# BAZE DE DATE GRAFICE ORIENTATE PE OBIECTE

**ing. Felicia Ionescu**

*Megasys SRL, Bucureşti*

**Rezumat**: Lucrarea prezintă modelarea obiectelor lumii reale într-o bază de date grafică, creată pe conceptele orientării pe obiecte: o colecţie mare de componente corelate între ele sunt organizate ierarhic, ceea ce reduce cantitatea de date şi măreşte integritatea acesteia. Tehnici adiţionale, cum sunt moştenirea atributelor şi instanţierea, sunt adoptate din limbajele de programare orientate pe obiecte, ca de ex. C++.
Scopul principal al acestei modelări ierarhice este creşterea vitezei de administrare a bazelor de date grafice, astfel încât să se obţină performanţe în generarea de imagini în timp real.

**Cuvinte-cheie**: baze de date grafice, modelarea orientată pe obiecte, structurare ierarhică, imagine generată pe calculator.

## 1. Introducere

Atât în domeniul sintezei de imagine în timp real, cât şi în alte domenii de grafică pe calculator, prelucrarea şi prezentarea imaginilor grafice de către calculator se poate face numai prin prelucrări asupra unor structuri simbolice, care reprezintă fenomenele şi obiectele naturale [5]. Tehnica numită *modelare* permite reprezentarea realităţii fizice în memoria calculatorului sub forma unor structuri simbolice, pe care calculatorul le poate manipula, transforma, reda pe display sau poate efectua operaţii de analiză asupra lor.

În sinteza de imagine în timp real, modelarea fenomenelor şi a obiectelor reprezintă un proces off-line procesului de sinteză de imagine propriu-zis : o colecţie de obiecte modelate, stocate sub forma unor baze de date grafice este încărcată şi prelucrată în timp real.

Algoritmul general de sinteză de imagine constă din parcurgerea (traversarea) bazei de date grafice pentru extragerea obiectelor vizibile şi generarea imaginii acestora.

Eficienţa acestei operaţii depinde în mare măsură de modul de organizare a bazei de date grafice.

Descrierea compactă a obiectelor tridimensionale complexe (ca, de exemplu, prin metoda triangularizării, [2] structurarea ierarhică şi reprezentarea obiect-orientată a bazelor de date grafice [3], [6], [8], reprezintă aspectele cele mai importante care trebuie să fie avute în vedere pentru obţinerea de performanţe de timp real în sinteza de imagine.

## 2. Reprezentarea structurii unei baze de date grafice orientată pe obiecte

O bază de date grafică orientată pe obiecte reprezintă o metodă de modelare a scenei de redare sub formă de obiecte grafice, organizate într-o structură ierarhică, care permite abstractizarea datelor şi moştenirea atributelor între diferite niveluri ierarhice de obiecte. Structura ierarhică a bazei de date se reprezintă sub forma unui graf aciclic direcţionat [7] , B(O, D), în care: $O = \{o_i \mid 1 \le i \le o, o = |O|\}$ - este mulţimea nodurilor grafului, care reprezintă clase de obiecte grafice $D = \{d_{i,j} = (o_i,o_j) \mid 1 \le i,j \le o\}$ - este mulţimea arcelor în graf.

Un arc $d_{j,j} = (o_i,o_j)$ reprezintă dependenţa prin derivare a clasei de obiecte $o_j$ de la clasa de obiecte $o_i$.

Fiecare nod frunză al grafului B reprezintă o clasă de obiecte grafice şi conţine descrierea geometrică şi topologică a unui obiect tridimensional, reprezentat prin suprafaţa de frontieră precum şi atributele asociate diferitelor elemente ale obiectului tridimensional [6].

Un obiect dintr-o clasă de obiecte reprezentată de un nod frunză al grafului este instanţierea obţinută în cursul traversării bazei de date, instanţiere care adaugă la atributele clasei respective, atribute moştenite de la clasele predecesoare acesteia.

Un nod intermediar din graful de reprezentare al bazei de date reprezintă o clasă de obiecte grafice care moşteneşte de la predecesorii săi atribute (de localizare şi de redare) şi care transmite, la rândul ei, aceste atribute (şi altele, proprii clasei respective) către toate clasele derivate.

În fiecare clasă de obiecte grafice, reprezentată de un nod (frunză sau intermediar), se pot defini atribute şi metode (funcţii).

**Atributele unei clase de obiecte grafice** se împart în trei categorii:

- atribute de redare, care definesc caracteristici de redare a obiectelor, cum ar fi proprietăţi de material, proprietăţi de selecţie a suprafeţelor etc;

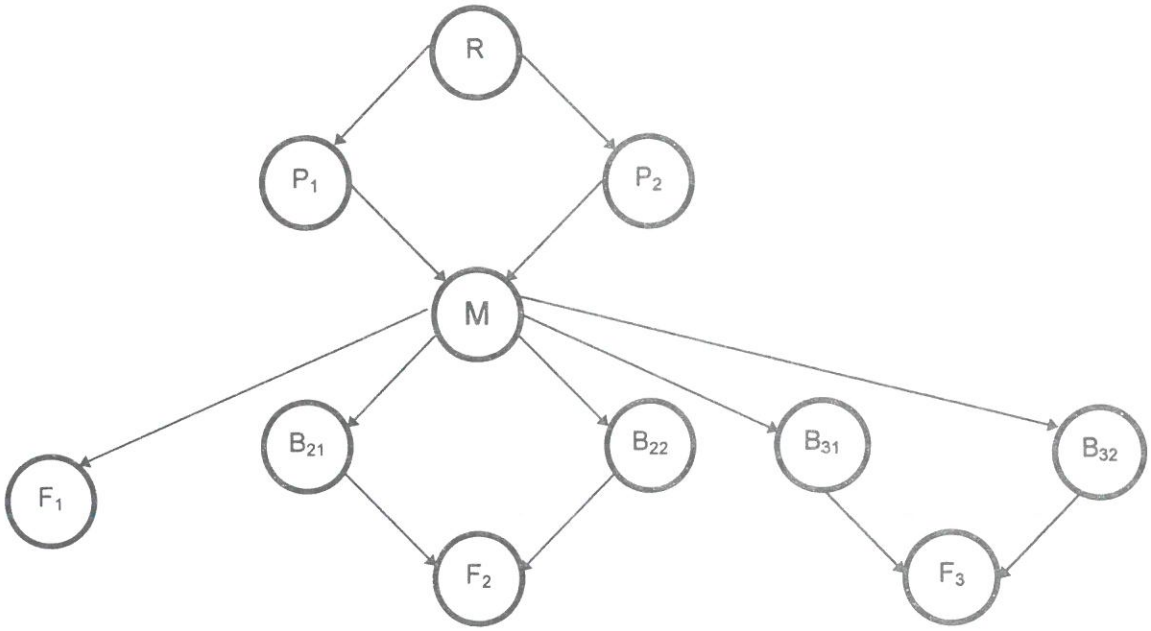


**Figura 1**. Clase de obiecte grafice într-o bază de date grafică

• atribute de localizare, care definesc poziţia relativă a nodului curent faţă de nodul părinte (deci a clasei curente faţă de clasa din care este derivată), sub forma unei matrici de transformare 4*4 (în reprezentarea în coordonate omogene);

• atribute de extensie, care definesc volumul de extensie al obiectului din nodul curent, în sistemul de referinţă local.

**Metodele definite într-o clasă de obiecte grafice** se referă la modul de compunere a atributelor clasei curente cu atributele clasei de bază.

Pentru atributele de redare, metoda definită poate specifica înlocuirea atributului moştenit cu un atribut redefinit de redare sau compunerea lor.

Pentru atributele de localizare, metoda defineşte modul de concatenare a matricii de transformare a nodului curent cu matricea de transformare moştenită de la nodul părinte: preconcatenare, postconcatenare sau înlocuire.

Fiind proiectată prin tehnica de modelare pe obiecte, în operaţiile executate în baza de date grafică se pot identifica analogii cu comportamentul claselor şi a obiectelor din limbajele de programare orientate pe obiecte, dar şi diferenţieri faţă de acesta [7].

Pentru fixarea noţiunilor utilizate în continuare vom descrie o structură particulară a unei baze de date, reprezentată în figura 1.

Nodul $F_1$ reprezintă o clasă de obiecte grafice care conţine descrierea, să presupunem, a părţii fixe (caroseria) unui vehicul.

Nodul $F_2$ reprezintă descrierea unei părţi mobile (roată-faţă) a vehiculului, care va fi instanţiat în două obiecte cu poziţii diferite, prin moştenirea de atribute de localizare, diferite de la clasele de bază $B_{21}$, respectiv, $B_{22}$ .

Similar, nodul $F_3$ reprezintă descrierea unei alte părţi mobile (roată-spate) a vehiculului, care va fi instanţiată în 2 obiecte, cu poziţii diferite prin moştenirea de atribute de localizare diferite de la clasele de bază $B_{21}$ şi, respectiv, $B_{22}$ .

Clasa de obiecte M reprezintă descrierea vehiculului, reprezentat prin asocierea tuturor obiectelor din clasele derivate din aceasta.

Nodul rădăcină R are 2 clase derivate $P_1$ şi $P_2$ care, prin instanţiere, vor produce 2 obiecte din clasa M (vehicul), cu poziţii diferite în funcţie de atributele de localizare conţinute în clasele $P_1$ şi, respectiv, $P_2$,.

***Baza de date grafică reprezintă o ierarhie de clase de obiecte în care o clasă derivată moşteneşte de la clasa de bază atribute de localizare şi de redare.***

Spre deosebire, însă, de clasele de obiecte din limbajele de programare, în ierarhia de clase a bazei de date grafică nu se admit moştenirile multiple : la instanţierea unui obiect dintr-o clasă dată se moştenesc numai atributele provenite de la toate clasele de bază ierarhic aflate pe un singur drum de la nodul rădăcină către nodul respectiv.

Ca şi ierarhia de clase a limbajelor de programare, ierarhia de clase de obiecte grafice ale bazei de date este exprimată sub forma unui *graf aciclic direcţionat*: nu se admit cicluri în graful bazei de date, cu alte cuvinte, o clasă nu poate fi clasă de bază pentru o alta care este derivată (direct sau indirect) din aceasta.

Oricare atribut conţinut de o clasă de obiecte poate fi redefinit într-o clasă derivată.

De exemplu, clasa M poate defini ca atribut de redare valoarea tranparenţei t a unei suprafeţe la o valoare $t = t_M$ .

Dacă în clasele derivate nu se redefineşte valoarea transparenţei t, atunci redarea tuturor obiectelor instanţiate din clasele derivate din clasa M vor avea valoarea transparenţei $t = t_M$. Dacă o clasă derivată (să spunem clasa $F_2$ ) redefineşte atributul t, prin $t = t_{F2}$, atunci toate obiectele instanţiate din clasa $F_2$ vor avea ca valoare a atributului t noua valoare redefinită $t_{F2}$ .

Fiecare clasă de obiecte din categoria nodurilor frunză ale grafului bazei de date defineşte un obiect tridimensional într-un sistem de referinţă propriu (local) al clasei.

Localizarea unui obiect în sistemul de referinţă univers se obţine prin concatenarea transformărilor de coordonate (descrise ca atribute de localizare ale claselor prin matrici de transformare), ale tuturor claselor de bază ierarhic superioare întâlnite în cursul parcurgerii unui drum în graf de la nodul rădăcină pînă la nodul care descrie clasa respectivă.

Imaginea grafică generată pentru o bază de date grafică B, pentru o poziţie de observare dată, reprezintă imaginea tuturor obiectelor instanţiate prin traversarea bazei de date.

***Se defineşte ca traversare a unei bazei de date grafice operaţia de parcurgere a tuturor drumurilor existente de la nodul rădăcină pînă la fiecare nod frunză al grafului bazei de date.***

Pentru exemplul dat, traversarea bazei de date înseamnă parcurgerea drumurilor:

$(R, P_1, M, F_1)$

$(R, P_1, M, B_{21}, F_2)$

$(R, P_1, M, B_{22}, F_2)$

$(R, P_1, M, B_{31}, F_3)$

$(R, P_1, M, B_{32}, F_3)$

$(R, P_2, M, F_1)$

$(R, P_2, M, B_{21}, F_2)$

$(R, P_2, M, B_{22}, F_2)$

$(R, P_2, M, B_{31}, F_3)$

$(R, P_2, M, B_{32}, F_3)$

S-au obţinut, prin traversare, 10 obiecte (2 obiecte din clasa $F_1$ , 4 obiecte din clasa $F_2$ şi 4 obiecte din clasa $F_3$ ).

Obiectele dintr-o clasă dată au aceeaşi topologie definită în clasa respectivă, dar atributele geometrice şi atributele de redare sunt definite prin compunerea tuturor atributelor întâlnite în parcurgerea drumului de la nodul rădăcină până la obiectul instanţiat, conform metodelor definite în toate clasele din acest drum.

Se pot remarca avantajele structurării ierarhice a bazei de date : în loc să fie descrise 10 obiecte cu topologia, atributele geometrice şi atributele de redare pentru fiecare obiect, s-au creat doar 3 clase de obiecte care descriu atribute topologice care nu se vor modifica în cursul traversării, atribute geometrice descrise într-un sistem de referinţă local şi, eventual, atribute de redare care vor redefini atributele de redare definite în clasele ierarhic superioare.

Din punct de vedere al localizării în spaţiu, fiecare nod al grafului bazei de date (deci fiecare clasă de obiecte grafice ) descrie o regiune limitată ca volum, într-un sistem de referinţă local.

## 3. Crearea structurii ierarhice a bazei de date

Structura ierarhică pe clase de obiecte a bazei de date grafice este o structură de date care se creează în memoria sistemului de sinteză de imagine la încărcarea bazei de date, dintr-un format extern care poate avea diferite reprezentări, în funcţie de standardul selectat.

Crearea acestei structuri este un proces care nu este considerat ca un proces de timp real, dat fiind că această structură se creează o singură dată, la lansarea programului, şi rămâne rezidentă în memorie pentru toate operaţiile ulterioare de generare de imagine în timp real.

Din această cauză, modulul de creare a structurii bazei de date grafice are ca cerinţă importantă de implementare crearea unei structuri cît mai eficiente din punct de vedere al traversării ei, care se execută în timp real, pentru fiecare imagine generată.

În acest scop, la crearea structurii bazei de date se execută calcule care depun în nodurile grafului cât mai multe informaţii necesare în cursul traversării. De exemplu: se memorează în fiecare nod numărul de arce incidente spre exterior, atribute de extensie ale nodului, etc.

Un nod (clasă de obiecte grafice ) în baza de date este descris generic de o clasă a programului, clasa **CBlock**. Pentru definirea acestei clase se definesc mai întîi următoarele clase şi structuri, care pot fi urmărite în continuare.

**CMyObject**

Este o clasă a programului din care se vor deriva toate clasele care necesită accesul la un obiect din clasa **Object** a compilatorului (Borland 3.1), din care este de altfel derivată. Clasa CMyObject redefineşte funcţiile virtuale pure ale clasei abstracte Object, ceea ce va permite instanţierea ulterioară a unor obiecte din clasele care se vor deriva din CMyObject.

```
class CMyObject:public Object
{
    public:
        // Define pure Virtual Members
        classType isA() const
        { return 0; }
        char _FAR *nameOf() const
        { return 0; }
        hashValueType hashValue()
        const
        { return 0; }
        int isEqual(const Object _FAR&)
        const { return 0; }
        void printOn(ostream& pstream)

        const{ }
};
```

**CPoint**

Este clasa care descrie componentele şi comportarea unui punct în spaţiul tridimensional. Atributele obiectelor din clasa CPoint sunt coordonatele acestuia, iar metodele sunt constructorii ( de iniţializare şi de copiere) şi funcţiile de redefinire a operatorilor. Destructorul este implicit destructorul clasei Object, din care CPoint este derivată indirect, prin clasa CMyObject.

```
class CPoint:public CMyObject
{
    private:
        // Data Members
        float x,y,z;
    public:
        friend class CPointArray;
        // Constructors
        CPoint(CPoint& p)
            {x=p.x; y=p.y; z=p.z;}
        CPoint(float xi,float yi,float zi)
            {x=xi; y=yi; z=zi;}
        // Methods
        int operator==(CPoint);
        CPoint operator=(CPoint);
        CPoint operator+(CPoint);
};
```

**CPointArray**

Este o clasă derivată din clasa **Array**, a compilatorului; ea descrie un vector de pointeri către obiecte din clasa CPoint şi reprezintă lista de vârfuri (puncte în spaţiul tridimensional) ale unui obiect .

CPointArray conţine metode de adăugare şi ştergere de puncte în lista de puncte (vector de puncte), constructori de iniţializare şi copiere, iar destructorul este implicit destructorul clasei Array, din care este derivată.

```
class CPointArray:public Array
{
    private:
        // Data Members
    public:
        // Constructor


    CPointArray(intmaxsize):Array(max){}
        // Methods
        void AddToPointArray(CPoint&
p);
        int SearchInPointArray(CPoint&
p);
        CPoint* CPointArray::GetAt(int
indx);
        int GetSize()
        { return lastElementIndex+1;}
        void
CopyPointArray(CPointArray &,

                SInsert*);
        void ReadPointArray(fstream &);
};
```

**CPolygon**

Este o clasă derivată din clasa CMyObject; ea descrie atributele unei suprafeţe poligonale a unui obiect tridimensional (număr de vârfuri, culoare, textură, transparenţă etc.) şi un vector de indexuri (VertList ) într-un vector din clasa CPointArray al obiectului tridimensional din care face parte suprafaţa respectivă.

```
class CPolygon:public CMyObject
{
    private:
        // Data Members
        int texture;
        // Texture
        int colour;
            // colour
        int t;
        // tranparence
        int VertList[maxV];        //
Vertex List
        int VertNum;
    // Vertex Number
    public:
        // Constructors
        CPolygon(int c,int ti)
        {       texture = -1;      colour
= c;
                t = ti;    VertNum = 0;}
```

```
                CPolygon(CPolygon& p);
                // Methods
                void
ReadPolygon(int,fstream&,int);
                void AddIndex(int );
                int     GetVertNum()  {  return
VertNum;}
    };
```

**CPolyArray**

Este o clasă derivată din clasa **Array** a compilatorului şi conţine un vector de pointeri către obiecte din clasa CPolygon, descriind astfel toate suprafeţele poligonale care alcătuiesc un obiect tridimensional.

```
    class CPolyArray:public Array
    {
        private:
                // Data Members
        public:
                // Constructor
                CPolyArray(int
max):Array(max){}
                // Methods
                void
AddToPolyArray(CPolygon& s);
                CPolygon*
CPolyArray::GetAt(int ind);
                int  GetSize()
                {       return
lastElementIndex+1;}
                void
CopyPolyArray(CPolyArray& );
                void
ReadPolyArray(int,fstream&, int);
    };
```

**CGeometry**

Este clasa care face conexiunea între cele 2 tipuri de date de descriere a unui obiect tridimensional : datele geometrice, descrise printr-un obiect de tipul CPointArray, şi datele topologice, descrise printr-un obiect de tipul CPolyArray.

Clasa CGeometry defineşte un obiect sortabil după un criteriu ales; pentru aceasta se derivează din clasa **Sortable** a compilatorului C++ şi redefineşte toate funcţiile virtuale pure ale acestei clase, ceea ce permite introducerea obiectelor din clasa CGeometry înt-un vector sortat după un identificator (Id ). Clasa CGeometry defineşte un nod frunză din graful bazei de date.

```
    class CGeometry:public Sortable
    {
        private:
                // Data Members
                int
                t ype;
                SInsert                 Id;
                CPointArray     PointArray;
                CPolyArray
        PolyArray;
        public:
                // Constructors
                CGeometry(int t,SInsert& s):

        PointArray(maxP),PolyArray(maxP)
                {       Id  = s;  type = t; }
                CGeometry(CGeometry& g);
                // Methods
                CPointArray* GetVArray()
                {       return &PointArray;}
                CPolyArray* GetSArray()
                {       return &PolyArray;}
                int  GetType(){ return type;}
                void SetType(int t) { type = t;}
                void ReadGeometry(fstream&);
                void WriteGeometry(ofstream&);
                // Define pure Virtual Members
    };
```

**SInsert**

Este o structură care descrie atributele de localizare a unui nod fiu relativ la nodul părinte şi anume: punctul de inserţie al nodului fiu (xi, yi, zi), un vector de scalare (sxi, syi, szi) şi un unghi de rotaţie (rai), din care se calculează matricea de transformare de localizare.

```
    struct SInsert
    {
        char            name[40];
        int             layer;
        float           xi,yi,zi;
        float           sxi,syi,szi;
        float           rai;
    };
```

**CInsertArray**

Este o clasă care defineşte un vector de pointeri la structuri de tipul SInsert, deci prin acest vector se definesc toţi fiii unui nod din graf.

```
    class CInsertArray
    {
        private:
                int
        cursize;
                int
        maxsize;
                SInsert* si;
        public:
                // Constructors - Destructoe
                CInsertArray(int n);
                CInsertArray(CInsertArray&);
                ~CInsertArray();
                // Methods
                int GetSize(){ return (cursize);}
                SInsert* GetSi() { return si;}
                void addChild(SInsert &)
    };
```

**CBlock**

Este clasa care descrie un nod în graful bazei de date prin intermediul a 2 obiecte şi anume:

BlockGeometry este un obiect din clasa CGeometry şi conţine o descriere a unui obiect tridimensional (prin obiectele PointArray şi PolyArray ale clasei) numai pentru nodurile frunză ale grafului; pentru nodurile intermediare, acest obiect este vid.

BlockChilds – este un obiect din clasa CInsertArray care memorează pointerii către toate obiectele de tipul SInsert prin care se definesc atributele de derivare ale nodurilor fiu dintr-un nod părinte. Pentru nodurile frunză ale grafului acest obiect este vid, iar pentru un nod intermediar descrie toate arcele incidente spre exterior ale acestuia.

```
class CBlock: public CMyObject
{
    private:
            // Data Members
            SInsert
    Id;
            CInsertArray
    BlockChilds;
            CGeometry
    BlockGeometry;
    public:
            // Constructors
            CBlock(char *bname):
            BlockGeometry(0,Neutral),

    BlockChilds(maxInsert)
            { Id = Neutral;
              strcpy(Id.name,bname); }
            CBlock(CBlock &);
            // Methods
            SInsert* GetId() { return &Id;}
            int GetSArraySize() ;
            CInsertArray* GetBlockChilds()
             { return &BlockChilds; }
            CGeometry*GetBlockGeometry()
            { return &BlockGeometry; }
             void ReadBlock(fstream&);
             void WriteBlock(ofstream&);
};
```

## 4. Algoritmul de traversare a bazei de date grafice

Operaţia de traversare a bazei de date este asociată cu o structură de date de tip stivă - TSS (Stiva de Stare a Traversării) . În fiecare element al acestei structuri, se memorează informaţiile de stare a traversării (atribute de redare şi atribute de localizare), salvate în cursul traversării nivelurilor inferioare ale structurii ierarhice.

Operaţiile cu stiva TSS sunt operaţii tipice de stivă:

**push** - pentru salvarea unei stări;

**pop** - pentru refacerea stării.

Memorarea informaţiilor de traversare a fiecărui nod reprezintă o modalitate foarte eficientă de calcul a localizării nodurilor din structura ierarhică: fiecare nod este localizat relativ la părintele său; pentru calculul localizării se concatenează matricea de transformare a nodului cu matricea de transformare, moştenită de la părintele său, astfel că pentru fiecare nod se calculează o singură transformare, pentru fiecare drum care îl traversează.

La atingerea unui nod frunză se apelează o funcţie de generare a imaginii obiectului obţinut prin instanţierea clasei nodului respectiv. Funcţia de generare a imaginii obiectului conţine toate operaţiile necesare pentru obţinerea imaginii tuturor suprafeţelor poligonale care alcătuiesc obiectul, cu atributele de localizare şi de redare, calculate pe drumul pe care a avut loc instanţierea.

În cursul traversării, pentru fiecare nod, se calculează localizarea în sistemul de referinţă univers a volumului de extensie al nodului şi poziţia acestuia faţă de piramida de vizibilitate. Dacă este îndeplinită condiţia de excludere sigură a acestui volum de extensie, atunci se ***"reteazã"*** toate arcele incidente spre exterior ale nodului, deci nu se va mai instanţia nici un obiect din clasele de obiecte derivate din clasa de obiecte reprezentată de nodul respectiv.

Această modalitate de excludere din calcul a unui obiect sau grup de obiecte aflate în afara piramidei de vizibilitate, reprezintă un instrument foarte puternic de reducere a timpului de calcul în cursul generării imaginilor.

Algoritmul de traversare a grafului bazei de date este un algoritm de tipul ***parcurgere graf în adâncime (depth first search)***, dar diferă de algoritmul clasic DFS [1], prin aceea că un nod poate fi vizitat de mai multe ori, pe fiecare drum care porneşte din nodul rădăcină şi ajunge la un nod frunză descendent al acestuia.

Având aceste elemente precizate, algoritmul de traversare a grafului bazei de date se poate defini (în limbaj like-C) ca o procedură recursivă pentru un nod N:

```
void tdb (nod N)
{
      Concatenare matrici de localizare
      if ( volum de extensie == vizibil)
      {
        if ( N == nod frunză)
              Generare Imagine Obiect
              else      {
                push TSS;
```

```
                for (fiecare nod terminal w al
                tuturor arcelor incidente spre
                exterior ale nodului N )
                        tdb (w);
            pop TSS
            }
    }
}
```

Traversarea începe cu nodul rădăcină; orice nod, atins pe un drum de la nodul rădăcină, care este identificat ca fiind invizibil relativ la piramida de vizibilitate, este abandonat, împreună cu toți succesorii lui pe drumul respectiv.

El poate fi reluat și identificat ca fiind vizibil în traversarea pe un alt drum care pleacă din nodul radăcină ( ceea ce înseamnă o altă localizare a obiectului, care poate să fie vizibilă).

În exemplul bazei de date din figura 1, obiectul M instanțiat prin derivare din clasa $P_1$ poate să fie invizibil, dar instanțiat prin derivare din clasa $P_2$ poate să fie vizibil, pentru o anumită poziție a observatorului.

Algoritmul recursiv de traversare a bazei de date grafice reprezintă, în esență, întreaga aplicație de sinteză de imagine. Analiza lui, procedurile de paralelizare, partiționare și planificare a proceselor pentru implementerea într-o arhitectură de calcul paralel reprezintă aspecte importante pentru obținerea de performanțe de timp real ale sistemelor de sinteză de imagine.

## 5. Concluzii

În prezentul articol, s-a prezentat abordarea orientată pe obiecte a bazelor de date grafice destinate, în primul rând, sistemelor de sinteză de imagine în timp real, unde eficiența reprezentării obiectelor modelate și posibilitatea administrării rapide și descentralizate a acestora reprezintă cerințe primordiale, care asigură performanțe de timp real .

Aplicarea metodei de modelare orientată pe obiecte a bazelor de date grafice, metodă cunoscută sub numele de **OMT** (Object Modelling Technique), permite construirea modelului prin evedențierea proprietăților lui esențiale, care este o abstractizare precisă și concisă a comportării lui.

## Bibliografie


1. **AHO, A.V., HOPCROFT, J.E., ULLMAN, J.D.:**Data Structures and Algorithms, Addison-Wesley, 1983.
2. **AUERHAMMER, F.** :Voronoi Diagrams – A Survey of Fundamental Geometric Data Structure. În: ACM Computing Surveys, Vol. 23, No. 3, September 1991.
3. **BRUNO, G., CASTELLA, A., AGARWAL, R.:** Object-Oriented Modelling for Simulation & Development of Large-Scale Systems. În: Proc. of the International Training Equipment Conference and Exibition, Hague, 1994.
4. **ELLIS, M. A., STROUSTRUP, B.** : The annotated C++ Reference Manual, Addison-Wesley, 1990.
5. **FOLEY, J.D., Van DAM, A.** : Fundamentals of Interactive Computer Graphics, Addison-Wesley, Reading, MA,1982.
6. **KALAY, Y.E.** : Modelling Objects and Environments, John Wiley & Sons, New York, 1989.
7. **NARSINGH, D.** : Graph Theory with Applications to Engineering and Computer Science, Prentice-Hall, New Jersey, 1984.
8. **SAMET, H., WEBER, R.E.** : Hierarchical Data Structure. În: Second Image Symposium - Image Processing, Computer Generated Images, Technology and Applications, Paris, 1986.