PARTIȚIONAREA DINAMICĂ A SPAȚIULUI DE TRAVERSARE A BAZELOR DE DATE GRAFICE

Dr. ing. Felicia Ionescu,

Simultec S.A., București

Rezumat: Lucrarea evidențiază condițiile de reprezentare a bazelor de date grafice din aplicațiile de sinteză de imagine, care să permită exploatarea eficientă, prin partiționarea dinamică a spațiului de traversare a acestora, în arhitecturi paralele. Domeniul în care are loc partiționarea dinamică este spațiul de stare a traversării, reprezentat printr-o stivă de stări, iar controlul global al traversării este asigurat prin identificarea unică a fiecărui obiect din baza de date, indiferent de localizarea structurii interne a bazei de date.

Cuvinte-cheie: sinteză de imagine, baze de date grafice, arhitecturi paralele, partiționare dinamică a algoritmilor.

1. Introducere

Bazele de date grafice reprezintă datele de intrare pentru diferitele categorii de sisteme de sinteză de imagine în timp real: sistemele vizuale ale simulatoarelor de antrenament în diferite domenii de activitate, tehnologia multimedia, realitate virtuală. Datorită cerințelor de putere mare de calcul, deosebit de necesară pentru generarea imaginilor în timp real, majoritatea și, de fapt, se poate afirma că toate aceste sisteme sunt realizate prin intermediul procesării paralele, în arhitecturi care acoperă domenii foarte vaste de abordări ca organizarea fluxului de date, a spațiului de memorie sau a modului de comunicație între procesoarele componente ale sistemului [1].

Un factor major în utilizarea eficientă a sistemelor multiprocesor îl constituie determinarea modului optim de partiționare a sarcinilor de calcul și planificarea acestora în procesoarele componente. Proliferarea sistemelor de calcul distribuit a generat investigații vaste în problema partiționării algoritmilor și sunt cunoscute un număr impresionant de strategii de partiționare și de planificare a algoritmilor în sisteme multiprocesor. Majoritatea acestor strategii se referă la rezolvarea unor anumite categorii de algoritmi, pentru constrângeri de planificare, specifice arhitecturii pe care se implementează. Generarea imaginilor unui mediu sintetic prin parcurgerea (traversarea) bazei de date grafice, care conține modelarea obiectelor și a fenomenelor sintetizate într-o arhitectură multiprocesor, necesită partiționarea spațiului de traversare a bazei de date și planificarea claselor partiției în procesoarele componente.

Partiționarea statică a unui arbore nestructurat, așa cum este arborele de traversare al unei baze de date oarecare, produce performanțe slabe datorită variației mari între dimensiunile spațiilor de traversare cu rădăcina în diferite noduri. Mai mult, deoarece spațiul de traversare este generat dinamic, în funcție de vizibilitatea din punctul dat de observare a obiectelor descrise de diferite noduri ale grafului bazei de date, este dificilă o estimare corectă a dimensiunilor spațiului, înainte de traversarea efectivă. Pentru echilibrarea încărcării procesoarelor, traversarea în paralel a bazei de date trebuie să fie executată prin partiționare și amplasare a partițiilor spațiului de traversare în mod dinamic.

Pentru partiționarea dinamică a spațiului de traversare, baza de date grafică trebuie să fie proiectată în corelație cu arhitectura paralelă a generatorului de imagine. În cadrul acestei proiectări, trebuie să fie prevăzute atribute și funcții ale claselor de obiecte grafice componente, care să permită implementarea strategiilor de partiționare dinamică, adecvate arhitecturii sistemului.

2. Definirea atributelor de partiționare dinamică

La crearea structurii interne a bazei de date grafice într-un sistem de sinteză de imagine, se execută o serie de calcule, pentru determinarea și introducerea unor atribute ale claselor de obiecte grafice, necesare partiționării dinamice a spațiului de traversare. Atributele care se determină și se memorează ca date membre ale claselor de obiecte din baza de date sunt descrise mai jos.

2.1. Etichetele nodurilor grafului

Baza de date grafică este generată sub forma unui graf aciclic direcționat, fiecare nod al grafului reprezentând o clasă de obiecte grafice. Clasele de obiecte grafice, derivate dintr-o clasă de bază, sunt reprezentate sub forma unui vector de pointeri către aceste clase derivate (m_ChildArray), vector care este un membru al clasei de bază. Exploatarea unei baze de date într-o arhitectură paralelă, în care diferitele procesoare componente accesează și prelucrează noduri diferite ale grafului bazei de date, necesită posibilitatea de identificare unică a fiecărui nod. Identificarea nodurilor nu este posibilă prin pointerii lor, deoarece aceștia pot avea valori diferite în reprezentările replicate ale bazei de date în procesoare diferite și, de aceea, se numerotează (etichetează) nodurile grafului și se memorează în fiecare nod atributul "label" (eticheta nodului), care permite definirea, în mod unic, a fiecărui nod în graf. Etichetarea nodurilor se face prin executarea unui algoritm de căutare în adâncime (depth-first-search), cu vizitarea o

singură dată a fiecărui nod şi cu etichetarea la vizitare.

Reprezentarea grafului bazei de date în limbaj C++ , folosind biblioteca de clase de obiecte MFC 4.0 a compilatorului Microsoft Visual C++ 4.0, este dată mai jos:

```
class CNode : public CObject
{
private:
    UINT            m_Label;
    CExtension      m_Ext;
    CChildArray     m_ChildArray;
public:
    CNode( ) { }
    CNode(CNode& obj);
    UINT GetLabel() { return m_Label;}
    CChildArray* GetChildArray()
{return &m_ChildArray;}
    void SetLabel(UINT Label)
            { m_Label = Label;}
    int GetChildArraySize()
{return m_ChildArray.GetSize();}
    ~CNode() { }
};

class CChild : public CObject
{
private:
    CMatrix         m_M;
    CNode*          m_pNode;
public:
    CChild();
    CChild(CChild& c);
    CNode* GetNode()
            {return m_pNode;}
    CMatrix* GetMatrix()
            {return &m_M;}
    ~CChild();
};

class CChildArray : public CObArray
{
public:
CChild* GetAt(int index);
int Add(CChild& o);
    int GetSize()
        {return CObArray::GetSize();
    ~CChildArray();
};
```

Un nod în graful bazei de date (obiect din clasa CNode) conţine, ca dată membru a clasei, eticheta nodului (m_Label) şi un vector (m_ChildArray) de obiecte CChild. Un obiect din clasa CChild defineşte un descendent al nodului curent, iar vectorul m_ChildArray defineşte toţi descendenţii nodului, deci toate clasele de obiecte grafice derivate. Un obiect din clasa CChild conţine un pointer către nodul descendent (m_pNode) şi matricea de localizare (m_M) a acestuia faţă de nodul părinte, care este o matrice 4*4 (pentru transformări de coordonate în spaţiul tridimensional omogen).

În cursul operaţiei de etichetare a nodurilor grafului se creează un vector de pointeri la noduri, ordonaţi după eticheta acestora (obiectul global NodeArray, din clasa CNodeArray), care memorează, în fiecare poziţie i, pointerul la nodul cu eticheta i. Operaţia de etichetare a nodurilor grafului se execută în fiecare din procesoarele componente ale sistemului şi ea asigură corelaţia dintre implementarea locală a bazei de date (pointerii la nodurile acesteia) şi controlul global al traversării (etichetele unice ale nodurilor).

```
class CNodeArray : public CObArray
{
public:
    CNodeArray(){ }
    CNode* GetAt ( int index );
    int Add (CNode& o);
    ~CNodeArray(){ }
};
```

Vectorul NodeArray permite startarea operaţiei de căutare în graf, începând cu un nod dat prin eticheta lui. Pentru aceasta se apelează funcţia:

```
CNode* pNode = NodeArray.GetAt(label),
```

pentru a obţine pointerul la nodul de etichetă dată (label).

În figura 1 se dă graful unei bazei de date, cu nodurile numerotate (etichetate) corespunzător, printr-o parcurgere în adâncime a grafului.

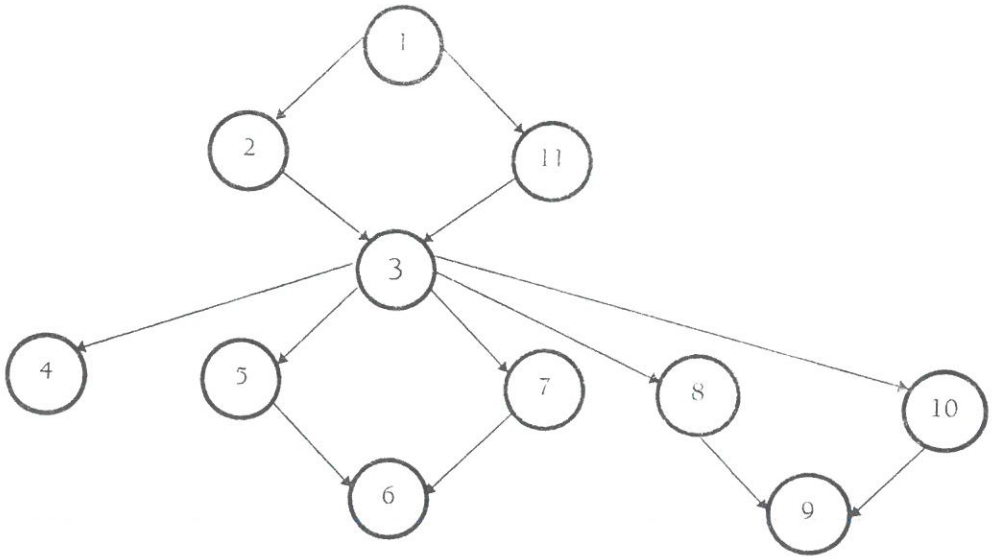

Figura 1. Etichetarea nodurilor grafului bazei de date

2.2. Volumul de extensie al nodurilor.

Volumul de extensie al unui nod este un obiect din clasa CExtension, definită astfel: el precizează un paralelipiped dreptunghic, definit în sistemul de referință local (al nodului), în care sunt incluse toate punctele obiectului descris de clasa respectivă.

```
class CExtension : public CObject
{
private:
    C3DPoint pm;
    C3DPoint pM;
public:
    CExtension() { }
    CExtension(CExtension& c);
    C3DPoint Getm() {return pm;}
    C3DPoint GetM() {return pM;}
    ~CExtension(){ }
};
```

Volumul de extensie al fiecărui nod se poate calcula după ce a fost creată structura internă a bazei de date. La încărcarea datelor de intrare, volumul de extensie se poate calcula numai pentru nodurile frunză, care conțin coordonatele (în sistemul de referință local) ale tuturor punctelor care aparțin obiectului. Pentru celelate noduri, volumul de extensie se calculează din volumele de extensie ale tuturor nodurilor succesoare ale acestuia.

Pentru aceasta, este necesară o altă parcurgere a grafului și anume o parcurgere în lărgime (breadth-first-search) care determină pentru fiecare nod, valorile de maxim și minim ale volumului lui de extensie, din valorile corespunzătoare ale tuturor succesorilor lui și calculează valoarea obiectului m_Ext, membru al clasei CNode.

3. Structurile de date asociate operației de traversare a bazei de date

Toate clasele de obiecte grafice, definite într-o bază de date orientată pe obiecte, sunt derivate dintr-o singură clasă de bază, ca rădăcină a ierarhiei de clase [2]. Această clasă rădăcină prevede un număr de capabilități utile pentru toate clasele derivate din ea, printre care cea mai importantă este localizarea bazei de date în sistemul de referință univers. Nodurile frunză ale grafului (nodurile care au succesori) descriu obiecte grafice tridimensionale, reprezentate prin suprafața de frontieră.

Algoritmul de traversare a bazei de date grafice este un algoritm de tipul căutare în adâncime (depth-first-search) în graful aciclic direcționat de reprezentare a bazei de date. [3]. Acest algoritm constă din căutarea tuturor drumurilor în graf. de la nodul rădăcină până la fiecare nod frunză, pentru obținerea obiectelor grafice vizibile.

Algoritmul de traversare a bazei de date (TDB) începe cu expandarea nodului inițial (nodul rădăcină) și generarea succesorilor acestuia. În fiecare pas următor, TDB selectează cel mai recent nod generat. Dacă acest nod nu are succesori (este nod frunză), se execută operațiile de

generare a imaginii obiectului instanțiat, după care TDB se reîntoarce și expandează un nod diferit.

Pentru execuția eficientă a operației de traversare a bazei de date într-o arhitectură paralelă se creează mai multe structuri de date, așa cum vor fi descrise în continuare.

3.1. Stiva de stare a traversării TSS

În fiecare pas al traversării, alternativele neexplorate trebuie să fie memorate într-un spațiu de stări al traversării. Acest spațiu de stări poate fi reprezentat eficient printr-o structură de date de tip stivă, stiva TSS. Deoarece dimensiunea stivei crește linear cu adâncimea arborelui de căutare în graf, cantitatea de memorie cerută de reprezentarea stivei este scăzută. Stiva TSS conține mulțimea de date care se partiționează și se distribuie între procesoarele componente și anume, spațiul de traversare a bazei de date.

Stiva TSS se reprezintă ca un obiect din clasa CStackArray, clasă derivată din clasa CObArray din biblioteca MFC, pentru care se definesc operațiile de înscriere în stivă (**Push**) și de extragere (**Pop**) a unor obiecte din clasa CEntry.

```
class CStackArray : public CObArray
{
public:
    int GetSize() {return CObArray::GetSize();}
    CEntry* GetAt(int index);
    int Push(CEntry& o);
    CEntry* Pop();
    ~CStackArray();
};
```

Clasa CEntry definește obiectele membre ale vectorului CStackArray și conține ca date membre identificatorul unui nod în cursul traversării, (obiectul m_Id) și o matrice de localizare (obiectul c_M), care reprezintă matricea de localizare curentă a nodului în sistemul de referință univers .

```
class CEntry : public CObject
{
private:
    CMatrix     c_M[4][4];
    CUIntArray  m_Id;
public:
    CEntry() { }
    CEntry(CEntry& obj);
    CUIntArray* GetId()
        {return &m_Id;}
    CMatrix* GetMatrix()
        {return &c_M;}
    void SetId(CUIntArray& id);
    void SetMatrix(CMatrix m)
        { c_M = m;}
    ~CEntry();
};
```

3.2. Identificatorul unui nod

Identificatorul unui nod pentru traversare (obiectul m_Id din clasa CEntry) este un vector de numere întregi (reprezentat printr-un obiect din clasa CUIntArray, clasă definită în biblioteca MFC) care reprezintă etichetele nodurilor drumului parcurs, începând cu nodul rădăcină, până la nodul respectiv, inclusiv. El definește, așadar, un nod împreună cu drumul în graf prin care este accesat.

De exemplu, pentru nodul 7, identificatorul este vectorul (1 2 3 7), când este atins printr-un drum ce trece prin nodul 2 .

3.3. Matricea de localizare curentă a unui nod

Matricea de localizare curentă a unui nod (obiectul c_M din clasa CEntry) este o matrice de localizare în sistemul de referință univers a nodului, de dimensiune 4*4 și ea reprezintă produsul matricilor de localizare a tuturor nodurilor parcurse pe drumul în graf.

4. Operații asociate traversării bazei de date

Pe parcursul traversării bazei de date se execută două operații importante, care au ca scop creșterea eficienței traversării și anume:

- operația de concatenare a transformărilor de localizare a nodurilor;
- operația de retezare a unui nod (și a tuturor succesorilor lui), dacă nodul este sigur invizibil.[4]

4.1. Operația de concatenare a transformărilor de localizare a nodurilor

Pentru eliminarea unor calcule inutile de localizare repetată a unui nod, fiecare obiect memorat în stiva de stare a traversării (obiect din clasa CEntry), conține matricea de localizare curentă a nodului în sistemul de referință univers (obiectul c_M, membru al clasei CEntry), a cărui valoare depinde de matricile de localizare a tuturor nodurilor parcurse pe drumul în graf, definit de identificatorul m_Id.

Operațiile cu matrici în sistemele grafice sunt, de cele mai multe ori, operații concatenate prin postmultiplicare (multiplicare la dreapta) [5]. Pentru fiecare nod, matricea de localizare curentă se calculează prin produsul dintre matricea de localizare curentă a nodului părinte și matricea de localizare a nodului față de acesta.

Fie pentru un punct de observare dat, matricea de transformare de observare V, calculată o singură dată pentru o imagine generată.

Pentru nodul rădăcină, matricea de localizare curentă are valoarea V .

Pentru nodul succesor următor (nodul 2), localizat față de nodul rădăcină printr-o matrice de localizare M_2, matricea de localizare curentă este: $V * M_2$.

În stiva TSS se introduce (printr-o operație Push) un obiect din clasa CEntry, care are datele membre de valori: $c\_M = V*M_2$;

m_Id = (1 2).

La atingerea unui nod frunză, matricea de localizare curentă conține toate transformările (concatenate) care trebuie să fie aplicate punctelor obiectului pentru transformarea din sistemul de referință local, în sistemul de referință observator.

De exemplu, pentru prima instanțiere a nodului 6, matricea de localizare curentă are valoarea:

$c\_M = V*M_2*M_3*M_5*M_6$

La generarea imaginii obiectului astfel instanțiat, vectorul fiecărui punct (P) este înmulțit cu matricea c_M și se obține vectorul P_o, în sistem de referință observator:

$P_o = V*M_2*M_3*M_5*M_6 *P$;

Această concatenare de matrici realizează ordinea corectă a transformărilor (mai întâi localizarea punctului în sistemul de referință univers, apoi transformarea de observare).

4.2. Operația de retezare a unui nod și a tuturor succesorilor lui

Pentru executarea acestei operații, înainte de a introduce un nod în stiva de stare a traversării, se face o transformare a volumului de extensie în sistemul de referință observator, prin înmulțirea matricii de localizare curentă (c_M) cu vectorii pm și pM (obiecte din clasa C3DPoint) care definesc volumul de extensie al nodului curent. Se obțin vectorii

pmo = c_M * pm;

pMo = c_M * pM;

care definesc volumul de extensie a unui nod în coordonate observator.

Aplicând algoritmul de excludere a acestui volum, ca test de invizibilitate sigură, se ia decizia dacă nodul este potențial vizibil și, în această situație, se înscrie în stiva TSS o intrare care conține identificatorul nodului; dacă nodul este sigur invizibil se abandonează traversarea lui, deci, și a tuturor succesorilor lui.

Inițializarea traversării bazei de date constă în expandarea nodului rădăcină. Pentru aceasta, fiecare nod succesor al acestuia este introdus în stiva TSS, prin intermediul unei intrări de tipul CEntry, care conține identificatorul și matricea curentă de localizare a nodului. Funcția InitTDB() realizează aceste operații:

```
void InitTDB(CNode* pRoot,CMatrix* pV)
{
int size = pRoot->GetChildArraySize();
CChildArray* pChildArray =
      pRoot->GetChildArray();

for (int i=0;i<size;i++)
{
CChild* pChild = pChildArray->GetAt(i);
CNode* pNode = pChild->GetNode();
CMatrix* pM = pChild->GetMatrix();
if (NodeIsVisible(pNode,pV,pM))
{
      CEntry* pEntry = new CEntry;
      CUIntArray* pId = pEntry->GetId();

      UINT label = pRoot->GetLabel();
      pId->Add(label);
```

```
        label = pNode->GetLabel();
        pId->Add(label);
        pEntry->SetMatrix((*pV)*(*pM));
        TSS.Push(*pEntry);
    }
  }
}
```

Pentru exemplul din figura 1, în acest moment stiva TSS conține două intrări (pointeri la obiecte CEntry) care, la rândul lor, conțin identificatorii nodurilor succesoare ale rădăcinii, vectorii (1 2) și (1 11).

După operația de inițializare, funcția de traversare a bazei de date TDB() extrage câte un nod din stiva TSS, atâta timp cât stiva nu este goală. Pentru fiecare mod extras din stivă, se generează imaginea obiectului, dacă nodul este nod frunză sau se expandează, dacă este nod intermediar. Pentru expandarea unui nod, se identifică toți descendenții acestuia; pentru fiecare nod descendent se calculează matricea de localizare curentă și volumul de extensie în sistemul de referință observator. Dacă nodul descendent este potențial vizibil, se introduce în stiva TSS o intrare (obiect CEntry) care conține identificatorul lui și matricea de localizare curentă.

Operația de inițializare a traversării se execută într-un singur procesor al arhitecturii paralele, iar funcția de traversare a bazei de date TDB se execută în toate procesoarele, cu posibilitatea de distribuire între acestea a stivei de stare a traversării, stiva TSS.

Fiecare procesor își întreține stiva sa locală a stărilor neexplorate, asupra căreia efectuează traversarea. Când stiva unui procesor devine vidă, el cere o stare neexplorată din stiva unui alt procesor. Când toate procesoarele au terminat de traversat spațiul de traversare (au atins nodurile frunză grafului), traversarea este terminată.

Traversarea grafului bazei de date produce un arbore de traversare care, pentru exemplul considerat, este dat în figura 2.

În nodurile arborelui sunt trecute numai etichetele nodurilor grafului expandate, dar rezultatul instanțierii unui obiect grafic depinde nu numai de clasa de obiecte grafice, descrise de nodul respectiv, ci și de toate clasele predecesoare acestuia, deci de drumul parcurs în graf (și în arborele de traversare al acestuia).

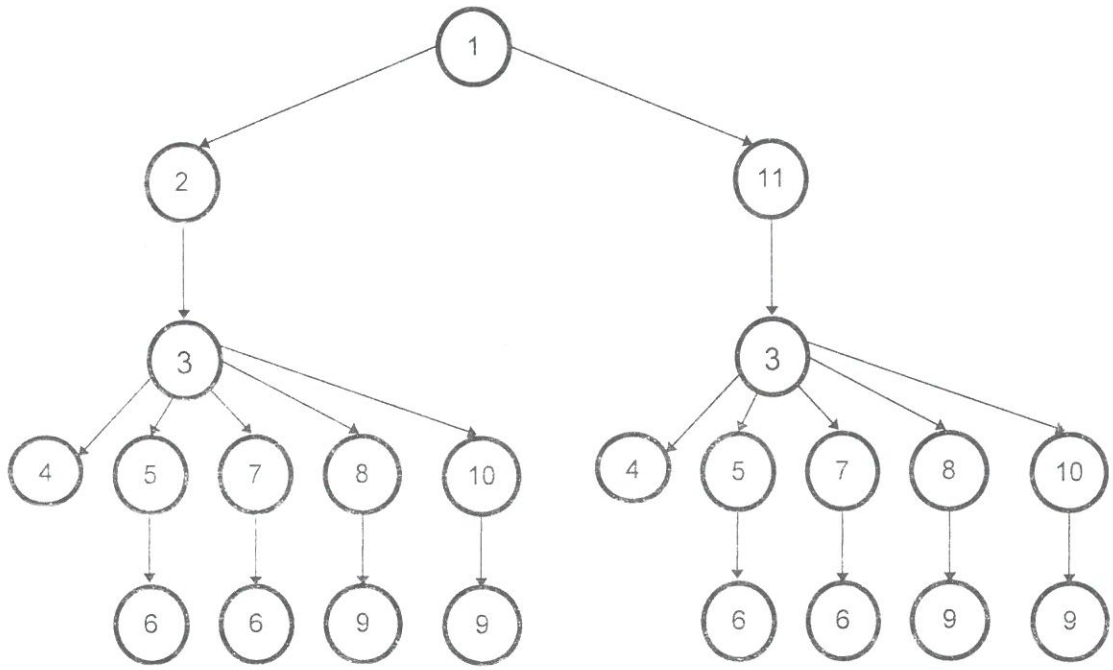

Figura 2. Arborele de traversare a grafului bazei de date.

Arborele de traversare are ca noduri terminale toate obiectivele grafice instanțiate, pentru care în algoritm se execută operația de generare a imaginii obiectului.

5. Concluzii

Sistemele de sinteză de imagine în timp real sunt realizări tipice de sisteme încorporate (embedded), cu cerințe de execuție în timp real, cerințe care se asigură prin intermediul procesării paralele, în

arhitecturi adecvate execuției eficiente a algoritmilor de sinteză de imagine. Un aspect important al proiectării corelate (co-design) hardware-software pentru astfel de sisteme îl reprezintă structurarea corespunzătoare a bazelor de date grafice, care să permită partiționarea dinamică a spațiului de traversare între procesoarele componente ale sistemului.

Bibliografie

1. **IONESCU, F.**: Structuri paralele pentru sinteza de imagine în timp real. Teză de doctorat, București, 1996.
2. **BRUNO, G., CASTELLA, A., AGARWAL, R.**: Object-Oriented Modelling for Simulation & Development of Large-Scale Systems, International Training Equipment Conference and Exibition Proceedings, The Hague, 1994.
3. **IONESCU, F.**: Baze de date grafice orientate pe obiecte. În: Revista Română de Informatică și Automatică, vol.6, Nr. 1, 1996.
4. **KALAY, Y.E.**: Modeling Objects and Environments, John Wiley & Sons, New York, 1989.
5. **FOLEY, J.D. , VAN DAM, A.**: Fundamentals of Interactve Computer Graphics, Addison-Wesley, Reading, MA,1982.