei scene este ordinea nodurilor pe ramurile fiu. Fiecare nod implementeaza propria lui actiune. Daca se doreste implementarea unei anumite actiuni intr-o scena se creeaza o instanta a clasei respective care va fi aplicata nodului radacina a grafului scenei. Pentru fiecare actiune baza de date va genera o stare a traversarii, stare ce reprezinta o coletie de elemnte sau parametrii in actiune la un moment dat. In mod obisnuit executia unei actiuni implica traversarea grafului de sus in jos si de la stanga la dreapta. Pe parcursul acestei traversari nodurile pot modifica starea traversarii in functie de coportamentul lor particular pentru acea actiune. Starea traversarii destinate afisarii consta dintr-o serie de elemente, fiecare din ele putand fi modificate de o anumita clasa de noduri. Cand actiunea de afisare este aplicata fiecare element este utlizat si interpretat intr-o maniera specifica. Cateva din elementele ce compun starea traversarii sunt urmatoarele:

- Transformarea geometrica curenta;
- Componentele de material curente;
- Modelul de iluminare curent;
- Stilul de desenare curent;
- Fontul curent;
- Coordonatele curente;
- Normalele curente;
- Luminile curente;
- Specificatiile de vizualizare curente;

Pasul urmator va consta in crearea mai multor noduri si adaugarea acestor in graful scenei ca fii a nodului radacina. Pentru aceasta scena simpla vom creea o camera si o vom pozitiona la 4 unitati pe axa z, o lumina directionala care sa lumineze scena, un nod de material pentru care proprietatea culoare va avea culoarea rosu, iar in final un nod cilindru, implicit cilindrul va fi creat centrat in origine in jurul axei y si va avea 2 unitati inaltime si 2 unitati in diametru. De asemenea cilindrul va fi avea cele doua parti, superioara si inferioara, acoperite.

```
    SoPerspectiveCamera *myCamera = new
SoPerspectiveCamera();
    SoDirectionalLight *myLight = new SoDirectionalLight();
    SoMaterial *myMaterial = new SoMaterial();
    root->addChild(new SoCylinder);
```

In acest moment graful scenei va arata astfel:



Vom continua cu crearea unei ferestre de afisare. Petru aceasta *SoQtRenderArea* este derivata din *SoQtGLWidget* care la randul ei este derivata din *SoQtcomponent*. Aceste trei clase impreuna definesc un set de functii ce usureaza sarcina programatorului in ceea ce priveste generarea si utilizarea unor ferestre bazate pe OpenGL/Motif.

```
myRenderArea->setSceneGraph(root);
```

Vom preciza scena ce urmeaza sa fie afisata in aceasta fereastra

```
myRenderArea->show();
```

Aceasta este urmata de un apel de tipul afiseaza care gestioneaza spatiul de afisare din fereastra

```
SoQt::show(mainWindow);
```

Apelul de mai sus genereaza si gestioneaza fereastra la nivel inalt.

```
SoXt::mainLoop();
```

Acest apel este echivalent cu *QtAppMainLoop*, si va ramane intr-o bucla infinita a carui rol este sa obtina si sa gestioneze evenimentele generate atat intern cat si cele care provin de la sistemul de ferestre.

Un alt element important este manipulatorul, care este un nod de tip special ce reactioneaza la evenimantele generate de utilizator si poate fi editat direct de catre utilizator. Manipulatorii contin in mod obisnuit parti ce sunt afisate in scene si ofera mijloace de translatatre a evenimentelor in modificari in baza de date. Manipulatorii ofera o modalitate usoara de integrare a interactiunilor 3D direct in aplicatie.

Biblioteca de componente Open Inventor

Biblioteca de componente Open Inventor ofera suport si integrare in sistemul de ferestre X. Acesta bilbioteca ofera urmatoarle caracteristici:

- O zona de afisare a obiectelor – fereastra.
- Bucla principala si initializarea rutinelor.
- Un utilitar de translatare a evenimentelor.
- Editoare.
- Utilitare de vizualizare.

Fereastra de afisare accepta un evenimet de la sistemul de ferestre X si il translateaza intr-un evenimet Inventor, si apoi este trecut unor obiecte speciale cum sunt manipulatorii, obiecte ce pot gestiona astfel de evenimente.

Biblioteca de componente Open Inventor contine de asemenea si un set de utilitare de vizualizare si editoare, elemnte ce intra in general in categoria componente. Componentele sunt module reutilizabile ce contin atat aria de afisare cat si interfata cu utilizatorul. Ele sunt utilizate pentru editarea nodurilor din graful scenei (materiale, lumini, transformari) cat si pentru diferite moduri de vizualizare a scenelor. Decat sa se rezolve aceeasi problema de mai multe ori (vizualizarea scenei) este mult mai simpla selectarea unei componenete Open Inventor si utilizarea ei in aplicatie. Exemple de componente sunt editorul de materiale, editorul de lumini directionale, *fly viewer* si *examiner viewer*.

Unul dintre cele mai importante aspecte din Open Inventor este abilitatea de a programa noi obiecte si operatii ca si extensii a unei biblioteci. O modalitate de extindere a unui set de obiecte oferite de Open Inventor este crearea de noi clase prin derivarea celor existente. O alta modalitate este introducerea de noi facilitati in Inventor prin utilizarea unor functii de tip *callback*, functii ce ofera un mecanism pentru introducerea unui comportament specializat in graful scenei sau pentru prototipizarea unor noi noduri fara subclasare. O functie de tip *callback* este o functie scrisa de utilizator si este apelata in anumite conditii. Functiile de tip *callback* oferite de Open Inventor contin urmatoarele:

- SoCallback – un nod generic in baza de date ce ofera o functie de tip *callback* pentru toate tipurile de actiuni efectuate asupra bazei de date.
- SoCallbackAction – traversare generica abazei de date cu o functie de tip *callback* la fiecare nod.
- SoEventCallback – un nod in baza de date ce apeleaza o functie definita de utilizator cand receptioneaza un eveniment.
- SoSelection – selectia nodului de *callback*.
- Manipulatori – ofera functii *callback* pentru procesarea evenimentelor.
- Componenta SoQT – ofera suport pentru propriile apeluri de *callback* cand apare o modificare.

Nici o descriere a sistemului Open Inventor nu ar fi completa fara prezentarea unor noduri speciale si mai ales a foematului de fisier, format ce este unul din punctele de rezistenta a Inventorului in competitia cu alte interfete de programare grafice si al limbajelor de descriere a scenelor.

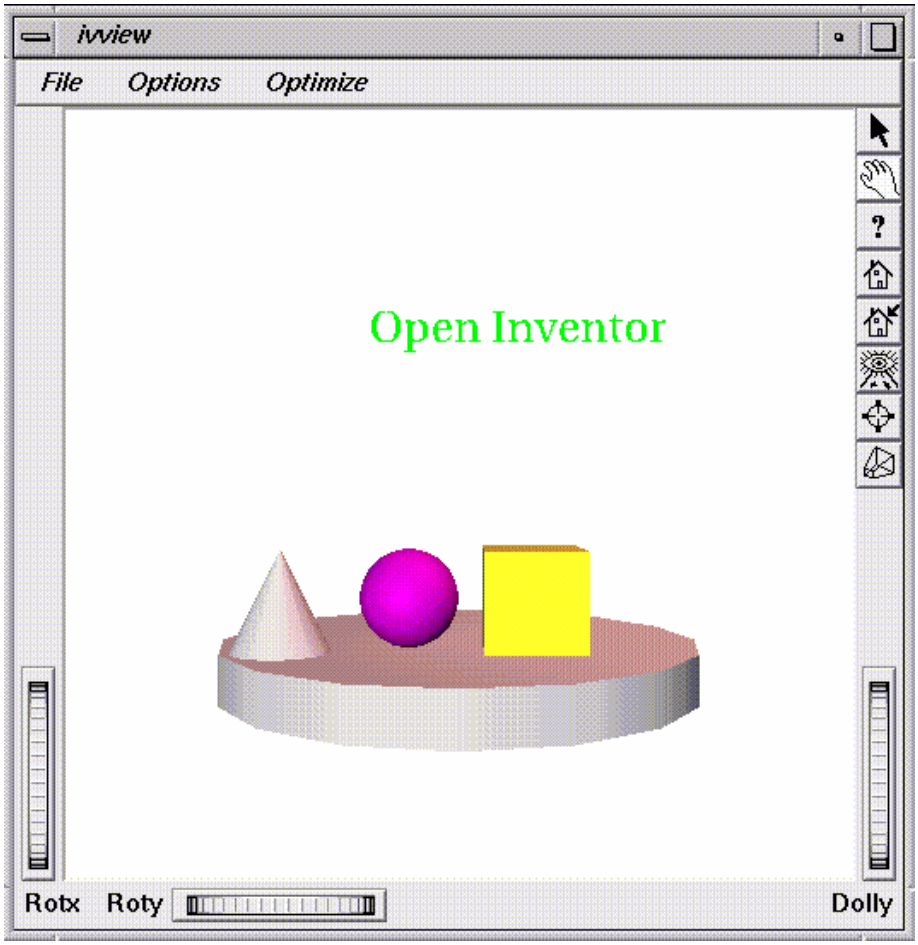
Sa consideram o scena constituita dintr-un cilindru , un cib, o sfera si un text 3D. Pe langa nodurile de tip forma obisnuite in aceasta situatie pot apare si noduri de tip text 2D, curbe indexate si neindexate de tip *nurb*, forme bazate pe *vertex*-uri si altele.

Transformarile ce urmaresc plasarea sunt aplicate cu ajutorul unor noduri de tipul Transform. Transformarile sunt cumulative, dar daca sunt folosite impreuna cu noduri de tipul Separator atunci se obtine o izolare intre mai multe secvente de transformari. Nodul separator are rolul de a salva starea transformarii inainte de traversarea fiilor lui si de restaurare a ei dupa incheierea transformarii. Nu este obligatoriu ca toate campurile unui nod sa fie specificate, automat acestea vor primi valori implicite. De exemplu pentru un

nod de tip cub pentru care nu au fost precizate latimea, inaltimea si adancimea valorile implicite sunt 1.0 1.0 1.0.

```
#Inventor V2.0 ascii

Separator {
    SpotLight {
        color 0.6 0.3 0.3
        location 5 10 0
        direction -1 -1 0
    }

    Separator {
        Transform {}
        Cylinder {
            radius 5
            height 1
        }
    }

    Separator {
        Material {
            diffuseColor 0.8 0.8 0.0
            specularColor 0.3 0.3 0.0
            ambientColor 0.2 0.2 0.2
        }
        Transform {
            translation 1.5 1.5 1.0
        }
        Cube {}
    }

    Separator {
        Material {
            diffuseColor 0.8 0.0 0.8
            specularColor 0.3 0.0 0.3
            ambientColor 0.2 0.0 0.2
        }
        Transform {
            translation -1.0 1.5 1.0
        }
        Sphere {}
    }

    Separator {
        Transform {
```

```
            translation -3.5 1.5 1.5
        }
        Cone {}
    }

    Separator {
        Material {
            diffuseColor 0.0 0.8 0.0
            specularColor 0.0 0.3 0.0
            ambientColor 0.0 0.2 0.0
        }
        Transform {
            translation -2.0 7.0 -1.0
            scaleFactor 0.1 0.1 0.1
        }
        Text3 { string "Open Inventor" }
    }

}
```



Pentru adaugarea de animatii simple se poate realiza prin utilizarea unor noduri speciale. Doua dintre ele vor fi prezentate in cele ce urmeaza.

Vom cuprinde o sursa de lumina intr-un nod de tip Blinker, astfel incat aceasta se va aprinde si stinge in mod repetat. Un nod de tip Blinker este un nod de tip comutator ce contine o serie de noduri fiu care vor fi activate pe rand. In continuare un nod de tip Rotor este inserat in scena si are rolul de a roti scena in jurul axei y.

```
.
.
.
Blinker {
    SpotLight {
        color 0.6 0.3 0.3
        location 5 10 0
        direction -1 -1 0
}

Rotor {
    rotation 0 1 0 0.1
    speed 1.0
}
.
.
.
```

Aceste exemple pot fi vizulalizate cu ajutorul unui utilitar de vizualizare de tipul *ivview, SceneViewer* sau *gview*, utilitare ce fac parte din pachetul Open Inventor.

Open Inventor este un utilitar destinat producerii rapide de aplicatii 3D. Programatorul poate spori de productivitatea prin utilizarea functiilor si primitivelor generate direct de Open Inventor. Un alt avantaj este optimizarea Inventorului in utilizarea OpenGL direct din interiorul Open Inventorului, in acest caz ne mai fiind necesare optimizari ulterioare. Probleme legate de Open Inventor pot sa apara la utilizarea acestuia intr-un mediu multiprocesor, metodele Inventorului nefiind optimizate pentru fire de executie.

3. Descrierea aplicatiilor

In cadrul proiectului este realizata aplicatia software VRoom, scopul acestuia este recrearea, in cadrul calculatorului, a unui spatiu real a unui laborator de cercetare in care se studiaza diversi algoritmi de miscare a unui dispozitiv haptic. Avantajul major al acestei abordari a studiului miscarii robotilor este faptul ca se pot obtine rapid rezultate asupra modului de comportare a robotului pentru diverse madalitati particulare de determinare a drumului robotului si a miscarii propriu-zise.

Camera implementata in proiectul prezent este o replica a laboratorului de cercetare Hashimoto de la Universitatea din Tokyo Japonia.

In prezent camera contine urmatoarele obiecte:

- Birouri;
- Scaune;
- Roboti;
- Rafturi de carti;
- Un ecran mare – folosit intr-un sistem referential pentru a stabili pozitia robotului;
- 8 camere plus cate una pe fiecare robot.

Intregul proiect a fost dezvoltat in C++, iat implemetarea grafica a obiectelor s-a realizat folosind interfata oferite de Coin/Open Inventor. In prezent camera implemeteaza reprezentarea grafica a obiectelor mentionate mai sus. Implementarea in camera virtuala a obiectelor fizice s-a realizat atat pe baza specificatiilor, date tehnice ale obiectelor respective, cat si pe baza unor poze a obiectelor reale.

Detalii de implementare:

Scaunul

Unul dintre primele obiecte implementate in camera virtuala a fost un scaun. Pentru respectivul obiect avem prezentate in figura atat o poza dupa obiectul real cat si reprezentarea sa in camera virtuala.

Poza reala Reprezentare virtuala

Fig. 5 Scaun

Functiile implementate pentru clasa Scaun Sunt urmatoarele:

```
class Chair{
  public:
    float sit_width,sit_height;//chair dimensions
      float back_width,back_height;
      float link_node_radius1,link_node_height1;
      float link_radius1;
      float link_node_radius2,link_node_height2;
      float leg_depth,leg_radius1,leg_radius2;
      float leg_height;//the height of the leg in [%]
            //0% => leg height=leg_depth
            //100% => leg height=2*leg_depth
      float foot_depth,foot_width,foot_radius;
      float wheel_radius,wheel_height;
```

```
        SoMaterial *faceMaterial;
        SoMaterial *backMaterial;

        Chair();
        //the center of the sistem is at the bottom of the chair in the middle
        SoSeparator *makeShape();//creates and return the shape
    private:
        SoSeparator *makeSitShape(float w,float h);//creates the sit of the chair
};
```

Construirea scaunului a constat din mai multe parti cele mai complexe fiind spatarul si sezutul, in principiu spatarul are aceeasi forma cu sezutul dar asupra lui s-a aplicat o rotire cu 90 grade. Parametrii folositi pentru simularea scaunului au urmatoarea forma:

```
faceMaterial=NULL;
backMaterial=NULL;
sit_width=0.45f;
sit_height=0.45f;
back_width=0.45f;
back_height=0.35f;
link_node_radius1=0.03f;
link_node_height1=0.05f;
link_node_radius2=0.1f;
link_node_height2=0.05f;
link_radius1=0.02f;
leg_depth=0.15f;
leg_radius1=0.04f;
leg_radius2=0.025f;
leg_height=0.9f;
foot_depth=0.025f;
foot_width=0.225f;
foot_radius=0.07f;
```

wheel_radius=0.02f;

wheel_height=0.02f;

Biroul

Acesta nu presupune constructii la fel de complicate ca si in cazul scaunului, in acest caz s-au utilizat forme geometrice regulate.

Birou – poza

Birou – implementare grafica

class Desk{

public:

float DESK_WIDTH, DESK_LONG, DESK_HIGH, THICK;//desk dimensions

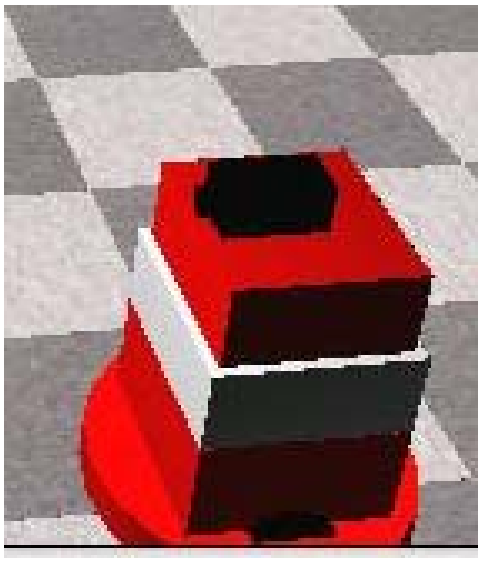
```
    float DESK_SIDE_EDGE, DESK_FRONT_EDGE;
    float BOX_WIDTH;
    float HANDEL_HEIGHT,HANDEL_RADIUS;
    float front_edge;
    int nomberOfBoxes;

    SoTexture2 *texture;
    SoSeparator *makeShape();
    SoSeparator *makeSkelet();
    SoSeparator *makeHandel();
    SoSeparator *addBox(int side);

    Desk();
};
```

Robotul

Implementarea grafica coresponzatoare robotului este cea mai saraca in detalii. Singurul element mai special asociat robotului este camera video montata in partea superioara.

```
class Robot{
 public:
  float wheel_radius, wheel_thick;
  float body_radius, body_height;
  float
camera_width,camera_height,camera_depth,camera_longs,camera_radius;

  Robot();
  SoSeparator *makeShape();
};
```

Robot – poza



Robot – reprezentare grafica

Raftul de carti

Permite o gama larga de optiuni, putand fi configurate, la instantirea obiectului, atat dimensiunile raftului, a cartilor cat si numarul de nivele. Din punctul de vedere al implementarii grafice nu a ridicat probleme deosebite, fiind construit din forme geometrice regulate.

```
class Shelf{
  public :
    int *nomberOfBooks;
    float width,height,depth;//dimensiunile raftului
    float bookWidth,bookHeight,bookDepth;//dimensiunile unei carti
    int nomber;//nomber of levels
    float DEPTH_SHELF;//inaltimea partii inchise
    int isClose;
    float radius;//use for not close shelves
    Shelf();
    SoSeparator *makeShape();
  private:
```

```
    SoSeparator *createBook();
};
```

Ecranul

Ecranul impreuna cu formele regulate de pe pardoseala permit plasarea robotului intr-un sistem de referinta si determinarea pozitiei prin mijloace optice.

Ecran + raft - poza

Ecran + raft – implementare grafica

```
class Screen{
 public:
  float width,height,depth;//dimensiunile totale
  float screen_width,screen_depth;
  float screenColor[3];//culoarea ecranului
  Screen();
  SoSeparator *makeShape();
};
```

Camere video

Principala problema legata de implementarea camerelor video a constat in stabilirea modului de trecere a vizualizarii de la o camera la alta. Solutia aleasa a constat in utilizarea unui singur obiect de tip *SoPerspectiveCamera* dar a carui perspectiva se modifica in functie de camera curenta.

```
SoQtRenderArea * renderarea = new SoQtRenderArea(window);
 camera->viewAll(userroot, renderarea->getViewportRegion());
camera->position.setValue(cameraPos[currentCamera][0],
cameraPos[currentCamera][1], cameraPos[currentCamera][2]);
camera->nearDistance=0.1;
 camera->pointAt(SbVec3f(lookAt[currentCamera][0],
lookAt[currentCamera][1],lookAt[currentCamera][2]),SbVec3f(after[currentCamera][0
],after[currentCamera][1],after[currentCamera][2]));
```

Legatura intre obiectele prezentate anterior se realizeaza in pe baza urmatorului cod:

```
#include <Inventor/Qt/SoQt.h>
#include <Inventor/Qt/SoQtRenderArea.h>

#include <Inventor/nodes/SoCube.h>
#include <Inventor/nodes/SoRotor.h>
#include <Inventor/nodes/SoArray.h>
#include <Inventor/nodes/SoDirectionalLight.h>
#include <Inventor/nodes/SoPerspectiveCamera.h>
#include <Inventor/nodes/SoSeparator.h>
#include <Inventor/nodes/SoEventCallback.h>
#include <Inventor/events/SoEvent.h>
#include <Inventor/events/SoKeyboardEvent.h>
#include <math.h>
#include <stdio.h>
#include <Inventor/nodes/SoText2.h>
#include <Inventor/nodes/SoFont.h>
#include <Inventor/nodes/SoTranslation.h>
#include <Inventor/nodes/SoRotation.h>
#include <Inventor/fields/SoMFString.h>
#include <Inventor/nodes/SoMaterial.h>
#include <Inventor/sensors/SoAlarmSensor.h>
#include <sys/types.h>
#include <unistd.h>
#include "Room.h"
#include "Chair.h"
#include "Case.h"
#include "options.h"
#include "roomObject.h"
#include <qmessagebox.h>
```

```
#include <qstring.h>

#define ANGLE_STEP M_PI/60
#define POSITION_STEP 0.03

 float cameraPos[9][3]={
  {2.55,-3.2,1},
  {-2.55,-3.2,1},
  {0.21,-2.25,0.85},
  {2.25,-0.25,1.15},
  {-2.25,-0.25,1.15},
  {2.3,2.5,1.1},
  {0.25,2.45,1.15},
  {-2.25,2.5,1.1},
  {0,0.9,-0.745}
 };

 float lookAt[9][3]={
  {0,0,0},
  {0,0,0},
  {0,0,0},
  {0,0,0},
  {0,0,0},
  {0,0,0},
  {0,0,0},
  {0,0,0},
  {0,-4,-0.745}
 };

float textPos[9][3]={
  {0,0,0},
```

```
    {0,0,0},
    {0,0,0},
    {0,0,0},
    {0,0,0},
    {0,0,0},
    {0,0,0},
    {0,0,0},
    {0,0,-0.7}
};

  float after[9][3]={
    {0,0,1},
    {0,0,1},
    {0,0,1},
    {0,0,1},
    {0,0,1},
    {0,0,1},
    {0,0,1},
    {0,0,1},
    {0,0,1}
  };

int currentCamera=8;
float x_t;
SoTranslation *trText;
SoTranslation *trTextRobot;
SoTranslation *robotTr;
SoRotation *robotRot;
SoPerspectiveCamera * camera;
SoFont *font;
SoText2 * text;
```

```
SoText2 * textRobot;
SoText2 * textHelp[6];

int noOfChairs,noOfCases;
float TIME_STEP= 85;
float (*chairsPos)[3],(*casesPos)[3];
int *chairsColor;
int flags[3]={0,0,0};
float robotPos[3],robotNextPos[3];
float robotFinalPos[3];
float interOrientation;
float angleStep;
float positionStep;
SoAlarmSensor *myAlarm;
SbTime *time_step;
int ready=0;
int cellNo;
int (*way)[2];
int wayCapacity;
int cellTaken[12][12];
float cellCenter[12][12][2];
int iInit,jInit,iFinal,jFinal;
int currentI;
int idPIPE[2];
bool running;
float robotSpeed;
Room *room;
SoSeparator * root;
SoSeparator * userroot;
QWidget * window;
bool helpOn;
```

```
bool isWay;

Options *optionsDlg;

void setText();
void  myKeyPressCB(void *userData, SoEventCallback *eventCB);
void getRoomConfiguration();
void myRaiseAlarm(void*,SoSensor*);
void myRaiseAlarmRot(void*,SoSensor*);
void testReady(void*,SoSensor*);
void myRaiseAlarmTr(void*,SoSensor*);
void updateCamera9();
void moveTo(float x,float y,float o);
void rotate(float o);
void findWay();
void getOptions();
void start();
void stop();
void help();
void makeRoom();

int main(int argc, char ** argv)
 {
   isWay=true;
   running=false;
   room=NULL;
   helpOn=true;
   userroot=NULL;
   getRoomConfiguration();
   way=new int[48][2];
   wayCapacity=48;
```

```
    window = SoQt::init(argv[0]);

    root = new SoSeparator;
    root->ref();

    SoEventCallback *myEventCB = new SoEventCallback;
    myEventCB->addEventCallback(SoKeyboardEvent::getClassTypeId(),myKeyPressCB,
root);
    root->addChild(myEventCB);

    SoSeparator *textSep = new SoSeparator;
    textSep->addChild(trText = new SoTranslation);
    trText->translation.setValue(-0.9,-0.8,0);
    textSep->addChild(font = new SoFont);
    font->name.setValue("TIMES-ROMAN");
    font->size.setValue(140);
    SoMaterial *matText=new SoMaterial;
    matText->diffuseColor.setValue(1,0,0);
    textSep->addChild(matText);
    textSep->addChild(text = new SoText2);
    root->addChild(textSep);

    SoSeparator *textSep2 = new SoSeparator;
    textSep2->addChild(trTextRobot = new SoTranslation);
    trTextRobot->translation.setValue(-0.9,-0.86,0);
    textSep2->addChild(font);
    textSep2->addChild(matText);
    textSep2->addChild(textRobot = new SoText2);
    root->addChild(textSep2);
```

```
  for(int j=0;j<6;j++){
    SoSeparator *textSepHelp = new SoSeparator;
    SoTranslation *trTextHelp;
    textSepHelp->addChild(trTextHelp = new SoTranslation);
    trTextHelp->translation.setValue(-0.9,0.9-j*0.1,0);
    textSepHelp->addChild(font);
    textSepHelp->addChild(matText);
    textSepHelp->addChild(textHelp[j] = new SoText2);
    root->addChild(textSepHelp);
  }

  root->addChild(camera = new SoPerspectiveCamera);

  makeRoom();

  SoQtRenderArea * renderarea = new SoQtRenderArea(window);
  camera->viewAll(userroot, renderarea->getViewportRegion());
  camera-
>position.setValue(cameraPos[currentCamera][0],cameraPos[currentCamera][1],came
raPos[currentCamera][2]);
  camera->nearDistance=0.1;
  camera-
>pointAt(SbVec3f(lookAt[currentCamera][0],lookAt[currentCamera][1],lookAt[current
Camera][2]),SbVec3f(after[currentCamera][0],after[currentCamera][1],after[currentC
amera][2]));
  setText();
  renderarea->setSceneGraph(root);
  renderarea->setBackgroundColor(SbColor(0.0f, 0.2f, 0.3f));

  renderarea->setTitle("Room Demo");
  renderarea->setIconTitle("Room Demo");
```

```
  renderarea->show();

  SoQt::show(window);
  myAlarm = new SoAlarmSensor;
  time_step=new SbTime;
  time_step->setMsecValue((int)TIME_STEP);
  ready=0;

  optionsDlg=new Options;

  findWay();

  SoQt::mainLoop();

  delete renderarea;
  root->unref();

  return 0;
 }

void makeRoom(){
 if(room==NULL)
  room=new Room;
 room->nomberOfChairs=noOfChairs;
 room->nomberOfCases=noOfCases;
 room->chairsPos=chairsPos;
 room->casesPos=casesPos;
 room->chairsColor=chairsColor;
 room->robotPos[0]=robotPos[0];
 room->robotPos[1]=robotPos[1];
```

```
  room->robotPos[2]=robotPos[2];
  cameraPos[8][0]=robotPos[0]+sin(robotPos[2])*0.1;
  cameraPos[8][1]=robotPos[1]-cos(robotPos[2])*0.1;
  lookAt[8][0]=robotPos[0]+sin(robotPos[2])*5;
  lookAt[8][1]=robotPos[1]-cos(robotPos[2])*5;

  if(userroot!=NULL){
    root->removeChild(userroot);
    //userroot->unref();
  }
  userroot = room->makeShape();
  robotTr=room->robotTranslation;
  robotRot=room->robotRotation;
  root->addChild(userroot);
}

void setCamera(int i){
  currentCamera=i;
  camera-
>position.setValue(cameraPos[currentCamera][0],cameraPos[currentCamera][1],came
raPos[currentCamera][2]);
    camera-
>pointAt(SbVec3f(lookAt[currentCamera][0],lookAt[currentCamera][1],lookAt[current
Camera][2]),SbVec3f(after[currentCamera][0],after[currentCamera][1],after[currentC
amera][2]));
    setText();
}

void
myKeyPressCB(void *userData, SoEventCallback *eventCB)
{
```

```
const SoEvent *event = eventCB->getEvent();

 // Check for the Up and Down arrow keys being pressed.
if (SO_KEY_PRESS_EVENT(event, SoKeyboardEvent::NUMBER_1)) {
  setCamera(0);
}
if (SO_KEY_PRESS_EVENT(event, SoKeyboardEvent::NUMBER_2)) {
  setCamera(1);
}
if (SO_KEY_PRESS_EVENT(event, SoKeyboardEvent::NUMBER_3)) {
  setCamera(2);
}
if (SO_KEY_PRESS_EVENT(event, SoKeyboardEvent::NUMBER_4)) {
  setCamera(3);
}
if (SO_KEY_PRESS_EVENT(event, SoKeyboardEvent::NUMBER_5)) {
  setCamera(4);
}
if (SO_KEY_PRESS_EVENT(event, SoKeyboardEvent::NUMBER_6)) {
  setCamera(5);
}
if (SO_KEY_PRESS_EVENT(event, SoKeyboardEvent::NUMBER_7)) {
  setCamera(6);
}
if (SO_KEY_PRESS_EVENT(event, SoKeyboardEvent::NUMBER_8)) {
  setCamera(7);
}
if (SO_KEY_PRESS_EVENT(event, SoKeyboardEvent::NUMBER_9)) {
  setCamera(8);
}
if (SO_KEY_PRESS_EVENT(event, SoKeyboardEvent::F2)) {
```

```
  getOptions();
 }
 if (SO_KEY_PRESS_EVENT(event, SoKeyboardEvent::F1)) {
  help();
 }
 if (SO_KEY_PRESS_EVENT(event, SoKeyboardEvent::F3)) {
  start();
 }
 if (SO_KEY_PRESS_EVENT(event, SoKeyboardEvent::F4)) {
  stop();
 }
 if (SO_KEY_PRESS_EVENT(event, SoKeyboardEvent::ESCAPE)) {
  exit(0);
 }
}

void help(){
 helpOn=!helpOn;
 setText();
}

void start(){
 if(!isWay){
   QMessageBox::information(0,"roomView - Message","There is a problem.");
   return;
 }
 if(!running){
  running=true;
  myAlarm->setFunction(testReady);
  myAlarm->setTimeFromNow(*time_step);
  myAlarm->schedule();
```

```
  }
}

void stop(){
  running=false;
}

void getOptions(){
  int i;
  stop();
  RoomObject *obj;
  optionsDlg->objects.clear();
  for(i=0;i<noOfChairs;i++){
    obj=new RoomObject;
    obj->x=chairsPos[i][0];
    obj->y=chairsPos[i][1];
    obj->alfa=chairsPos[i][2]*180/M_PI;
    obj->type=CHAIR_OBJ;
    obj->color=chairsColor[i];
    optionsDlg->objects.append(obj);
  }
  for(i=0;i<noOfCases;i++){
    obj=new RoomObject;
    obj->x=casesPos[i][0];
    obj->y=casesPos[i][1];
    obj->alfa=casesPos[i][2]*180/M_PI;
    obj->type=BOX_OBJ;
    obj->color=0;
    optionsDlg->objects.append(obj);
  }
  optionsDlg->startPoint[0]=robotPos[0];
```

```
optionsDlg->startPoint[1]=robotPos[1];
optionsDlg->startPoint[2]=robotPos[2]*180/M_PI;
optionsDlg->finalPoint[0]=robotFinalPos[0];
optionsDlg->finalPoint[1]=robotFinalPos[1];
optionsDlg->finalPoint[2]=robotFinalPos[2]*180/M_PI;
optionsDlg->speed=robotSpeed;
for(i=0;i<8;i++){
  optionsDlg->lookAtPoints[i][0]=lookAt[i][0];
  optionsDlg->lookAtPoints[i][1]=lookAt[i][1];
  optionsDlg->lookAtPoints[i][2]=lookAt[i][2];
}
optionsDlg->init();
optionsDlg->exec();
if(optionsDlg->ok){
  noOfChairs=0;
  noOfCases=0;
  for(obj=optionsDlg->objects.first();obj!=NULL;obj=optionsDlg->objects.next()){
    if(obj->type==CHAIR_OBJ)
        noOfChairs++;
    else
        noOfCases++;
  }
  delete [] chairsPos;
  delete [] casesPos;
  delete [] chairsColor;
  chairsPos=NULL;
  casesPos=NULL;
  chairsColor=NULL;
  if(noOfChairs>0){
    chairsPos=new float[noOfChairs][3];
    chairsColor=new int[noOfChairs];
```

```
}
if(noOfCases>0)
  casesPos=new float[noOfCases][3];
int iChair=0;
int iCase=0;
for(obj=optionsDlg->objects.first();obj!=NULL;obj=optionsDlg->objects.next()){
  if(obj->type==CHAIR_OBJ){
       chairsPos[iChair][0]=obj->x;
       chairsPos[iChair][1]=obj->y;
       chairsPos[iChair][2]=obj->alfa*M_PI/180;
       chairsColor[iChair]=obj->color;
       iChair++;
  }
  else{
       casesPos[iCase][0]=obj->x;
       casesPos[iCase][1]=obj->y;
       casesPos[iCase][2]=obj->alfa*M_PI/180;
       iCase++;
  }
}

robotPos[0]=optionsDlg->startPoint[0];
robotPos[1]=optionsDlg->startPoint[1];
robotPos[2]=optionsDlg->startPoint[2]*M_PI/180;
robotFinalPos[0]=optionsDlg->finalPoint[0];
robotFinalPos[1]=optionsDlg->finalPoint[1];
robotFinalPos[2]=optionsDlg->finalPoint[2]*M_PI/180;
robotSpeed=optionsDlg->speed;
TIME_STEP=100*POSITION_STEP/robotSpeed;
TIME_STEP*=1000;
time_step->setMsecValue((int)TIME_STEP);
```

```
    for(i=0;i<8;i++){
      lookAt[i][0]=optionsDlg->lookAtPoints[i][0];
      lookAt[i][1]=optionsDlg->lookAtPoints[i][1];
      lookAt[i][2]=optionsDlg->lookAtPoints[i][2];
    }
    makeRoom();
    setCamera(currentCamera);
    ready=0;
    running=false;
    currentI=0;
    findWay();
  }
}

void setText(){
    char buf[64];
    sprintf(buf,"CCD%d\0",currentCamera+1);
    text->string.setValue(buf);

sprintf(buf,"Robot(%.2f,%.2f,%.0f)\0",robotPos[0],robotPos[1],robotPos[2]*180/M_PI)
;
    textRobot->string.setValue(buf);
    textHelp[0]->string.setValue(helpOn?"F1 - Hide help":"F1 - Show help");
    textHelp[1]->string.setValue(helpOn?"F2 - Options":"");
    textHelp[2]->string.setValue(helpOn?"F3 - Start":"");
    textHelp[3]->string.setValue(helpOn?"F4 - Stop":"");
    textHelp[4]->string.setValue(helpOn?"\"n\" - CCD \"n\"":"");
    textHelp[5]->string.setValue(helpOn?"ESC - Quit":"");

}
```

```
void getRoomConfiguration(){
  FILE *f;
  f=fopen("room.config","r");
  char buffer[32];
  int i;
  noOfChairs=0;
  noOfCases=0;
  chairsPos=NULL;
  casesPos=NULL;
  chairsColor=NULL;

robotPos[0]=robotNextPos[0]=robotPos[1]=robotNextPos[1]=robotPos[2]=robotNext
Pos[2]=0;
  while(fscanf(f,"%s",buffer)!=EOF){
    if(buffer[0]=='#')
      break;
    if(strcmp("speed",buffer)==0){
          if(fscanf(f,"%s",buffer)!=EOF){
             robotSpeed=atof(buffer);
             TIME_STEP=100*POSITION_STEP/atof(buffer);
             TIME_STEP*=1000;
          }
          else{
             printf("Wrong format for room.config\n");
             exit(1);
          }
    }
    if(strncmp("chair",buffer,5)==0){
      if(fscanf(f,"%s",buffer)!=EOF){
          i=atoi(buffer);
          noOfChairs=i;
```

```
chairsPos=new float[i][3];
chairsColor=new int[i];
while(i--){
 if(fscanf(f,"%s",buffer)!=EOF){
  chairsPos[i][0]=atof(buffer);
 }
 else{
  printf("Wrong FORMAT for room.config\n");
  exit(1);
 }
 if(fscanf(f,"%s",buffer)!=EOF){
  chairsPos[i][1]=atof(buffer);
 }
 else{
  printf("Wrong FORMAT for room.config\n");
  exit(1);
 }
 if(fscanf(f,"%s",buffer)!=EOF){
  chairsPos[i][2]=(atof(buffer)/180.0f)*M_PI;
 }
 else{
  printf("Wrong FORMAT for room.config\n");
  exit(1);
 }
 if(fscanf(f,"%s",buffer)!=EOF){
  chairsColor[i]=atoi(buffer);
 }
 else{
  printf("Wrong FORMAT for room.config\n");
  exit(1);
 }
```

```
   }
  }
  else{
       printf("Wrong FORMAT for room.config\n");
       exit(1);
  }
}
if(strncmp("case",buffer,4)==0){
  if(fscanf(f,"%s",buffer)!=EOF){
       i=atoi(buffer);
       noOfCases=i;
       casesPos=new float[i][3];
       while(i--){
         if(fscanf(f,"%s",buffer)!=EOF){
           casesPos[i][0]=atof(buffer);
         }
         else{
           printf("Wrong FORMAT for room.config\n");
           exit(1);
         }
         if(fscanf(f,"%s",buffer)!=EOF){
           casesPos[i][1]=atof(buffer);
         }
         else{
           printf("Wrong FORMAT for room.config\n");
           exit(1);
         }
         if(fscanf(f,"%s",buffer)!=EOF){
           casesPos[i][2]=(atof(buffer)/180.0f)*M_PI;
         }
         else{
```

```
            printf("Wrong FORMAT for room.config\n");
            exit(1);
          }
        }
    }
    else{
        printf("Wrong FORMAT for room.config\n");
        exit(1);
    }
}
if(strncmp("robot",buffer,5)==0){
    if(fscanf(f,"%s",buffer)!=EOF){
            robotPos[0]=atof(buffer);
    }
    else{
            printf("Wrong FORMAT for room.config\n");
            exit(1);
    }
    if(fscanf(f,"%s",buffer)!=EOF){
            robotPos[1]=atof(buffer);
    }
    else{
            printf("Wrong FORMAT for room.config\n");
            exit(1);
    }
    if(fscanf(f,"%s",buffer)!=EOF){
            robotPos[2]=(atof(buffer)/180.0f)*M_PI;
    }
    else{
            printf("Wrong FORMAT for room.config\n");
            exit(1);
```

```
    }
    if(fscanf(f,"%s",buffer)!=EOF){
        robotFinalPos[0]=atof(buffer);
    }
    else{
        printf("Wrong FORMAT for room.config\n");
        exit(1);
    }
    if(fscanf(f,"%s",buffer)!=EOF){
        robotFinalPos[1]=atof(buffer);
    }
    else{
        printf("Wrong FORMAT for room.config\n");
        exit(1);
    }
    if(fscanf(f,"%s",buffer)!=EOF){
        robotFinalPos[2]=(atof(buffer)/180.0f)*M_PI;
    }
    else{
        printf("Wrong FORMAT for room.config\n");
        exit(1);
    }
  }
 }
}

void myRaiseAlarm(void*,SoSensor*){
 if(running){
  if(flags[0]==0){
   float d=sqrt((robotPos[0]-robotNextPos[0])*(robotPos[0]-
robotNextPos[0])+(robotPos[1]-robotNextPos[1])*(robotPos[1]-robotNextPos[1]));
```

```
interOrientation=asin((robotNextPos[0]-robotPos[0])/d);
if(interOrientation<0){
    if((robotNextPos[1]-robotPos[1])<0)
      interOrientation=interOrientation+M_PI*2;
    else
      interOrientation=fabs(interOrientation)+M_PI;
}
else{
    if((robotNextPos[1]-robotPos[1])>0)
      interOrientation=M_PI-interOrientation;
}
float deltaAngle=interOrientation-robotPos[2];
if(deltaAngle>0){
    if(deltaAngle<=M_PI)
      angleStep=ANGLE_STEP;
    else
      angleStep=-ANGLE_STEP;
}
else{
    if(deltaAngle>=-M_PI)
      angleStep=-ANGLE_STEP;
    else
      angleStep=ANGLE_STEP;
}
if(interOrientation==robotPos[2]){
    flags[0]=1;

    myAlarm->setFunction(myRaiseAlarm);
    myAlarm->setTimeFromNow(*time_step);
    myAlarm->schedule();
}
```

```
  else{

      myAlarm->setFunction(myRaiseAlarmRot);
      myAlarm->setTimeFromNow(*time_step);
      myAlarm->schedule();
  }
}
else{
  if(flags[1]==0){
      positionStep=POSITION_STEP;
      myAlarm->setFunction(myRaiseAlarmTr);
      myAlarm->setTimeFromNow(*time_step);
      myAlarm->schedule();
  }
  else{
      if(flags[2]==0){
        interOrientation=robotNextPos[2];
        float deltaAngle1=interOrientation-robotPos[2];
        if(deltaAngle1>0){
          if(deltaAngle1<=M_PI)
            angleStep=ANGLE_STEP;
          else
            angleStep=-ANGLE_STEP;
        }
        else{
          if(deltaAngle1>=-M_PI)
            angleStep=-ANGLE_STEP;
          else
            angleStep=ANGLE_STEP;
        }
        myAlarm->setFunction(myRaiseAlarmRot);
```

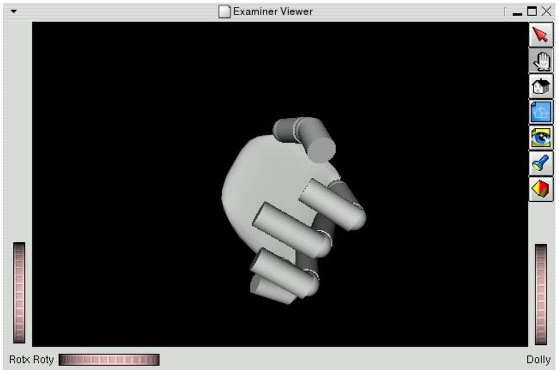
```
            myAlarm->setTimeFromNow(*time_step);
            myAlarm->schedule();
          }
          else{
            if(currentI<=cellNo){
              myAlarm->setFunction(testReady);
              myAlarm->setTimeFromNow(*time_step);
              myAlarm->schedule();
            }
          }
      }
    }
  }
  else{
    flags[0]=0;
    flags[1]=0;
    flags[2]=0;
  }
}

void myRaiseAlarmRot(void*,SoSensor*){
  if(running){
    robotPos[2]+=angleStep;
    if(robotPos[2]<0)
      robotPos[2]+=2*M_PI;
    if(robotPos[2]>2*M_PI)
      robotPos[2]-=2*M_PI;
    if(fabs(robotPos[2]-interOrientation)<ANGLE_STEP){
      robotPos[2]=interOrientation;
      robotRot->rotation.setValue(SbVec3f(0,1,0),robotPos[2]);
      if(flags[0]==0)
```

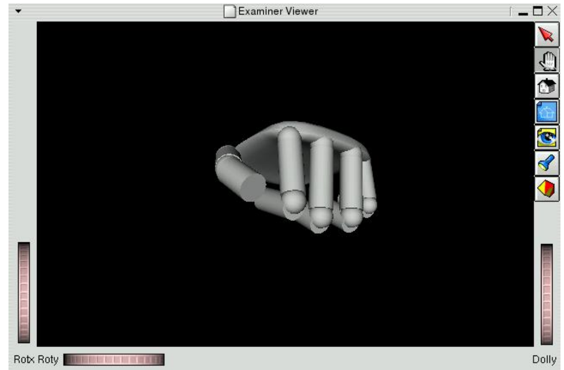
```
        flags[0]=1;
    else
        flags[2]=1;
    updateCamera9();
    myAlarm->setFunction(myRaiseAlarm);
    myAlarm->setTimeFromNow(*time_step);
    myAlarm->schedule();
   }
   else{
    robotRot->rotation.setValue(SbVec3f(0,1,0),robotPos[2]);
    updateCamera9();
    myAlarm->setFunction(myRaiseAlarmRot);
    myAlarm->setTimeFromNow(*time_step);
    myAlarm->schedule();
   }
  }
  else{
   flags[0]=0;
   flags[1]=0;
   flags[2]=0;
  }
}
```

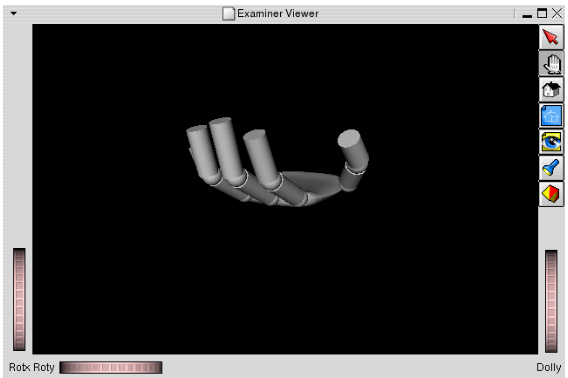
Pentru vizualizarea acestora sunt posibile doua modalitati:

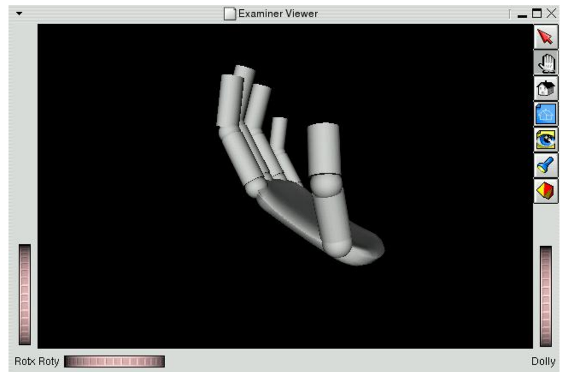
- Folosirea unui utilitar ce permite vizualizarea libera a mediului virtual;
- Afisarea perspectivei fiecarei camere in parte – este posibila vizualizarea camerei virtule din perspectiva camerelor video fixe, singura camera care poate oferi imagini in miscare este camera fixata pe robot.

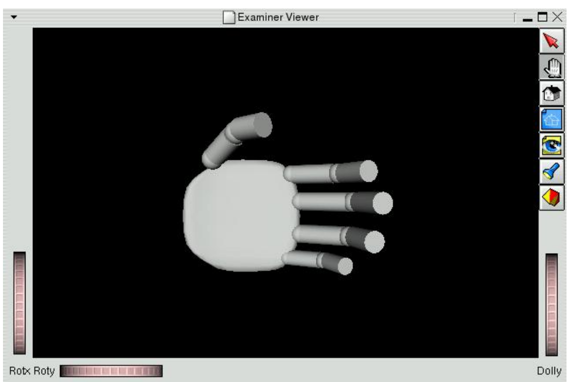
Rezultatele simularii dispozitivului haptic au utilizat tehnologia folosita la camera virtuala.

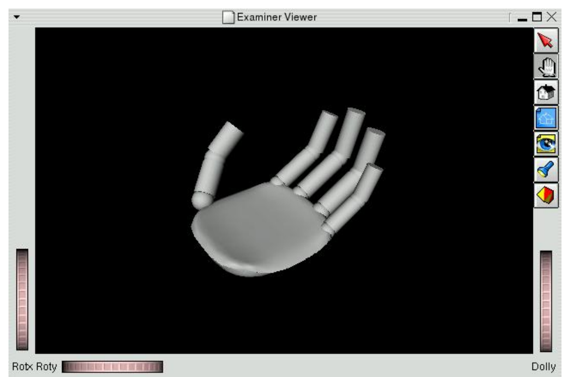

Examiner Viewer
Rotx Roty
Dolly

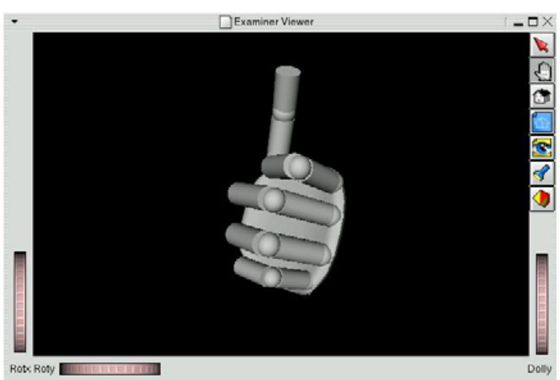

Examiner Viewer
Rotx Roty
Dolly

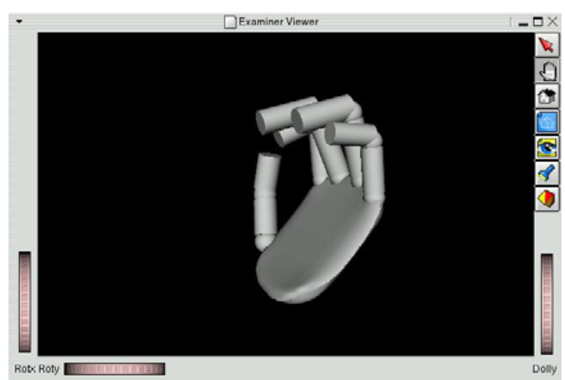

Examiner Viewer
Rotx Roty
Dolly


Examiner Viewer
Rotx Roty
Dolly


Examiner Viewer
Rotx Roty
Dolly


Examiner Viewer
Rotx Roty
Dolly


Examiner Viewer
Rotx Roty
Dolly

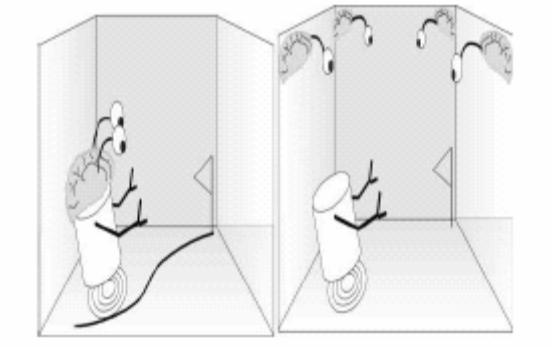

Examiner Viewer
Rotx Roty
Dolly

4. Alte aplicatii posibile ale camerei virtuale

Rolul Camerei Virtuale este de a permite simularea, pentru roboti, a unor comportamente si algoritmi de miscare inainte de implementarea lor in viata reala. Un exemplu tipic este evaluarea obiceiurilor de mers a locuitorilor unui spatiu inteligent. In acest caz conceptul de spatiu inteligent mediu virtual este aplicat pentru ajutorarea oamenilor cu handicap in spatii aglomerate.

Un robot mobil cu functii extinse este folosit ca un robot asistent mobil pentru persoane cu deficiente de vedere, care la randul lui este asistat in functionare de o retea de senzori numita spatiu inteligent. Robotul poate ghida si proteja o persoana cu deficiente vizuale intr-un spatiu aglomerat cu ajutorul saptiului inteligent. Spatiul va invata modul de evitare a obstacolelor de catre obiectele dinamice (participantii umani) prin urmarirea miscarilor acestora. Cunostintele astfel obtinute sunt codificate intr-o structura matematica si sunt transmise robotului. Robotul estimeaza calea obiectelor dinamice si ajuta persoanele cu deficiente sa evite coliziunea.

In general se urmareste creasterea inteligentei robotului, agentului, ce opereaza intr-un spatiu restrans. Conceptul de spatiu inteligent este insa opus acestei directii. In acest caz spatiul inconjurator va fi dotat cu senzori si inteligenta in locul robotului. In aceste conditii un robot fara nici un senzor sau inteligenta poate opera intr-un astfel de spatiu. Diferenta dintre un spatiu conventional si un spatiu inteligent se poate vedea in figura urmatoare:



Elementele importante ce constitue spatiul inteligent curent sunt urmatoarele:

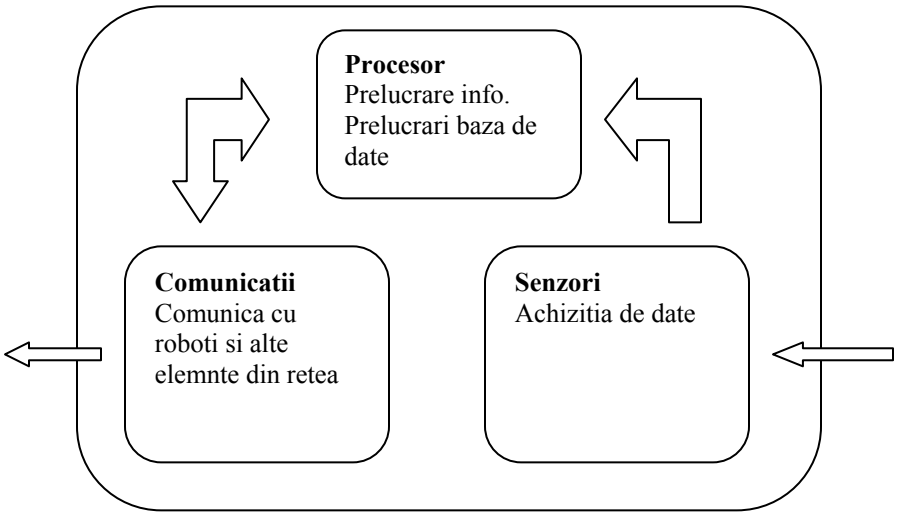
- Reteaua distribuita inteligenta;

- Camera virtuala;
- Robotul mobil;

Reteaua distribuita inteligenta este elementul de baza din componenta spatiului inteligent. Ea este constituita din trei elemente de baza:

- senzori – camere video cu microfon;
- unitatea de procesare – calculator;
- dispozitivul de comunicare – legatura la LAN;

In primul rand senzorii monitorizeaza mediul dinamic ce contine oameni si roboti. Urmeaza procesarea datelor si luarea deciziilor. Al treilea elemnt este comnunicatia intre cele elemntele retelei si cu robotii prin LAN.



Camera virtuala – antrenarea unui sistem adaptiv direct cu subiecti umani este extrem de eficienta in ceea ce priveste specificarea comportamentului pentru sistemele complexe. Una dintre dificultatile care apar in antrenarea unui sistem real este sistemul in sine. Mediul de antrenament trebuie sa fie un model pentru toate situatiile viitoare ce pot aparea, iar construirea unui mediu real doar pentru procesul de antrenament nu este doar o cheltuiala inutila (luand in considerare si posibilele deterioarari ale echipamentelor) dar cate odata este situatie nerezolvabila. Alte aspecte ale necesitatii spatiilor virtuale au fost deja dezbatute in telerobotica. Multe arii de aplicatii sunt nesigure pentru operatorul uman (antrenorul in acest caz), sau in afara domeniului de acoperire.

5. Concluzii

Open Inventor este un mediu destinat dezvoltarii rapide de soft, dar ofera si facilitati ce conduc la dezvoltarea de aplicatii de mari dimensiuni.

Ca si platforma de dezvoltare pachetul de aplicatii Coin/Open Inventor impreuna cu sisteme de operare din familia Linux, ofera opartunitati deosebite, atat datorita costurilor reduse, cat si datorita dezvoltarii de aplicatii sub licenta Open Source ceea ce permite schimbul de idei si cod si determina construirea de aplicatii robuste.

Din pacate viitorul Open Inventorului, tehnologia de baza in aplicatiile de fata, nu este clara. Pe langa apariatia unor noi interfete de programare grafica precum Cosmo3D, OpenGL++, Optimizer si Farenheit pe acesteasta scena a aplicatiilor 3D competitorul cel mai serios este Direct3D de la Microsoft. Alte interfete de programare sunt DirectModel de la HP si Java3D dar a carui mare dezavantaj este lipsa unui format de fisier in care sa poata fi salvate scenele grafice. In acest sens Open Inventor are un avantaj care combinat cu larga sa disponibilitate, pe mai multe platforme si sisteme de operare, ii ofera mai sanse de dezvoltare in viitor.

6. Bibliografie

1. J Wernecke, “The Inventor Mentor Programming Object-Oriented 3D Graphics with Open Inventor”, Addison-Wesley Publishing Company, 1994.
2. Open Inventor Architecture Group, “Open Inventor C++ Reference Manual”, Addison-Wesley Publishing Company, 1994.
3. Open Inventor Architecture Group, “The Inventor ToolMaker”, http://techpubs.sgi.com/library/, 1994.
4. Open Inventor Architecture Group, “Open Inventor: Nodes Quick Reference”, http://techpubs.sgi.com/library/, 1994.
5. P. Korondi, H. Hasimoto, “Intelligent Space, As An Integrated Intelligent System”, Electrical Drives and Power Electronics, Slovakia,2003
6. http://playerstage.sourceforge.net/