Revista Română de Interacţiune Om-Calculator 3(1) 2010, 17-32

Integrarea scenelor 3D în aplicaţii grafice interactive

Titus Felix Furtună

Academia de Studii Economice din Bucureşti
Piaţa Romană, Nr. 8, 010374, Bucureşti
E-mail: titus@ase.ro

Rezumat. Realizarea modelelor geometrice tridimensionale şi utilizarea lor în aplicaţii informatice implică eforturi mari în producţia de software aplicativ. Un model conţine atât elemente geometrice complexe cât şi proprietăţi precum: culoare, iluminare, textură, material. Spre deosebire de o imagine 2D, un model tridimensional aduce cu el o precizie desăvârşită. Actualele produse de modelare 3D permit construirea, vizualizarea şi modificarea unor modele ca şi cum acestea ar fi obiecte reale. Dacă se doreşte însă vizualizarea unor astfel de modele într-un alt context aplicativ, diferit de cel de modelare, singura soluţie rămâne importul acestor modele şi prezentarea lor în contextul aplicaţiei gazdă. Astfel, un prim obiectiv al prezentului articol este integrarea scenelor 3D construite cu programe specializate în aplicaţii grafice scrise în C++ , C# şi Java într-un context dinamic interactiv folosind OpenGL şi Java 3D. Alegerea acestor limbaje a fost determinată de neta lor dominaţie pe piaţă în ceea ce priveşte cererea şi oferta de competenţe. Al doilea obiectiv este asigurarea interactivităţii cu obiectele unei scene 3D în contextul unor aplicaţii C++, C# sau Java. În articol este prezentată şi o soluţie de implementare pentru integrarea unei scene construite în 3D Studio Max în aplicaţii C# utilizând OpenGL.

Cuvinte cheie: aplicaţii grafice interactive, OpenGL, C#, modele tridimensionale

1. Introducere

În prezent există biblioteci scrise în diverse limbaje de programare (C, C++, Java) care facilitează importul unor scene 3D în aplicaţii grafice complexe. Iată câteva exemple de astfel de biblioteci: Collada, OpenNurbs, Truevision 3D, Blender 3D, Spring, Quake engine, Yenka 3D Importer, 3D Ajax etc. Majoritatea sunt dezvoltate pentru jocuri. Orice bibliotecă impune însă restricţii de utilizare, fie tehnice fie financiare şi pot fi utilizate intr-un context care nu întotdeauna asigură flexibilitate totală aplicaţiilor. În acest articol este prezentată o soluţie de programare concretă, deschisă, pentru dezvoltarea unei biblioteci proprii, având ca scop integrarea scenelor 3D în

aplicații grafice complexe. Aria tematică acoperită este cea a aplicațiilor de realitate virtuală, arie conectată la domeniul HCI, iar punctul de vedere abordat este cel al programatorului, al dezvoltatorului de software.

Programarea în limbaje evoluate precum C++, C# sau Java oferă programatorilor un control aproape total asupra aplicațiilor, de aceea acestea sunt preferate pentru dezvoltarea unor aplicații software complexe. În aceste limbaje pot fi scrise aplicații universale, care să ruleze în diverse domenii de activitate: economic, tehnic-ingineresc, medical etc., şi în care, de multe ori, este utilă includerea unor elemente de grafică 3D. Limbajul în sine, prin specificația standard oferă doar facilități pentru grafica 2D. De exemplu, în C++ sau în C# pot fi utilizate facilitățile grafice oferite de GDI (*Graphics Device Interface*)/GDI+, în timp ce în Java pot fi folosite facilitățile grafice oferite de Java2D. Pentru grafica 3D, fie se scrie cod în limbajul de bază implementând algoritmii necesari, cale dificil de urmat şi ineficientă, fie se utilizează specificații precum OpenGL, DirectX sau Java3D.

OpenGL este o specificație software de nivel scăzut dezvoltată de Silicon Graphics în scopul de a facilita realizarea unor interfețe programabile independente de platformă. Specifiația este simplă, flexibilă, concisă şi implementabilă pe platforme diferite. Primitivele OpenGL pot fi apelate sub diverse limbaje de programare, precum C++, Java, Fortran sau Ada [2,3].

Ca şi OpenGL, DirectX este o interfață programabilă de nivel scăzut, dezvoltată de Microsoft pentru platformele Windows. Grafica 3D este realizată sub DirectX de componenta Direct3D. Direct3D include începând cu versiunea 8 şi vechiul DirectDraw pentru redare grafică 2D. Direct3D este competitorul direct al OpenGL, fiecare produs având avantajele şi dezavantajele sale.

Java3D este o colecție de clase dezvoltate în limbajul Java, care formează o interfață de programare pentru grafica 3D (API – *Application Programming Interface*). Java 3D oferă programatorilor Java posibilitatea de a scrie applet-uri şi aplicații Java cu conținut grafic 3D interactiv. Comparativ cu alte API-uri 3D procedurale, DirectX sau OpenGL, care au fost proiectate pentru a optimiza la maximum performanțele şi a asigura un control total asupra aplicațiilor, Java 3D reprezintă o cale prin care un programator Java experimentat poate crea grafică 3D suficient de performantă, fără să fie necesar să cunoască toate detaliile implementărilor de nivel scăzut [10,6].

2. Utilizarea fişierelor text în salvarea scenelor 3D

Fişierele text pot fi o bună soluție pentru salvarea scenelor 3D în vederea utilizării lor în aplicații interactive scrise în diverse limbaje de programare. Toate limbajele de programare dispun de biblioteci cu funcții specializate în tratarea şirurilor de caractere, astfel că parsarea unui fişier text în conformitate cu o structură dinainte cunoscută a acestuia nu constituie o problemă pentru programatori. Exemple de astfel de fişiere text sunt fişierele *Wavefront Object* (OBJ), *Autodesk Scene Export File* (ASE) sau *Stanford Triangle Format* (PLY). În continuare va fi prezentat formatul OBJ care are o arie de răspândire mai largă.

Fişierele Wavefront Object (OBJ). Fişierele OBJ sunt utilizate pentru a salva scene 3D care conțin atât forme geometrice poligonale, cât şi curbe şi suprafețe curbe. Formatul OBJ a fost dezvoltat de către compania Wavefront Technologies şi a fost adoptat şi de către alți dezvoltatori de aplicații grafice 3D precum: 3D Studio Max, 3DPaintBrush, Blender, Maya, Mathematica, FreeCad etc. [5]

Obiectele geometrice poligonale sunt definite prin puncte, linii şi suprafețe. În structura unui fişier OBJ intră următoarele elemente: date de tip vertex, elemente geometrice, curbe şi suprafețe, conexiuni între curbe şi suprafețe, atribute de afişare/randare, comenzi generale.

Datele de tip vertex sunt utilizate pentru a specifica elemente de geometrie, texturare, normale la suprafețe, puncte de control pentru suprafețe, matrice de transformare.

Sintaxa elementelor de tip vertex este specificată în tabelul 1.

Tabelul 1. Sintaxa elementelor de tip vertex

v x y z w sau *v x y z*	Punct din geometria obiectului	(*x*, *y*, *z*, *w*) / (*x*, *y*, *z*) sunt coordonatele omogene/carteziene ale punctului. În cazul în care valoarea *w* nu este specificată, aceasta se consideră 1.0.
vn i j k	Normala într-un punct al unei	*i*,*j*,*k* reprezintă componentele normalizate ale vectorului normală la

	suprafețe poligonale sau curbe	suprafață intr-un punct
vt u v w	Textură la suprafață	*u*, *v*, *w* reprezintă coordonatele de mapare pentru un punct din textură. Elementul *w* apare la texturile 3D.
vp u v w	Punct de control în cadrul unei curbe sau a unei suprafețe curbe	*u*, *v*, *w* reprezintă parametrii curbei sau suprafeței pentru un punct de control

Aceste elemente apar în fişierul OBJ grupate pe categorii (*v*, *vn*, *vt*, *vp*). Ele sunt apoi referite în componența obiectelor geometrice din care fac parte printr-un număr întreg reprezentând numărul său de ordine. O suprafață poligonală va fi descrisă astfel:

$f\ v_1/vt_1/vn_1\ v_2/vt_2/vn_2\ ...\ v_i/vt_i/vn_i\ ...\ v_m/vt_m/vn_m$

unde *m* reprezintă numărul de puncte al suprafeței, iar (v_i, vt_i, vn_i) reprezintă un triplet prin care sunt specificate pentru un punct oarecare, *i*, din componența suprafeței, coordonatele geometrice, coordonatele de mapare şi normala la suprafață în punctul respectiv.

Obiectele geometrice sunt separate în grupuri. Un grup este marcat din punct de vedere sintactic astfel:

g name

Toate suprafețele definite după un astfel de marcaj aparțin unui singur grup cu numele *name*. În situația în care nu sunt salvate normalele la suprafețe, poate fi utilizat un marcaj asemănător celui de grup geometric, pentru a grupa suprafețele în aşa numitele grupuri *smooth*. Normalele la suprafețele din acelaşi grup smooth vor fi calculate ca şi cum acestea ar face parte din acelaşi plan. Marcajul este inserat astfel:

s int sau *s off*

unde *int* reprezintă grupul smooth din care fac parte suprafețele care urmează marcajului, iar *off* sau *int* = 0 se utilizează când nu se specifică un grup smooth.

Elementele de tip material sunt specificate prin fişiere adiționale cu extensia *.mtl*.

Marcajul pentru utilizarea unor anumite proprietăți material se face în felul următor:

usemtl name

cu semnificația că grupului geometric curent îi vor fi aplicate aceste proprietăți.

În fişierul MTL sunt detaliate apoi proprietățile de tip material care sunt marcate în fişierul OBJ cu numele *name*. Inceputul descrierii este făcut printr-un marcaj *newmtl name*. În tabelul 2 sunt prezentate formatele în care sunt specificate elemente de tip material.

Tabelul 2. Formatele în care sunt specificate elemente de tip material

ka r g b	Culoarea provenită din componenta ambientală a luminii
kd r g b	Culoarea provenită din componenta difuză
ks r g b	Culoarea provenită din componenta specular ă
illum	Specifică utilizarea sursei de lumină astfel: 0 - dezactivarea iluminării 1 - utilizarea doar a componentelor difuză şi ambientală 2 - utilizarea tuturor componentelor de lumină
Ns sh	Stabileşte coeficientul de strălucire (de la 0 la 128)
*map*_kd *texFile*	Specifică un fişier textură

3. Aplicații grafice interactive cu OpenGL şi Java3D

Integrarea obiectelor tridimensionale construite cu produse specializate în aplicații OpenGL sau Java3D implică utilizarea elementelor de interacțiune cu obiectele din scenă. Atât OpenGL cât şi Java3D au facilități puternice de selecție şi *picking*, permițând utilizatorului unei aplicații să identifice obiecte din scenă cu ajutorul mouse-ului. În timp ce selecția este o noțiune mai largă, de alegere a unor componente dintr-un întreg în diverse feluri, după diverse criterii, picking-ul se referă la selecția cu ajutorul mouse-ului. Picking-ul este aşadar o formă particulară de selecție.

3.1 *Picking*-ul şi selecţia în OpenGL

Mecanismul de *picking*-selecţie implementat în OpenGL permite selecţia obiectelor din scenă aflate în interacţiune cu un volum de vizualizare construit cu ajutorul mouse-ului. Selecţia nu se face direct asupra obiectelor tridimensionale, ci asupra primitivelor aflate în componenţa acestora şi care interacţionează cu volumul de vizualizare [2,3,4]. Modul de operare este următorul:

1. *Desenarea scenei* şi actualizarea *frame-buffer*-ului cadru în concordanţă cu conţinutul scenei.

2. *Intrarea în modul selecţie*. O aplicaţie OpenGL se poate afla la un moment dat în modul redare, modul selecţie sau modul *feedback*. Modul redare corespunde redării propriu-zise a scenei şi are ca efect modificarea *frame-buffer*-ului cadru. Modul selecţie nu modifică *frame-buffer*-ul cadru, în schimb, primitivele care sunt desenate în modul selecţie şi care interacţionează cu un volum de vizualizare specificat, produc înregistrări într-un *buffer* de selecţie. Comanda prin care se stabilesc modurile redare, selecţie sau *feedback* este:

```
GLint glRenderMode(GLenum mode);
```

unde *mode* poate fi una din constantele GL_RENDER (implicită), GL_SELECT, sau GL_FEEDBACK.

3. *Redesenarea scenei*. Redesenarea scenei în modul selecţie este precedată de definirea unui volum de vizualizare care să fie utilizat în selecţie şi de construirea unei stive de nume cu ajutorul căreia să poată fi identificate primitivele care interacţionează cu *buffer*-ul de selecţie. Construirea volumului de vizualizare pentru selecţie cu ajutorul mouse-ului mai este cunoscută şi sub denumirea de *picking*.

4. Părăsirea modului selecţie.

5. *Procesarea datelor* de selecţie returnate.

Picking-ul este procesul prin care se realizează selecţia cu ajutorul unui dispozitiv cu cursor (de exemplu mouse-ul). Mecanismul presupune construirea unei matrice speciale de *picking* care, impreună cu matricea de proiecţie, restricţionează desenarea la o mică regiune a *viewport*-ului apropiată de cursor. Selecţia este apoi declanşată printr-un simplu *click* pe mouse.

Crearea matricei de proiecţie care să restricţioneze desenarea la o regiune de ecran se realizează prin comanda:

```
void gluPickMatrix(GLdouble x, GLdouble y, GLdouble width,
     GLdouble height, GLint viewport[4]);
```

unde (*x*, *y*) reprezintă centrul regiunii (în coordonate ecran), *width* şi *height* reprezintă mărimea regiunii (tot coordonate ecran), iar *viewport* reprezintă marginile viewport-ului actual. Deoarece matricea creată prin comandă multiplică matricea curentă de pe stivă, ea trebuie precedată de un apel *glLoadIdentity*().

Stiva de nume constituie baza informației furnizate prin selecție. Pentru a crea o stivă de nume, mai întâi aceasta este inițializată prin comanda:

```
void glInitNames(void);
```

Comenzile de utilizare a stivei sunt:

```
void glPushName(GLuint nom); //Punerea unui nume pe stivă
void glPopName(void); // Extragerea unui nume de pe stivă
void glLoadName(GLuint nom); // Înlocuirea numelui din vârful
     stivei
```

Rezultatul selecției constă într-un tablou de selecție (vector de întregi) umplut cu înregistrări ale selecției. În modul selecție, o primitivă care intersectează volumul de vizualizare determină o astfel de înregistrare. Fiecare înregistrare de selecție memorată în tablou conține patru informații şi prin urmare, ocupă patru elemente în tablou:

- numărul de nume existente pe stivă la momentul generării înregistrării;
- valorile maxime şi minime pentru coordonatele *z* ale vârfurilor primitivelor intersectate de volumul de vizualizare (sunt valori cuprinse între 0 şi 1 multiplicate cu 2^{31}-1);
- conținutul stivei de nume după selecție.

La intrarea în modul selecție este inițializat un pointer către începutul tabloului de selecție. Ori de câte ori se face o înregistrare, se actualizează pointerul. La ieşirea din modul de selecție este furnizat numărul înregistrărilor efectuate prin comanda *glRenderMode*. Deasemenea, comanda va sterge stiva şi va reseta pointerul la stivă.

În continuare este prezentată o metodă standard de procesare a selecției, scrisă în C#.

```
private int procesareSelectie()
{
int hits; // variabila in care intorc numarul de primitive selectate
```

```
// stabilire bufer de selectie si numar maxim de primitive selectate
Gl.glSelectBuffer(maxSelect, selectBuff);
// trecere in mod select
Gl.glRenderMode(Gl.GL_SELECT);
// creare si initializare stiva de selectie
Gl.glInitNames();Gl.glPushName(0);
// preluare viewport curent
Gl.glGetIntegerv(Gl.GL_VIEWPORT, viewport);
int w = viewport[2], h = viewport[3];
double aspectRatio = (double)w / h;
Gl.glMatrixMode(Gl.GL_PROJECTION);
Gl.glLoadIdentity();
// stabilire volum de selectie
Glu.gluPickMatrix(xpos, h-ypos, 4, 4, viewport);
Glu.gluPerspective(unghiPerspective, aspectRatio, dMin, dMax);
// trasare in mod selectie
Gl.glMatrixMode(Gl.GL_MODELVIEW);
renderScene(Gl.GL_SELECT);
// determinarea numarului de primitive intersectate
hits = Gl.glRenderMode(Gl.GL_RENDER);
if (hits <= 0) return -1; else return hits;
}
```

Buffer-ul de selecție, numărul maxim de înregistrări şi *viewport*-ul utilizat în selecție au fost declarate ca şi câmpuri ale clasei care conține metoda, astfel:

```
private int maxSelect = 100;
private int[] buf_selectie = new int[400];
private int[] viewport = new int[4];
```

Comenzile OpenGL au fost invocate prin interfața Tao [8].

3.2 Pickingul în Java3D

Java 3D are implementate mecanisme care permit o uşoară manipulare a obiectelor 3D prin *picking*-selecţie şi detecţia coliziunilor [1,9,10]. Utilizarea acestor mecanisme este simplă şi nu implică efort de programare şi nu constituie obiectivul acestui articol. Vor fi amintite în continuare principalele clase prin care este implementat mecanismul de *picking*-selecţie:

1. *PickTool* este clasa de bază în operaţiunile de *picking*. Această clasă are metode care întorc câte un obiect *PickResult* pentru fiecare obiect grafic (primitivă sau Shape3D) selectat. Obiectul *PickResult* poate apoi să fie interogat pentru a obţine diverse detalii. Gradul de detaliere al informaţiei întoarse, precum şi aria de sensibilitate pentru picking se stabileşte cu ajutorul unui parametru pick-mode, furnizat prin constante ale clasei *PickTool*. Acest parametru poate avea valorile:

2. *PickCanvas* este o clasă derivată din *PickTool* care simplifică procesul de selecţie prin captarea evenimentelor de mouse utilizând obiectele *Canvas*. Sunt selectate primitivele grafice aflate în apropierea poziţiei de *click* pe obiectul *Canvas*.

3. Clasa *PickResult* este utilizată pentru a memora informaţii despre obiectele selectate.

4. Studiu de caz. Prezentarea unor obiecte 3D Studio Max într-o aplicaţie C#

Studiul de caz constă într-o aplicaţie scrisă în C# care prezintă scene construite în diverse produse de modelare 3D, cu posibilitatea selecţiei şi animării obiectelor din scenă. Aplicaţia se limitează la scene salvate în fişiere OBJ, ASE şi PLY. Grafica 3D este realizată în OpenGL utilizând biblioteca Tao [8]. *Tao Framework* este o colecţie de clase C# care facilitează accesul la biblioteca OpenGL sub platforma .NET [7]. Aplicaţia este structurată astfel:

- un set de clase de tip parser care analizează fişiere de tip OBJ, ASE şi PLY şi construiesc modelul geometric al scenei;
- o clasă care redă scena într-o fereastră *Form* şi care asigură interacţiunea cu elementele scenei.

Setul de clase de tip parser este grupat într-o bibliotecă structurată astfel:

```
namespace Parser3dsLibrary
{
    public abstract class Parser { ... }
    public class ASEParser : Parser { ... }
    public class OBJParser : Parser { ... }
    public class PLYParser : Parser { ... }
    public class GeomObject { ... }
  }
```

Un obiect *Parser* corespunde unei scene şi conține informații referitoare la scenă: fişerul în care a fost salvată scena, lista obiectelor din scenă şi elemente de tip material (componentele difuză, speculară şi ambientală). Clasele derivate au rolul de a analiza fişierul sursă şi de a extrage informațiile privind geometria corpurilor şi proprietățile de tip material. Fiecare clasă derivată din *Parser* tratează un anumit tip de fişier. Astfel, clasa *OBJParser* este destinată analizării fişierelor scenă în format OBJ. Obiectele din clasa *GeomObject* memorează caracteristicile geometrice (vârfuri, suprafețe, normale) şi de culoare ale unui obiect din scenă.

Clasa de tip *Form* care redă scena implementează următoarele elemente de interacțiune cu obiectele din scenă:

- manipularea obiectelor din scenă prin transformări geometrice (translații şi rotații) aplicate corpurilor la acționarea mouse-ului şi a tastaturii;
- selecția obiectelor din scenă cu ajutorul mouse-ului prin mecanismul de picking-selecție.

Un exemplu de implementare a unui comportament la acționarea mouse-ului este metoda următoare:

```
private void fogl_MouseMove(object sender, MouseEventArgs e)
{
if (e.Button == MouseButtons.Left){
       if (e.X < px) { uy--; fogl.Invalidate(); }
       else if (e.X > px) { uy++; fogl.Invalidate(); }
```

```
        if (e.Y < py) { ux--; fogl.Invalidate(); }
        else if (e.Y > py) { ux++; fogl.Invalidate(); }
        px = e.X; py = e.Y;
        }
}
```

Semnificația variabilelor utilizate este următoarea:

px, *py* - poziția precedentă a mouse-ului ;

ux, *uy* - unghiuri de rotație în raport cu axele Ox şi Oy;

fogl - fereastra OpenGL;

La deplasarea mouse-ului ținând butonul stânga apăsat, unghiurile de rotație în raport cu axele Ox şi Oy sunt incrementate în funcție de direcția de deplasare a mouse-ului. Acest lucru are ca efect rotația corespunzătoare a tuturor corpurilor sau a corpurilor selectate din scenă.

Redarea scenei se realizează prin metoda *renderScene* astfel:

```
private void renderScene(int mode)
{
transform();
for (int k = 0; k < scene.Count(); k++){
        if (mode == Gl.GL_SELECT) Gl.glLoadName(nrCorp[k]);
        if (select[k])drawObject(scene.GetObject(k));
        }
}
```

Metoda *transform* invocată mai sus este o metodă care realizează rotațiile şi translațiile parametrizate cu ajutorul mouse-ului şi a tastaturii asupra obiectelor din scenă. Variabilele utilizate în metodă au semnificațiile:

scene - este obiectul *Parser* corespunzător scenei;

select - este un vector de valori booleene care arată dacă un obiect din scenă a fost sau nu selectat;

nrCorp - este un vector de numere intregi asociate corpurilor din scenă. Aceste valori sunt depuse pe stiva de nume în procesul de *picking*-selecție.

Metoda care desenează un corp este *drawObject* şi este prezentată în continuare.

```
private void drawObject(GeomObject obj)
{
List<double[]> vertices = obj.GetVertices(),
       sNormals = obj.GetSNormals(), vNormals = obj.GetVNormals();
List<double[]> tex = obj.GetTexture();
List<int[]> faces = obj.GetFaces();
float[] color = obj.GetColor();
if (color == null)color = scene.GetDiffuseMaterial();
Gl.glColor3fv(color);
List<double[]> pVertices, pNormals;
if (sNormals != null && vNormals != null)
       if(normalVertex)
              for (int i = 0; i < faces.Count; i++){
                pVertices = new List<double[]>();
                pNormals = new List<double[]>();
                for (int j = 0; j < faces[i].Count(); j++){
                  pVertices.Add(vertices[faces[i][j]]);
                  pNormals.Add(vNormals[faces[i][j]]);
                }
                drawPolygon(pVertices, pNormals, color, tex);
              }
       else
              for (int i = 0; i < faces.Count; i++){
                pVertices = new List<double[]>();
                for (int j = 0; j < faces[i].Count(); j++)
                  pVertices.Add(vertices[faces[i][j]]);
                drawPolygon(pVertices, sNormals[i], color, tex);
              }
else
       for (int i = 0; i < faces.Count; i++){
              pVertices = new List<double[]>();
              for (int j = 0; j < faces[i].Count(); j++)
```

```
                pVertices.Add(vertices[faces[i][j]]);
            drawPolygon(pVertices, color, tex);
        }
}
```

Variabilele utilizate în metodă sunt furnizate de obiectul *GeomObject* rezultat în urma parsării şi reprezintă informații citite din fişierul scenă (OBJ, ASE sau PLY): coordonate de vârfuri, normale la suprafețe, texturi sau proprietăți (material, culoare). Desenarea efectivă a unui poligon (de cele mai multe ori un triunghi) se realizează prin metodele *drawPolygon* care au în interior apeluri pentru primitiva OpenGL *GL_POLYGON*.

Mecanismul de *picking*-selecție este declanşat prin dublu *click* pe mouse şi este tratat prin metoda următoare:

```
private void fogl_MouseDoubleClick(object sender, MouseEventArgs e)
{
xpos = e.X; ypos = e.Y; // Pozitia mouse-ului
int nrO = procesareSelectie();
if (e.Button == MouseButtons.Left){
        for (int k = 0; k < nrCorp.Length; k++) select[k] = false;
        for (int k = 0; k < nrO; k++)
                selection(selectBuff[k * 4 + 3], true);
        }
else {
        for (int k = 0; k < nrCorp.Length; k++) select[k] = true;
        for (int k = 0; k < nrO; k++)
                selection(selectBuff[k * 4 + 3], false);
        }

fogl.Invalidate();
}
```

Un dublu *click* stânga elimină ceea ce nu este in volumul de selecție, în timp ce un dublu *click* dreapta elimină ceea ce este in volumul de selecție.

Metoda care tratează *picking*-ul în OpenGL, *procesareSelectie*, a fost prezentată în capitolul 3. Prin metoda *selection* sunt stabilite corpurile selectate, astfel:

```
void selection(int k, bool selected){
for (int i = 0; i < nrCorp.Length; i++)
       if (nrCorp[i] == k) {select[i] = selected; return; }
}
```

Semnificația câmpurilor *select* şi *nrCorp* a fost prezentată mai sus.

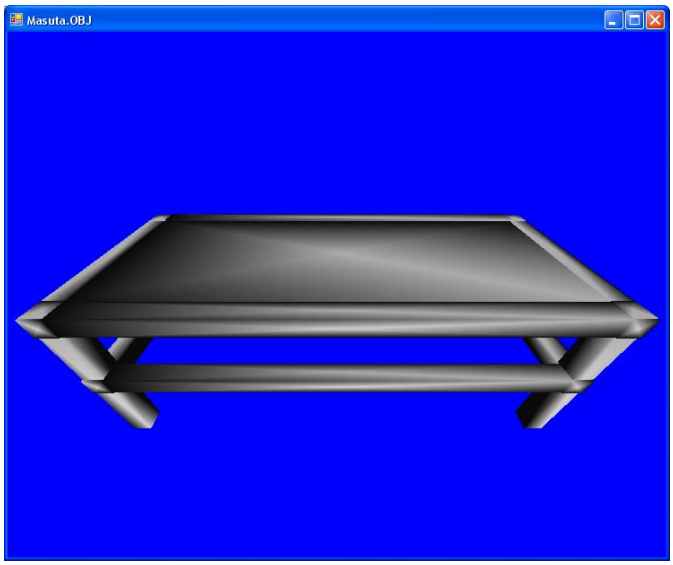

Figura 1. Obiectul masuta.obj

În figura 1 este vizualizat un obiect 3D construit în 3D Studio Max [5]. Este un obiect de mobilier al cărui model 3D a fost memorat ca şi câmp într-o tabela a unei baze de date, alături de alte informații. Obiectul poate fi descompus şi vizualizat în structură, la nivel de componente, pe care aplicația le poate selecta efectuând dublu-*click* stânga sau dreapta pe mouse. În figura 2 sunt prezentate câteva astfel de selecții. Acesta este un exemplu de utilizare a claselor prezentate, într-o aplicație de gestiune, pentru a

extinde funcționalitatea acesteia prin prezentarea de informații complete, inclusiv grafice.

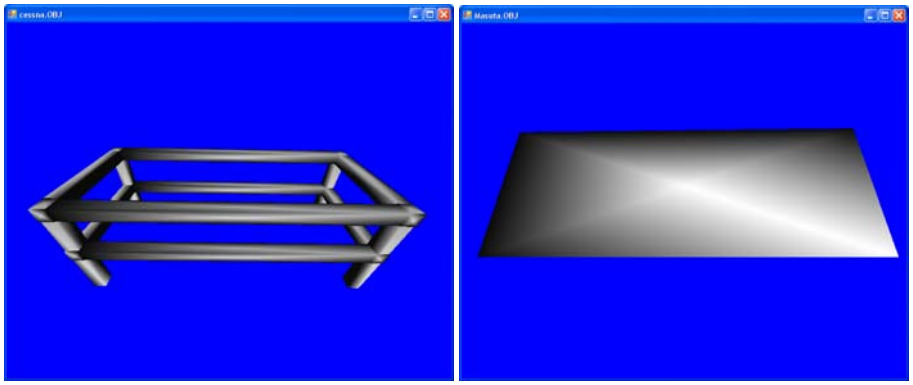

Figura 2. Componente ale obiectului

5. Concluzii

Integrarea scenelor 3D în aplicații eterogene complexe implică inevitabil programare grafică la nivel scăzut. În același timp efortul de programare pentru construirea unor modele 3D la acest nivel este foarte mare. Apare astfel ca justificat procedeul prin care, scene 3D construite prin instrumente specializate, sunt exportate şi salvate în fişiere cu formate consacrate, pentru ca apoi acestea să fie integrate în alte aplicații. Aplicațiile care integrează scene 3D au sarcina de a asigura partea de interactivitate cu elementele din scenă. OpenGL, Direct3D sau Java3D pot fi bune soluții de integrare a scenelor 3D, asigurând o bună interactivitate şi performanțe ridicate în redare. Soluția propusă constă într-o arhitectură simplificată dar extensibilă de clase prin care se realizează integrarea scenelor 3D într-o aplicație C#. Avantajele acestui demers față de utilizarea unor biblioteci standard sunt:

- flexibilitatea şi adaptabilitatea la orice aplicație scrisă în C#;
- utilizarea specificației OpenGL în aplicații C#, combinând atât avantajele OpenGl cât şi cele oferite de platforma .NET în realizarea unei aplicații portabile, independente de platformă [7];
- posibilitatea de a implementa elemente de interacțiune cu obiecte din scena.

Referințe

Falcidieno, B., Herman, I., Pienovi, C., *Computer Graphics and Mathematics*, New York, Springer, 1992

McReynolds, T., Blythe, D., *Advanced Graphics Programming Using OpenGL*, Morgan Kaufmann Publishers, 2005

Randi, J.,R., *OpenGL Shading Language*, Addison Wesley Professional, 2006

Baciu, R., *Programarea aplicațiilr grafice 3D cu OpenGL*, Cluj-Napoca, Editura Albastră, 2005

Dariush Derakhshani, Randi L. Munn, *Introducing 3ds Max 9*, Wiley Publishing, 2007

Furtuna, F., Grafică interactivă cu aplicații în Java şi Java 3D, Editura ASE

Dârdală, M**.,** Reveiu, A., Smeureanu, I., *Using DLL as Interface between API and VC#.NET Applications,* Informatica Economică, vol. X, nr. 1, Editura INFOREC, Bucureşti, 2006, pag 73-76

*** *Tao Framework*, http\\www.taoframework.com, http://en.wikipedia.org/wiki/Tao_Framework

***, *Euclidian Space – building a 3D world*, http://www.euclideanspace.com/

***, *Java 3D API*, http://java.sun.com/products/java-media/3D/