2. Componente software pentru dezvoltarea aplicatiilor grafice

Software-ul utilizat pentru implementarea aplicaţiilor de grafică se împarte în două categorii mari:

- *limbaje de nivel înalt*, care in general conţin unit-uri sau librării de functii pentru modul de lucru grafic;
- *programe specializate* (medii pentru dezvoltare grafică), care de regulă conţin biblioteci de obiecte grafice şi funcţii de prelucrare grafică avansată.

Dezvoltarea aplicaţiilor de grafică pe calculator sub sistemul de operare Windows utilizează conceptul de *interfaţă grafică de dispozitiv* - Graphical Device Interface (GDI). **Componentele GDI sunt următoarele**:

- contextul de dispozitiv,
- primitivele grafice,
- transformări de coordonate,
- funcţii pentru afişarea fonturilor şi a textelor,
- funcţii pentru afişarea pensulelor, peniţelor,
- funcţii pentru controlul culorilor (paleta de culori).

Orice aplicaţie de grafică pe calculator asigură (stabileşte) legătura cu un echipament de ieşire (monitor de afişare, plotter, imprimantă, etc) oarecare prin două mecanisme ce ţin de *contextul dispozitivului* (device context) şi de *controlorul de dispozitiv* (device driver):

aplicaţie → *context dispozitiv* → *controlor de dispozitiv*→ echipament de ieşire

Contextul de dispozitiv reprezintă un *parametru utilizat de toate funcţiile GDI* care precizează sub forma unei *structuri de date* **atributele curente de afişare** la un dispozitiv (echipament de ieşire). Datele conţinute reprezintă informaţii despre capabilităţile dispozitivului grafic al calculatorului (placa video, monitorul, etc.), atribute grafice privitoare la modul de trasare (desenare) a obiectelor, culoarea de desenare, culoarea fundalului, tipul de linie utilizată, fontul şi culoarea textelor, zona disponibilă pentru desenare, etc.. Toate aceste date sunt conţinute într-o partiţie de memorie a mediului de programare, iar accesul din aplicaţie se realizează prin intermediul unui *indicator al contextului de dispozitiv* - Handle Device Context (HDC) cunoscut în limbajul curent sub denumirea de *handler*.

Din punctul de vedere al **controlului dispozitivului** grafic sub sistemul de operare Windows posibilităţile de afişare sunt asigurate de un **set de rutine pentru gestionarea API** cunoscut sub denumirea de **DirectX.** Acesta permite controlul eficient al acceleratorului grafic independent de dispozitivul de afişare. Acest pachet de rutine este folosit curent de aplicaţiile multimedia.

Sub sistemul de operare Windows funcţiile de bază GDI sunt conţinute în modulul GDI.EXE. Marele avantaj al funcţiilor GDI este că asigură independenţa aplicaţiilor de dispozitivul grafic, permiţând ca acestea să poată rula pe platforme hardware diferite după ce au fost **compilate** pe acestea.

Pe lângă aplicaţiile uzuale ce folosesc funcţiile GDI, posibilităţile sistemului de operare Windows vizează o gamă largă de domenii pentru aplicaţii performante de grafică pe calculator cum sunt: jocurile, aplicaţiile multimedia şi grafica 3D. Pentru acestea Windows pune la dispoziţie suportul necesar pentru înglobarea unor utilitare ce operează cu biblioteci grafice complexe. Astfel de biblioteci sunt WinG care conţine funcţii de afişare rapidă pentru jocuri şi **biblioteca Open GL** (în mod tradiţional Opengl32.lib, Opengl32.dll), care aduce un suport puternic pentru grafica 3D şi pentru aplicaţiile de realitate virtuală. Biblioteca OpenGL poate fi exploatata cu utilitare specifice pentru a oferi o *interfaţă pentru programarea aplicaţiilor*- Application Programming Interface (API). Un astfel de utilitar este **GLUT - OpenGL Utilitary Toolkit.**

Directive grafice în programarea aplicaţiilor cu limbaje de nivel înalt

Comenzile date calculatorului pentru executarea elementelor de grafică se numesc **directive grafice**. Acestea sunt recunoscute de compilatorul sau interpretorul mediului de programare si se regăsesc în toate limbajele de programare înaltă cu denumiri şi sintaxe intuitive şi, au ca efect executarea aşa numitelor *primitive grafice*, care fac parte din funcţiile GDI (*Graphical Device Interface*).

Conceptul de primitiva grafică

Primitiva grafică se bazează pe noţiunea de *primitivă geometrică* care reprezintă o formă elementară considerată ireductibilă cu ajutorul căreia se pot reprezenta forme geometrice tot mai complexe. În toate limbajele de programare de nivel înalt se găsesc primitive grafice clasificate dupa criteriul dimensionalităţii astfel:

Punctul – dimensiunea 0, este asociat cu un pixel fiind elementul grafic ireductibil (atomul) fundamental;

Linia, curba – dimensiunea 1, include şi poliliniile dar şi curbe de tip Bezie, Spline.

Regiuni plane – dimensiunea 2, formate din contururi de linii închise, foarte popular fiind triunghiul - utilizat ca primitivă grafică pentru generarea oricărui poligon, dar şi forme limitate de curbe închise (cerc, elipsă) si combinate (curbe si linii) de tip sector circular;

Regiuni volumetrice – dimensiunea 3.

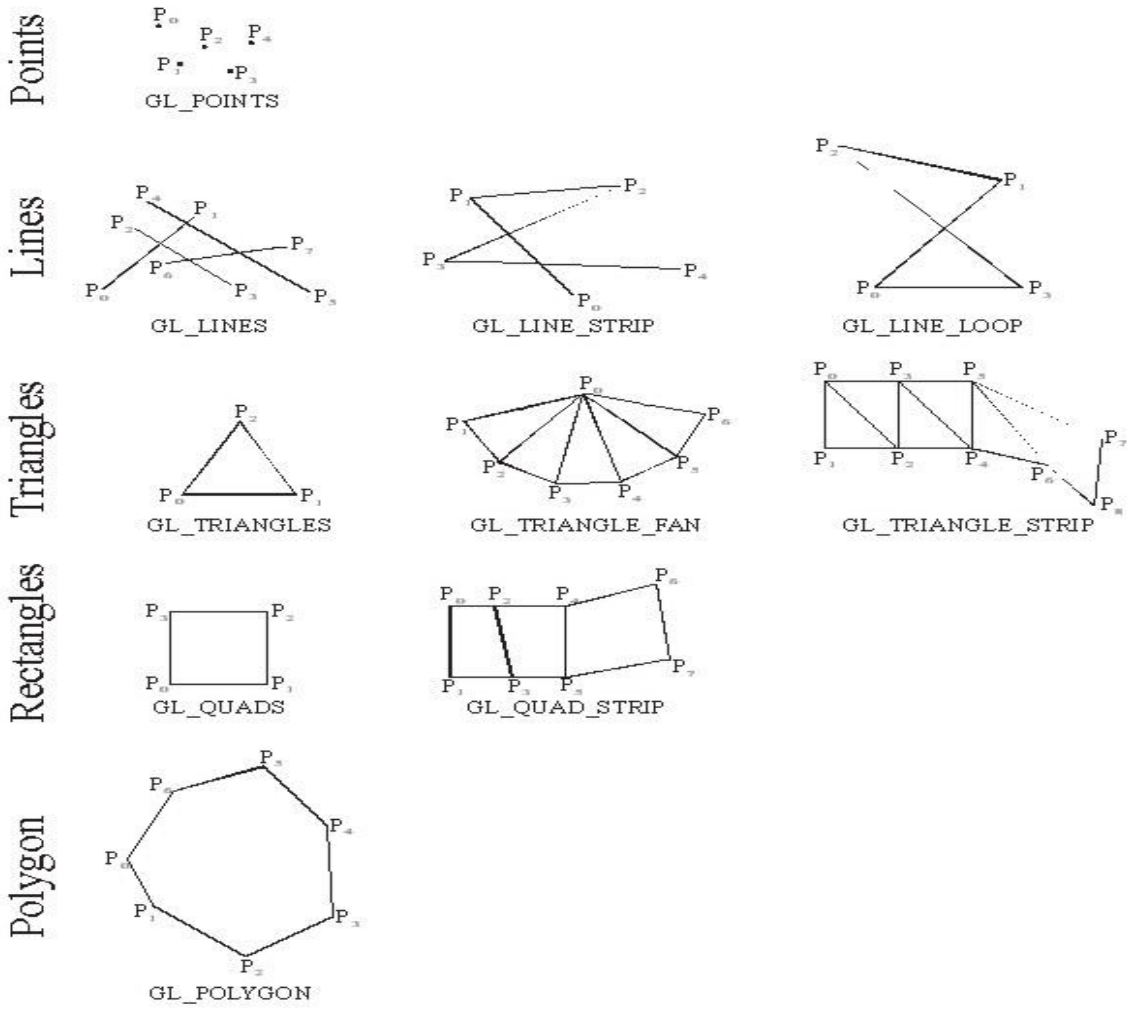
Grafică utilizând funcţiile GDI (*Graphical Device Interface*). Primitivele grafice GDI sunt:

Primitiva	Funcţia GDI şi argumentele
Putpixel	`PutPixel(x,y, color)`
Linie	`MoveTo(hDC, x, y), LineTo(hDC, x, y),`
Linie frântă	`Polyline(hDC, VectXY, NrPuncte)`
Poligon	`Polygon(hDC, VectXY, NrPuncte)`
Poligoane Multiple	`PolyPolygon(hDC,VectXY,NrPunctePolig, NrPoligoane)`
Elipsă, cerc	`Ellipse(hDC,DrXstg_sus,DrYstg_sus,DrXdr_jos,DrYdr_jos)`
Arc de elipsă	`Arc(hDC,DrXss,DrYss,DrXdj,DrYdj,xStart,yStart,xEnd,yEnd)`
Coardă	`Chord(hDC,DrXss,DrYss,DrXdj,DrYdj,xPct1,yPct1,xPct2,yPct2)`
Sector de elipsă	`Pie(hDC,DrXss,DrYss,DrXdj,DrYdj,xPct1,yPct1,xPct2,yPct2)`
Drptunghi	`Rectangle(hDC,Xstg_sus,Ystg_sus,Xdr_jos,Ydr_jos)`
Dreptunghi cu colţuri rotunjite	`RoundRect(hDC,Xstg_sus,Ystg_sus,Xdr_jos,Ydr_jos, XdiamElipsa, YDiamElipsa)`

Exemple de primitive grafice în OpenGL

Primitiva/Descriere	**Funcţia şi argumentul**
Desenează n puncte independente, singulare	`GL_POINTS`
Desenează segmente de dreaptă izolate (distincte) pe baza perechilor de puncte definite (v0,v1), (v2,v3),… Dacă n este impar, ultimul vârf este ignorat	`GL_LINES`
Desenează linia poligonală formată din segmente conectate	`GL_LINE_STRIP`
Desenează linia poligonală *închisă* formată din segmentele (ultimul nod se conecteaza cu primul)	`GL_LINE_LOOP`
Desenează o serie de triunghiuri independente folosind vârfurile (v0,v1,v2), (v3,v4,v5), …samd. Dacă n nu este multiplu de 3, atunci ultimele 1 sau 2 vârfuri sunt ignorate	`GL_TRIANGLES`
Desenează o serie de triunghiuri conectate. Fiecare al treilea punct defineste un triunghi impreuna cu ultimele doua puncte definite. (Ordinea *este aleasă* astfel ca triunghiurile să aibă orientare si conectare in banda)	`GL_TRIANGLE_STRIP`
Deseneaza un grup de triunghiuri conectate in evantai: fiecare al treilea punct defineste un triunghi impreuna cu primul (Po) si ultimul.	`GL_TRIANGLE_FAN`
Deseneaza un grup de patrulatere independente	`GL_QUADS`
Deseneaza un grup de patrulatere conectate: un nou patrulater este desenat dupa fiecare pereche de puncte definita dupa precedenta pereche.	`GL_QUADS_STRIP`
Deseneaza un singur poligon convex	`GL_POLYGON`

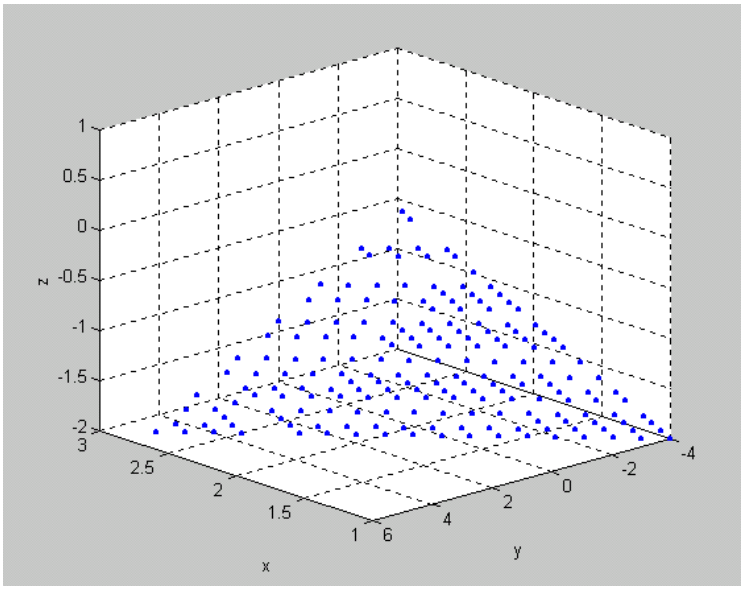
Terminologie: vertex – nod / vertices – noduri.



Aplicaţie

Algoritm şi program demonstrativ pentru determinarea unei suprafeţe plane (generarea planului prin puncte discrete). Exemplu implementat în Matlab.

```
%plan
clear
%definirea planului in forma canonica
a=-5
b=1
c=-3
d=3
%determinarea punctelor care apartin planului
for x=1:0.01:5
   for y=-5:0.01:5
      for z=-2:0.01:2
         p=a*x+b*y+c*z+d
%conditia de apartenenta la plan si reprezentare grafica a
         if p==0
            plot3(x,y,z,'.')
            hold on
         end
      end
   end
end
grid
xlabel('x')
ylabel('y')
zlabel('z')
```



Tema:

1) Determinaţi tăieturile (pe axele de coordonate) planului modelat în aplicaţia de mai sus.
2) Rescrieţi programul de calcul astfel încât să permită definirea planului prin tăieturi.
3) Rescrieti aplicatia folosind directive grafice in C++