

**UNIVERSITATEA TITU MAIORESCU
FACULTATEA DE INFORMATICA**

GRAFICA PE CALCULATOR

Conf. dr.ing. Mironela PIRNAU

*Acest material este destinat studentilor **anului II**, specializarea **Informatică**.*

*Disciplina **Grafica pe calculator** utilizează noțiunile predate la disciplinele Bazele informaticii, Programare procedurala si alte discipline studiate în anul I si anul II sem I.*

O neînțelegere a notiunilor fundamentale prezentate în acest curs poate genera dificultăți în asimilarea conceptelor complexe introduse în alte cursuri de specialitate.

Nota finală acordată fiecărui student, va conține următoarele componente în procente menționate:

- proba scrisă	50%
- realizare proiect	30%
- teste pe parcursul semestrului	20%

Fiecare student va realiza un proiect care conține două parti: un referat de maxim 6 pagini, despre unul din subiectele tratate în curs și minim 8 programe asemănătoare cu cele tratate în suportul de curs. Toate funcțiile utilizate vor fi explicate ca rol și parametrii.

Proiectul se va susține în ultimele două laboratoare.

Bibliografie	<ol style="list-style-type: none">1. Suportul propriu de curs pentru studenții de la învățământ ID (accesibil de pe platforma e-learning/CD)2. Baci, R., Programarea aplicațiilor grafice 3D cu OpenGL, Editura Albastră, Cluj-Napoca, 2005.3. Moldoveanu, F., Racoviță, Z., Hera, G., Petrescu, Ș., Zaharia, M., Grafica pe calculator, Editura Teora, București, 1996.4. Ionescu, F., Grafica în realitatea virtuală, Editura Tehnică, București 2000.5. Hearn, Donald, Backer, M. Pauline, Computer Graphics, Prentice-Hall, Inc, Englewood Cliffs, New Jersey, 19866. Foley, J., A. van Dam, E. S. van Dam, S. K. van Dam, J. F. van Dam, Computer Graphics: principles and practice, Addison Wesley Publishing Company, second edition, 1993.7. Neider, J., Davis, T., Woo, M., OpenGL Programming Guide, Addison-Wesley, Menlo Park, 1993.8. Dave Shreiner, Graham Sellers, John Kessenich, Bill Licea-Kane, The Khronos OpenGL ARB Working Group, OpenGL Programming Guide Eighth Edition, ISBN-13: 978-0-321-77303-6, ISBN-10: 0-321-77303-9, 20139. https://docs.unity3d.com/Manual/index.html10. https://developer.nvidia.com/opengl11. https://www.opengl.org/
--------------	--

Contents

1. OBIECTIVE GENERALE	5
1.1 Informatii generale despre grafica pe calculator	5
1.2 Modelele cromatice RGB și CMYK	7
1.3 Fractali si utilizarea vectorizarii	8
2. <i>Utilizarea obiectelor imagine</i>	11
2.1 <i>Softurilor dedicate pentru realizarea fisierelor grafice</i>	11
2.2 <i>Utilizarea Interfetelor API</i>	13
2.3 <i>DirectX rol si functionare</i>	14
3. <i>Prezentarea si utilizarea bibliotecii OpenGL</i>	17
3.1 Biblioteca OpenGL	17
3.2 Utilizarea functiilor din GLUT.h	20
3.3 Aplicatii rezolvate	26
4. <i>Realizarea desenelor 2D / 3D folosind OpenGL</i>	41
4.1 <i>Functii pentru lucru cu ferestre</i>	41
4.2 Gestiunea meniurilor	44
4.3 Afișarea obiectelor 3D predefinite	46
4.4 Recapitulare functii principale in OpenGL	53
4.5 Probleme rezolvate	62
4.6 Probleme propuse	75
5. Unity Game Engine	78
5.1 Unity Game Engine	78
5.2 Functii din clasa MonoBehaviour	81
5.3 Parcurgerea unui tutorial	84
Model bilet 1	85
Model bilet 2	86
Model bilet 3 /Partea I	87
Model bilet 4	88

1. **OBIECTIVE GENERALE**

- *Intelegerea conceptelor generale corespunzatoare graficii raster si graficii vectoriale;*
- *Insusirea modului in care se utilizeaza teoria culorilor in grafica;*
- *Rolul fractarilor si a vectorizarii.*

Keywords: GUI, RGB, CMYK, GIS, FRACTALI

Cuprins

- 1.1 Informatii generale despre grafica pe calculator
- 1.2 Modelele cromatice RGB și CMYK
- 1.3 Fractali si utilizarea vectorizarii

1.1 Informatii generale despre grafica pe calculator

Grafica pe calculator reprezinta acele metode si tehnici de conversie a datelor catre si de la un dispozitiv grafic prin intermediul calculatorului.

Aplicarea graficii pe calculator este folosita din următoarele domenii principale:

- **Vizualizarea informației**
- **Proiectare**
- **Modelare** (simulare)
- **Interfață grafică pentru utilizatori *GUI***

În prezent cunoașterea elementelor de bază ale graficii pe calculator este necesară

- inginerului,
- omului de știință,
- artiștilor plastici,
- designerilor,
- fotografilor,
- pictorilor de animație etc.

Datorită calculatorului putem avea la dispoziție în câteva fracțiuni de secundă variații multiple de culoare, forme, compoziții etc.

În baza tehnologiilor graficii computerizate s-au dezvoltat:

- Interfața de utilizator – GUI (---inteligenta)
- Proiectarea digitală în arhitectură și grafica industrială
- Efecte vizuale specializate, cinematografia digitală
- Grafica pe computer pentru filme, animație, televiziune
- Rețeaua Internet --- SM
- Conferințele video
- Televiziunea digitală
- Proiecte multimedia, proiecte interactive
- Fotografia digitală și posibilitățile avansate de prelucrare a fotografiei
- **Grafica și pictura digitală** (cu 2 laturi esențiale – imitarea materialelor tradiționale și noile instrumente de lucru digitale)
- Vizualizarea datelor științifice și de afaceri
- Jocuri pe calculator, sisteme de realitate virtuală (de ex. simulatoare pentru aviație)

- Sisteme de proiectare automatizată
- Tomografie computerizată
- Poligrafia
- **Grafica laser**

Grafica pe calculator este de asemenea și un domeniu de activitate științifică

Grafica digitală

OBSERVAȚIE: **Grafica rastru și vector stau la baza graficii digitale.**

Grafica digitală este un domeniu al informaticii care acoperă toate aspectele legate de formarea imaginilor utilizând un computer.

Termenul englez corespunzător este **computer graphics**.

Grafica digitală mai este numită uneori grafică de computer, grafică de calculator, grafică pe calculator, grafică computerizată.

Grafica digitală este o activitate în care computerul este utilizat pentru sintetizarea și elaborarea imaginilor, dar și pentru prelucrarea informației vizuale obținute din realitatea ce ne înconjoară.

Grafica digitală se divizează în mai multe categorii:

- Grafica bidimensională (se constituie din **grafică raster** și grafică vector)
- Grafica Fractal
- Grafica **tridimensională** (este des utilizată în animație, spoturi publicitare, jocuri la computer etc.)
- CGI grafică (**CGI** – în engleză imagini, personaje generate de computer)
- Grafica animată
- Grafica video
- Grafica web
- Grafica Multimedia

La baza graficii digitale (și în special grafica bidimensională) stau două tipuri de calculații, **Raster** (sau rastru) și Vector (vectorială).

Grafica rastru și vector stau la baza graficii digitale, ele sunt utilizate de programele 3D dimensionale, programe pentru montare video, animație, etc.

Prezentarea la calculator a imaginii de tip **Raster** se face în baza segmentării unei suprafețe cu ajutorul unor pătrățele mici care se numesc pixeli. O imagine rastru este un tablou format din mai mulți pixeli. Cu cât mai mulți pixeli avem în imagine cu atât calitatea detaliilor e mai înaltă.

Acest tip de grafică permite să cream și să reproducem oricare imagine cu multitudinea de efecte și subtilități, indiferent de complexitate. Imaginile procesate cu ajutorul scannerului sau aparatelor foto sunt formulate ca **raster**.

- *Neajunsurile* – e complicată redimensionarea graficii rastru, deoarece la interpolare (adăugare de pixeli în raport de cei existenți în imagine) computerul inventează calculând pixelii nu întotdeauna satisfăcător, cu toate că sunt diverse metode de interpolare.

Grafica vectorială este un procedeu prin care imaginile sunt construite cu ajutorul descrierilor matematice prin care se determină **poziția, lungimea și direcția liniilor folosite în desen**. Grafica vectorială e bazată ca principiu pe desen cu ajutorul liniilor calculate pe o suprafață. **Liniile pot fi drepte sau curbe.**

În cazul imaginilor vectoriale fișierul stochează liniile, formele și culorile care alcătuiesc imaginea, ca formule matematice. Imaginile Vector pot fi mărite și micșorate fără a pierde calitatea.

O imagine poate fi modificată prin manipularea obiectelor din care este alcătuită, acestea fiind salvate apoi ca variații ale formulelor matematice specifice. Este des utilizată în imagini cu funcții decorative, caractere-text, logotipuri, infograme, decor, cat și în proiecte sofisticate 3D, montare video, animație, etc.

Există sisteme sofisticate de control cromatic al imaginilor. Adaptarea valorilor cromatice se aplică în funcție de necesități și scopuri. Astfel, pentru controlul de culoare la computere, există mai multe modele cromatice (sau modele de culoare), care pot fi folosite pentru definirea cromatică a imaginii, pentru selectarea culorilor în procesul desenării și creării imaginii, pentru arhivare, etc.

1.2 Modelele cromatice RGB și CMYK

În general există două tipuri majore de modele cromatice:

- amestecul aditiv RGB (din engleză de la Red-Green-Blue, roșu-verde-albastru) – amestec de culori lumină – se utilizează pentru ecran, monitor, televiziune, scanare, aparate foto, design web, animație și altele.
- amestec substractiv CMYK (de la Cyan-Magenta-Yellow-black) – amestec de culori pigment – se utilizează pentru proiecte poligrafice destinate tiparului color.

Unele modele de culoare admit variații ale diapazonului cromatic, din acest motiv sunt împărțite în diverse profiluri de culoare destinate diferitelor necesități. De exemplu, pentru modelul RGB există mai multe profiluri. sRGB IEC61966-2.1 – diapazon mediu utilizat mai des pentru pagini web, sau Adobe RGB (1998) – diapazon mai mare utilizat pentru prelucrarea imaginilor destinate produselor pentru tipar.

Pe lângă modelele cromatice RGB și CMYK mai există și altele precum: Lab, HSB / HSL / HSI, Culori Indexate, Culori Pantone/PMS (Pantone Matching System), Culori Spot.

Desktop publishing, abreviat *DTP*, este o expresie engleză din domeniul tipografiei și care s-ar putea traduce prin "publicare cu ajutorul calculatorului de tip *desktop*". Este procesul de creare și editare (modificare) a unui document cu ajutorul unui computer având ca obiectiv final tipărirea lui.

În acest domeniu există programe de calculator specializate ce pot fi împărțite în mai multe categorii:

- programe de creație și prelucrare foto (Adobe Photoshop, Corel Photopaint);
- programe de creație și prelucrare de grafică vectorială (Adobe Illustrator, Corel Draw);
- programe de paginare (Adobe InDesign, QuarkXPress, Adobe Pagemaker, Corel Ventura);
- programe utilitare (Adobe Acrobat Professional, Adobe Acrobat Distiller etc);

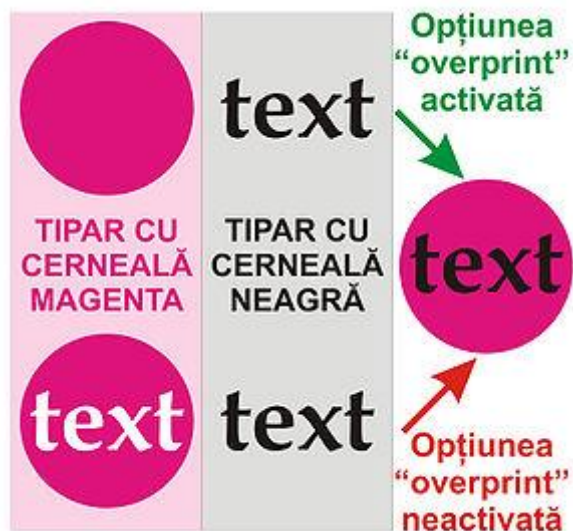
Cel mai utilizat program pentru realizarea documentelor ce urmează a fi tipărite este Corel Draw. Are setări speciale pentru separații, overprint

Monitoarele sunt aparatele ce decodifică semnalul electric și generează culorile RGB (Red, Green, Blue – roșu, verde, albastru) prin suprapunere de lumină. Când nu există nici un fel de lumină este generată culoarea neagră, când intensitatea prin cele trei canale este maximă, se obține culoarea albă.

Teoretic s-ar putea tipări și folosind direct spațiul de culori RGB, însă datorită imperfecțiunii hârtiei, cernelii și a mașinilor de tipar, se folosește spațiul CMYK (Cyan, Magenta, Yellow, black), urmând ca negrul, care se tipărește la sfârșit de procedură, să acopere aceste imperfecțiuni.

La o lucrare pretențioasă este indicat ca imaginile, textele și obiectele colorate să fie combinate atent, mai ales când se lucrează cu culori Pantone, pentru ca la sfârșit să nu rezulte amestecaturi nedorite de culoare.

Funcția de overprint este suprapunerea la tipar a două culori diferite, culoarea de deasupra (ultimă) acoperind total culoarea de dedesubt (anterioară). Dacă activăm opțiunea Overprint la culoarea neagră, atunci cercul va putea rămâne întreg și nu va mai fi nevoie să fie mai înainte decupat în locurile unde urmează a fi tipărit textul negru.



1.3 Fractali si utilizarea vectorizarii

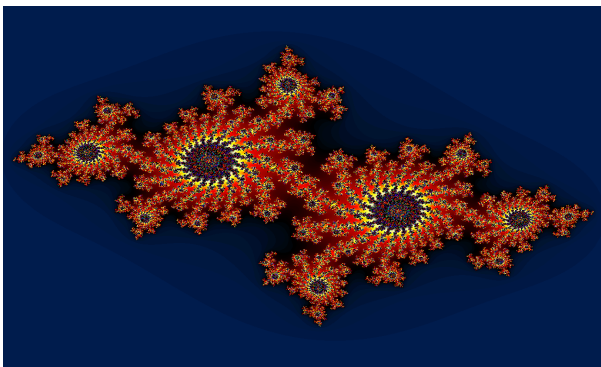
Fractal Explorer este un generator de fractali care poate produce imagini frumoase, misterioase matematic generate. Fractal Explorer poate face seturi de fractali clasici, polinomial, (cum ar fi Mandelbrot, Julia, Newton și variante ale lor). - Fractali complecși 4D (numiți «quaternions»), «atractori stranii» 3D și IFS. De asemenea, FE are multe posibilități pentru crearea de efecte speciale și de îmbunătățire a imaginilor.

Termenul *fractal* provine din latinescul *fractus*, care înseamnă "spart", "fracturat". Acest termen a fost introdus de Benoît Mandelbrot, în 1975.

Un fractal este un obiect matematic care are o structură detaliată la orice scară. În structura unui fractal, fiecare parte este asemănătoare cu fractalul întreg (este autosimilar).

Fractalii, aceste deosebite obiecte matematice, de o mare complexitate, sunt generați printr-un procedeu matematic relativ simplu (metoda iterației).

Dimensiunea geometrică a unui fractal se bazează pe dimensiunea **Hausdorff**, care este o extensie a dimensiunii euclidiene. Dacă în geometria euclidiană un obiect nu are decât o dimensiune întreagă, în geometria fractală dimensiunile sunt, în general, numere reale neîntregi pozitive



În grafica digitală se operează cu diverse elemente grafice, pentru elaborarea și controlul imaginilor ca de exemplu: **pixel, punct, linie, curbă, poligon etc.**

Afișarea și crearea imaginilor vectoriale

Display-urile computerelor sunt alcătuite din puncte minuscule numite pixeli. Imaginile bitmap sunt de asemenea construite folosind aceste puncte. Cu cât sunt mai mici și mai apropiate, cu atât calitatea imaginii este mai ridicată, dar și mărimea fișierului necesar pentru stocarea ei. Dacă imaginea este afișată la o mărime mai mare decât cea la care a fost creată inițial, devine granulată și neclară, deoarece pixelii din alcătuirea imaginii nu mai corespund cu pixelii de pe ecran.

Pentru a crea și modifica imagini vectoriale sunt folosite programe software de desen vectorial. O imagine poate fi modificată prin manipularea obiectelor din care este alcătuită, acestea fiind salvate apoi ca variații ale formulelor matematice specifice. Operatori matematici din software pot fi folosiți pentru a întinde, răsuci, colora diferitele obiecte dintr-o imagine. În sistemele moderne, acești operatori sunt prezentați în mod intuitiv folosind interfața grafică a calculatorului.

Conversia din și în format raster

Vectorizarea imaginilor este utilă pentru eliminarea detaliilor nefolositoare dintr-o fotografie. Conversia din format vectorial se face practic de fiecare dată când este afișată imaginea, astfel încât procesul de îl salva ca bitmap într-un fișier este destul de simplu.

Mult mai dificil este procesul invers, care implică aproximarea formelor și culorilor din imaginea bitmap și crearea obiectelor cu proprietățile corespunzătoare. Numărul obiectelor generate este direct proporțional cu complexitatea imaginii. Cu toate acestea, mărimea fișierului cu imaginea în format vectorial nu va depăși de obicei pe cea a sursei bitmap.

Aplicațiile grafice avansate pot combina imagini din surse vectoriale și raster și pun la dispoziție unelte pentru amândouă, în cazurile în care unele părți ale proiectului pot fi obținute de la o cameră, iar altele desenate prin grafică vectorială.

Vectorizarea

Aceasta se referă la programe și tehnologii/servicii folosite pentru a converti imagini de tip bitmap în imagini de tip vectorial. Exemple de utilizare:

- În Proiectarea asistată pe calculator (CAD) schițele sunt scanate, vectorizate și transformate în fișiere CAD printr-un process denumit sugestiv hârtie-CAD.
- În GIS imaginile provenite de la sateliți sunt vectorizate cu scopul de a obține hărți.
- În arta digitală și fotografie, imaginile sunt de obicei vectorizate folosind plugin-uri pentru programe ca Adobe Photoshop sau Adobe Illustrator, dar vectorizarea se poate face și manual. Imaginile pot fi vectorizate pentru o mai bună utilizare și redimensionare, de obicei fără diferențe mari față de original. Vectorizarea unei fotografii îi va schimba aspectul din fotografic în pictat sau desenat; fotografiile pot fi transformate și în siluete. Un avantaj al vectorizării este că rezultatul poate fi integrat cu succes într-un design precum un logo.

Dezavantaje și limitări

Principalul dezavantaj al imaginilor vectoriale este că, fiind alcătuite din obiecte descrise cu formule matematice, atât numărul acestor obiecte cât și complexitatea lor sunt limitate, depinzând de biblioteca de formule matematice folosită de programul de desenare. De exemplu, dispozitivele digitale, cum ar fi camerele foto sau scannerele, produc fișiere raster care nu pot fi reprezentate fidel folosind imagini vectoriale. Chiar și în cazul în care se reușește vectorizarea unei astfel de imagini, editarea acestora la complexitatea originală este dificilă. Un alt dezavantaj este că formatele în care sunt stocate imaginile vectoriale sunt foarte complexe. Implementarea acestor formate pe dispozitive diferite este problematică din această cauză. Conversia dintr-un format în altul este de asemenea dificilă.

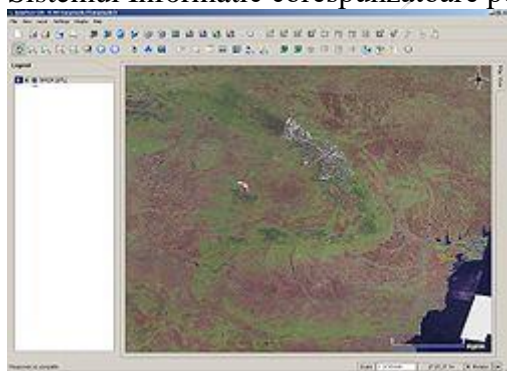
Datorită flexibilității în ceea ce privește rezoluția imaginilor vectoriale, acestea sunt folosite intensiv pentru crearea materialelor ce trebuie imprimate la mărimi foarte diverse: același fișier poate fi folosit pentru un card de vizită cât și pentru un panou publicitar, în ambele cazuri rezultatele fiind foarte clare și precise.

O altă aplicație semnificativă a graficii vectoriale este în modelarea suprafețelor 3D, unde se dorește o calitate ridicată a obiectelor.

GIS este acronimul provenit de la **Geographic Information System (Sistem Informatic Geografic** - uneori tradus în forma **SIG** în limba română). Acest sistem este utilizat pentru a crea, stoca, a analiza și prelucra informații distribuite spațial printr-un proces computerizat. Tehnologia GIS poate fi utilizată în diverse domenii științifice cum ar fi: managementul resurselor, studii de impact asupra mediului, cartografie, planificarea rutelor.

Specific unui GIS este modul de organizare a informației gestionate. Există două tipuri de informație: una grafică care indică repartitia spațială a elementelor studiate și alta sub formă de bază de date pentru a stoca atributele asociate acestor elemente (de ex. pentru o șosea lungimea ei, lățimea, numărul benzilor, materialul de construcție etc.).

Informația grafică poate fi de două feluri: **raster** sau **vectorială**. Grafica raster este o modalitate de reprezentare a imaginilor în aplicații software sub forma de matrice de pixeli în timp ce grafica vectorială este o metoda de reprezentare a imaginilor cu ajutorul unor primitive geometrice (puncte, segmente, poligoane), caracterizate de ecuații matematice. Specific sistemelor GIS este asocierea unui sistem de coordonate geografice matricii de pixeli (la imaginile raster) sau vectorilor - procedeul poartă numele de **Georeferențiere**. Astfel unui obiect (reprezentat fie printr-o imagine, fie printr-un vector) îi este asociată o poziție unică în Sistemul Informatic corespunzătoare poziției geografice din lumea reală.



Informație tip raster - imagine satelitară - dintr-un sistem GIS

Datorită informațiilor asociate graficii, Sistemele Informatic Geografice beneficiază de toate oportunitățile de interogare pe care le oferă sistemele moderne de baze de date și în plus pot oferi ușor analize orientate pe anumite zone geografice - așa numitele hărți tematice.

Un exemplu comun de Sistem Informatic Geografic îl reprezintă Sistemele de Navigație. Harta rutieră în formă vectorială este georeferențiată astfel încât Sistemul de Poziționare Globală (Global Positioning System - GPS) să poată indica poziția exactă a autovehiculului. Planificarea rutei este în fapt o hartă tematică obținută în urma unei interogări spațiale (căutarea distanței celei mai scurte între două puncte) combinată cu o interogare a bazei de date asociate drumurilor

din hartă astfel încât să fie respectate o serie de condiții (limitări de viteză, gabarit, sensuri de circulație, interdicții, etc.).

Datorită impactului pozitiv, sistemele software GIS s-au dezvoltat foarte mult. Există pe piață un număr foarte mare de produse, atât ale dezvoltatorilor consacrați (ESRI, Intergraph, Autodesk, MapInfo, etc.) dar și de tip Open source (Grass GIS, Quantum GIS, GVSIG, OpenJump, etc.).

2. *Utilizarea obiectelor imagine*

OBIECTIVE GENERALE:

- *Cunoasterea softurilor dedicate pentru realizarea fisierelor grafice.*
- *Intelegerea modului de functionare a Interfetelor API.*
- *Intelegerea modului de functionare pentru DirectX.*

KEYWORDS: *Adobe Photoshop, Interfete API, MicroSoft DirectX*

Cuprins

2.1 Softurilor dedicate pentru realizarea fisierelor grafice

2.2 Utilizarea Interfetelor API

2.3 DirectX rol si functionare

2.1 *Softurilor dedicate pentru realizarea fisierelor grafice*

Adobe Photoshop este un software folosit pentru editarea imaginilor digitale pe calculator, program produs și distribuit de compania americană Adobe Systems și care se adresează în special profesioniștilor domeniului.

Adobe Photoshop, așa cum este cunoscut astăzi, este vârful de lance al gamei de produse software pentru editare de imagini digitale, fotografii, grafică pentru tipar, video și Web de pe piață. Photoshop este un program cu o interfață intuitivă și care permite o multitudine extraordinară de modificări necesare în mod curent profesioniștilor și nu numai: editări de luminozitate și contrast, culoare, focalizare, aplicare de efecte pe imagine sau pe zone (selecții), retușare de imagini degradate, număr arbitrar de canale de culoare, suport de canale de culoare pe 8, 16 sau 32 biți, efecte third-party etc. Există situații specifice pentru un profesionist în domeniu când alte pachete duc la rezultate mai rapide, însă pentru prelucrări generale de imagine, întrucât furnizează instrumente solide, la standard industrial, Photoshop este efectiv indispensabil.

Alături de aplicația Photoshop (ajuns la versiunea CS5), este inclusă și aplicația ImageReady, cu un impresionant set de instrumente Web pentru optimizarea și previzualizarea imaginilor (dinamice sau statice), prelucrarea pachetelor de imagini cu ajutorul sistemului droplets-uri (mini-programe de tip drag and drop) și realizarea imaginilor rollover (imagini ce își schimbă aspectul la trecerea cu mouse-ul peste), precum și pentru realizarea de GIF-uri animate.

Principalele elemente prin care Photoshop se diferențiază de aplicațiile concurente și prin care stabilește noi standarde în industria prelucrării de imagini digitale sunt:

- Selecțiile

- Straturile (Layers)
- Măștile (Masks)
- Canalele (Channels)
- Retușarea
- Optimizarea imaginilor pentru Web

Photoshop poate citi majoritatea fișierelor raster și vector. De asemenea, are o serie de formate proprii:

- **PSD** (abreviere pentru Photoshop Document). Acest format conține o imagine ca un set de straturi (Layers), incluzând text, măști (mask), informații despre opacitate, moduri de combinare (blend mode), canale de culoare, canale alfa (alpha), căi de tăiere (clipping path), setări duotone precum și alte elemente specifice Photoshop. Acesta este un format popular și des răspândit în rândul profesioniștilor, astfel că este compatibil și cu unele aplicații concurente Photoshop.
- **PSB** (denumit Large Document Format) este o versiune mai nouă a formatului PSD, conceput special pentru fișiere mai mari (2GB) sau cu o informație prezentă pe o suprafață definită de laturi mai mari de 30.000 de pixeli (suportă până la 300.000x300.000 pixeli).
- **PDD** este un format mai puțin întâlnit, fiind asociat inițial aplicației Adobe PhotoDeluxe, astăzi (după 2002) compatibil doar cu aplicațiile Adobe Photoshop sau Adobe Photoshop Elements.
- **Camera RAW**: Instrumentul oferă acces rapid și facil la imaginile tip RAW produse de majoritatea camerelor foto digitale profesionale și de mijloc. Camera RAW se folosește de toate detaliile acestor fișiere pentru a obține un control total asupra aspectului imaginii, fără a modifica fișierul în sine.
- **Adobe Bridge**: Un browser complex, de ultimă generație, ce simplifică gestionarea fișierelor, poate procesa mai multe fișiere de tip RAW în același timp și pune la dispoziția utilizatorului informația metadata de tip EXIF etc.
- **Multitasking**: Adobe introduce posibilitatea de a folosi toate aplicațiile sale din suita "Creative suite 2" în sistem multitasking.
- **Suport High Dynamic Range (HDR) pe 32 biți**: Creează și editează imagini pe 32 biți, sau combină cadre fotografice de expuneri diferite într-una ce include valorile ideale de la cele mai intense umbre până la cele mai puternice zone de lumină.
- **Shadow/Highlight**: Îmbunătățește contrastul fotografiilor subexpuse sau supraexpuse, inclusiv imagini CMYK, păstrând în continuare echilibrul vizual al imaginii.
- **Vanishing Point**: Oferă posibilitatea de a clona, picta sau lipi elemente ce automat se transpun în perspectiva obiectelor din imagine.
- **Image Warp**: Capacitatea de a deforma imaginile plane după o matrice ușor editabilă, folosind mouse-ul.
- **Corectarea deformărilor cauzate de lentile: Lens Distort** corectează cu ușurință efectele obișnuite date de lentilele aparatelor foto precum cele cilindrice, sferice, tip pâlnie, "efectul de vignetă" (funcție de poziționarea față de lumină, colțurile fotografiilor sunt fie întunecate, fie luminate în contrast cu restul fotografiei) sau aberațiile cromatice.
- **Personalizarea aplicației**: Posibilitatea de a personaliza orice scurtătură sau chiar funcțiile din meniul aplicației și posibilitatea de a salva modificările pentru fiecare mod de lucru în parte.
- **Control îmbunătățit al straturilor (layers)**: capacitatea de a selecta mai multe straturi în același timp.
- **Smart objects**: abilitatea de a deforma, redeforma și a reveni la starea inițială a obiectelor fără a pierde din calitate.

Aplicațiile grafice se realizează prin utilizarea următoarelor clase de produse software: editoare grafice, biblioteci grafice, programe grafice specializate, desktop publishing, worksheet graphics (birotică).

Editoarele grafice sunt destinate prelucrării grafice în domeniul proiectării asistate de calculator dintre produsele care înglobează editoare grafice amintim: Auto Cad, Freelanc 2 Plus, Windows Paint, Corel Draw.

Bibliotecile grafice sunt destinate prelucrărilor prin intermediul limbajelor de programare de nivel înalt (C, Visual C, Visual Basic, Delphi, Open GL etc...). Acestea conțin biblioteci cu rutine (primitive) grafice care realizează funcții grafice necesare aplicații grafice.

Programe grafice specializate sunt destinate rezolvării problemelor din anumite domenii, oferind utilizatorului să enunțe probleme cât mai simplu și în concordanță cu limbajul utilizat în domeniul său. De exemplu: Matlab, Mathematica, Mathcad.

Desktop publishing sunt produse software destinate realizării de publicații (ziare, reviste, reclame, etc.) la birou, parcurgând toate etapele dintr-o tipografie clasică. De exemplu: Xerox Ventura Publisher care este compatibil cu mai multe procesoare de text și cu unele editoare grafice, Pagemaker, Xpress, Super Paint, Publish IT, etc.

2.2 Utilizarea Interfetelor API

O interfata API este un cod sursă oferit de către sistemul de operare sau o bibliotecă pentru a permite apeluri la serviciile care pot fi generate din API-uri respective de către un program. Termenul API este folosit în 2 sensuri:

- O interfata coerentă care constă din clase sau seturi de funcții sau proceduri interconectate.

- Un singur punct de intrare, cum ar fi o metodă, o funcție sau o procedură.

Două Interfete API foarte cunoscute sunt Single UNIX Specification și Microsoft Windows API.

Interfete API sunt deseori încorporate în Software Development Kit (SDK)

Modelul de design a Interfetelor API

Există o multitudine de modele de design a Interfetelor API. Cele prevăzute pentru execuție rapidă deseori constă din funcții, proceduri, variabile și structuri de date. Există și alte modele cum ar fi interpretatori (emulatori) care evaluează expresii în ECMAScript (cunoscut sub numele JavaScript) sau alt layer abstract, oferind programatorului posibilitatea de a evita implicarea în relațiile funcțiilor cu nivelul inferior al abstracției.

Unele Interfete API, cum sunt cele standard pentru un sistem de operare, sunt implementate ca biblioteci de cod separate care sunt distribuite împreună cu sistemul de operare. Altele au integrat funcționalitatea interfeței API direct în aplicație. Microsoft Windows API este distribuită cu sistemul de operare. Interfetele API pentru sisteme embedded, cum sunt console de jocuri video, în general intră în categoria API-urilor integrate direct în aplicație.

O interfata API care nu necesită drepturi mari de acces sunt numite "open" (OpenGL ar fi un exemplu).

Câteva exemple de Interfete API:

- Single UNIX Specification (SUS)
- Microsoft Win32 API
- Java Platform, Enterprise Edition API's
- OpenGL cross-platform API
- DirectX for Microsoft Windows
- Google Maps API
- Wikipedia API
- Simple DirectMedia Layer (SDL)
- svglib pentru Linux și FreeBSD

- Carbon si Cocoa pentru Macintosh OS

Microsoft Windows API's

Windows API, neoficial WinAPI, este numele dat de catre Microsoft pentru un set de Interfete API disponibile in sisteme de operare Microsoft Windows. Aceste interfete au fost construite pentru a fi folosite de catre programatori C/C++ si sunt cel mai direct mod de a interactiona cu sistemul Windows pentru aplicatii software. Accesul la nivel inferior la sistemul Windows, in general necesar pentru drivere, este oferit de catre Windows Driver Foundation in versiunea curenta a Windows-ului.

In sisteme de operare Windows este disponibil un Software Development Kit (SDK), care ofera documentatia si unelte pentru a permite dezvoltatorilor crearea aplicatiilor folosind Interfete API si tehnologii Windows asociate.

Compilatoare suportate

Pentru a dezvolta software care foloseste Interfetele API Windows, este nevoie de compilator care poate importa si manipula fisierele DLL si obiectele COM caracteristice Microsoft-ului. Compilatorul trebuie sa accepte dialectul limbajelor C sau C++ si sa manipuleze IDL (interface definition language) fisiere si fisiere header care expun denumirile functiilor interioare ale Interfetelor API. Aceste compilatoare, unelte, librarii si fisiere header sunt impreunate in Microsoft Platform SDK (Software Development Kit). Pentru mult timp familia de compilatoare si unelte Microsoft Visual Studio si compilatoare Borland, au fost singurele care puteau la cerintele sus mentionate. Acuma exista *MinGW* si *Cygwin* care ofera interfete bazate pe *GNU Compiler Collection*. *LCC-Win32* este disponibil pentru utilizare non-comerciala, continand compilator C si intretinut de catre Jacob Navia. *Pelles C* este compilator C gratuit oferit de catre Pelle Orinius.

2.3 DirectX rol si functionare

Microsoft DirectX

Istoric

Microsoft DirectX este o colecție **Application Programing Interface (APIs)** care se ocupa cu sarcinile legate de multimedia, in special programarea jocurilor si video pe platformele Microsoft. La început numele acestor API-uri începeau toate cu Direct, cum ar fi: *Direct3D*, *DirectDraw*, *Direct Music*, *DirectPlay*, *DirectSound*, etc. Numele DirectX a fost folosit pentru a scurta numele acestor API-uri, unde litera X înlocuia numele diferitului API. Mai târziu când Microsoft a început sa dezvolte propria consola de jocuri a avut grija ca litera X sa apară in nume pentru a se face legătura cu DirectX.

Direct3D (componenta 3D din API-ul DirectX) este folosita in mod intens in jocurile video pentru Microsoft Windows, Sega Dreamcast, Microsoft Xbox (360 sau One). Direct3D este folosit de asemenea in aplicații software pentru vizualizare si taskuri grafice precum CAD/CAM.

La sfârșitul anului 1994 era gata sa lanseze următorul sau sisteme de operare, Windows 95. Cei de la Microsoft aveau teama că sistemul de operare precedent al său MS-DOS sa fie văzută ca o platforma mai bună pentru programatorii de jocuri. DOS permitea acces direct la placi video, tastaturi, mouse, device-uri audio, etc. pe când Windows 95 – cu modelul sau de memorie protejată – restricționa accesul către aceste componente folosind un model mult mai standardizat. Pentru a rezolva acesta “problemă” Microsoft a lansat DirectX.

Prima versiune DirectX a fost lansata in Septembrie 1995 si se numea SDK-ul Windows Games. Adoptarea inițiala a lui DirectX a fost foarte lenta. Programatorii se temeau ca DirectX sa fie înlocuit precum predecesorul său WinG. De asemenea era penalitatea de performanță față de DOS, care avea acces direct la componente. DirectX 2.0 a devenit o componenta a Windows-ului o data cu lansarea Windows 95 OSR2 la jumătatea lui 1996.

Echipa din spatele DirectX avea sarcina dificilă de a testa fiecare DirectX lansat cu o multitudine de hardware și software. O varietate de plăci grafice, plăci audio, plăci de bază, CPU-uri, device-uri de input, jocuri și alte aplicații multimedia erau testate cu fiecare lansare beta sau lansare finală. De asemenea echipa DirectX a construit și distribuit teste care permitea industriei hardware să își testeze hardware-ul și driverele pentru compatibilitate. Înainte să apară DirectX, Microsoft includea OpenGL. La acel moment pentru OpenGL trebuia hardware foarte performant și era focusat pe ingineri și utilizatori de CAD. Direct3D a fost intenționat precum un partener minor al lui OpenGL focusat pe jocuri. Pe măsura ce jocurile 3D au devenit mai populare, OpenGL s-a dezvoltat pentru a include tehnici de programare mai bune pentru multimedia interactivă precum jocuri, dând dezvoltatorilor posibilitatea de a alege între Direct3D sau OpenGL precum un API 3D. O "luptă" între dezvoltatorii care susțineau standardul deschis OpenGL și cei care susțineau Direct3D al lui Windows.

În versiunea specifică consolelor DirectX era API-ul utilizat în Microsoft Xbox, Xbox 360, și Xbox One. API-ul a fost dezvoltat precum o colaborare între Microsoft și nVidia – partenerul care a dezvoltat componentele grafice hardware. În 2002, Microsoft a lansat DirectX 9, care suporta un program mult mai lung de shading decât înainte cu un pixel și vertex shader versiune 2.0. Microsoft a continuat să actualizeze DirectX introducând shader Model 3.0 în DirectX 9.0c, care a fost lansat în August 2004.

DirectX este compus din mai multe API-uri:

- Direct3D (D3D) – pentru grafica 3D
- DXGI – pentru a enumera adaptoarele și monitoare pentru Direct3D 10 și mai sus
- Direct2D – pentru grafica 2D
- DirectWrite – pentru fonturi
- DirectCompute - pentru calculele GPU (Graphical Procesor Unit)
- DirectSound3D (DS3D) – pentru sunetele 3D
- DirectX Media – compus din DirectAnimation pentru animații Web 2D/3D, DirectShow pentru a rula multimedia și pentru media streaming, DirectX Transform pentru interacțiuni Web și Direct3D Retained Mode pentru grafica 3D de nivel înalt.
- DirectX Diagnostics (DxDiag) – o unealtă pentru a diagnostica și genera rapoarte pentru componentele relevante pentru DirectX cum ar fi audio, video și driverele de input
- DirectX Media objects – suporta obiectele de streaming cum ar fi: encodare, decodare și efecte.
- DirectSetup – pentru instalarea componentelor DirectX și detecția versiunii curente
- XACT3 – API pentru audio de nivel înalt
- XAudio2 - API pentru audio de nivel jos

Microsoft a renunțat la dezvoltarea acestor componente DirectX, dar încă le suporta:

- DirectDraw – pentru grafica 2D. S-a renunțat la această componentă în favoarea lui Direct2D
- DirectInput – pentru interfața cu device-urile de input precum tastaturi, mouse, joystick-uri precum și alte controller de jocuri. A fost înlocuit după versiune 8 cu XInput pentru Xbox 360 sau standardul WM_INPUT pentru tastatură și mouse.
- DirectPlay – pentru comunicarea pe rețea. A fost înlocuit după versiune 8 în favoarea Games for Windows Live și Xbox Live
- DirectSound – pentru redarea și înregistrarea sunetelor. Înlocuit cu XAudio2 și XACT3
- DirectMusic – pentru redarea colonei sonore autorizate prin DirectMusic Producer. Înlocuit cu XAudio2 și XACT3

DirectX 10

Un update important al lui DirectX a fost DirectX 10. Acesta a fost disponibil doar în Windows Vista sau o versiune mai nouă, versiunile mai vechi de Windows fiind limitate la versiunea 9.0c. Modificările aduse lui DirectX 10 au fost majore. Multe componente au fost depreciate și înlocuite

cu unele mai noi fiind păstrate doar pentru compatibilitate: DirectInput a fost înlocuit cu XInput, DirectSound cu Cross-Platform Audio Creation Tool (XACT) și adițional a pierdut suport pentru accelerare audio hardware, deoarece stack-ul audio în Windows Vista reda sunetul în software în CPU.

Pentru a realiza compatibilitatea cu jocurile mai vechi DirectX în Windows Vista conține mai multe versiuni de Direct3D

- Direct 3D 9 – emulează Direct3D 9 așa cum ar rula în Windows XP
- Direct 3D 9Ex – permite acces total la noile capacități WDDM în timp ce menține compatibilitate cu aplicațiile Direct3D existente
- Direct 3D 10 – Proiectat în jurul noului model de driver din Windows Vista, precum și noi îmbunătățiri la capacitățile de randare și flexibilitate, incluzând Shader Model 4.

Direct 3D 10.1 a fost un update incremental al lui Direct 3D 10.0 și avea nevoie de Windows Vista Service Pack 1.

DirectX 11

Microsoft a dezvoltat DirectX 11 în 2008 cu noi caracteristici:

- Suport GPGPU (DirectCompute)
- Direct 3D 11 cu suport pentru țesătură și suport pentru procesare multithread ceea ce ajută dezvoltatorii de jocuri să utilizeze procesoarele multicore mai bine

Direct 3D 11 rulează pe orice versiune de Windows începând cu Windows 7. Partea din noul API, cum ar fi multi-threaded poate fi suportat și de hardware compatibil Direct3D 9/10/10.1 dar țesătură hardware și Shader Model 5.0 are nevoie de hardware specific DirectX 11. **DirectX 11.1** este inclus în Windows 8 și suportă WDDM 1.2 pentru performanță sporită, integrare îmbunătățită a lui Direct2D (versiune 1.1), versiuni îmbunătățite ale lui Direct3D și DirectCompute și include DirectXMath XAudio2 și XInput. De asemenea suportă stereoscopic 3D pentru jocuri și video. **DirectX 11.2** este inclus în Windows 8.1 și a adăugat noi caracteristici ale lui Direct2D precum realizări geometrice. **DirectX 11.X** este un superset al lui DirectX 11.2 care rulează pe Xbox One și include câteva caracteristici noi precum DrawBundles, care au fost mai târziu anunțate pentru DirectX 12. **DirectX 11.3** a fost anunțat simultan cu DirectX 12 și este pentru o îmbunătățirea lui DirectX 11.2 fiind sisteme de operare care nu suportă DirectX 12 (de la Windows 7 până la Windows 8.1)

DirectX 12

DirectX 12 a fost anunțat în martie 2014 și a fost lansat odată cu Windows 10 în iulie 29 2015. DirectX 12 rulează de asemenea pe Xbox One (sub forma de 11.X) precum și Windows Phone.

Principala caracteristică a lui DirectX 12 a fost introducerea unui nou API scris într-un limbaj de programare low-level ceea ce a dus la reducerea încărcării suplimentare aduse de drivere. Acum dezvoltatorii își pot implementa propria listă de comenzi și buffere direct în GPU, permițând utilizarea eficientă a resurselor prin calcul paralel. Suport pentru utilizarea mai multor plăci grafice pe același sistem simultan. Înainte de DirectX12 multi GPU suport era suportat doar prin implementările producătorilor de hardware prin implementări precum AMD CrossFire sau NVIDIA SLI.

- Multi-adaptor implicit - este suportat similar cu versiunile precedente de DirectX unde fiecare cadru este alternativ randat pe fiecare placă video (similară) alternativ
- Multi-adaptor explicit - vor oferi două API-uri distincte dezvoltatorilor
 - GPU-uri legate – va permite DirectX să perceapă plăcile grafice în SLI sau CrossFire ca o singură placă grafică
 - GPU-uri independente – va permite DirectX-ului să suplimenteze GPU-ul dedicat cu cel integrat în CPU sau să combine plăci Video AMD cu NVIDIA.

Observație:

Există alternative pentru această arhitectură, unele mai complete decât altele. Nu există o soluție unică care să ofere toate funcționalitățile care le ofera DirectX, dar prin combinare de

librării - OpenGL, OpenAL, SDL, OpenML, FMOD, etc - utilizatorul poate să implementeze o soluție comparabilă cu DirectX și în același timp gratuită și cross-platform (poate fi folosită pe diferite platforme).

3. *Prezentarea și utilizarea bibliotecii OpenGL*

Obiective generale:

- Cunoașterea importanței și rolului standardului OPENGL;
- Cunoașterea modului de instalare și utilizare a Bibliotecii OpenGL;
- Cunoașterea și înțelegerea modului de realizare a programelor grafice folosind funcțiile din GLUT.h
- Realizarea aplicațiilor grafice folosind principalele tipuri de primitive geometrice pentru a fi desenate folosind OpenGL.

KEYWORDS: OPENGL; GLUT.H; Primitive Geometrice;

Cuprins:

3.1 Biblioteca OpenGL
3.2 Utilizarea funcțiilor din GLUT.h
3.3 Aplicații rezolvate

3.1 Biblioteca OpenGL

Dintre bibliotecile grafice existente, **biblioteca OpenGL, scrisă în limbajul C**, este una dintre cele mai utilizate, datorită faptului că implementează un număr mare de funcții grafice de bază pentru crearea aplicațiilor grafice interactive, asigurând o interfață independentă de platforma hardware.

Deși introdusă în anul **1992**, **biblioteca OpenGL a devenit în momentul de față cea mai intens folosită interfață grafică pentru o mare varietate de platforme hardware și pentru o mare varietate de aplicații grafice 2D și 3D, de la proiectare asistată de calculator (CAD), la animație, imagistică medicală și realitate virtuală.**

Datorită suportului oferit prin specificațiile unui consorțiu specializat (OpenGL Architecture Review Board) și a numărului mare de implementări existente, în momentul de față OpenGL este în mod real singurul standard grafic multiplatformă. www.opengl.org.

Aplicațiile OpenGL pot rula pe platforme foarte variate, începând cu PC, stații de lucru și până la supercalculatoare, sub cele mai cunoscute sisteme de operare: Linux, Unix, Windows NT,

Windows 95, Mac OS. Funcțiile OpenGL sunt apelabile din limbajele Ada, C, C++ și Java. Din consorțiul de arhitectură OpenGL fac parte reprezentanți de la firmele Compaq, Evans-Sutherland, Hewlett-Packard, IBM, Intel, Intergraph, Microsoft și Silicon Graphics. Pentru scrierea aplicațiilor folosind interfața OpenGL de către utilizatori finali nu este necesară obținerea unei licențe.

Folosind biblioteca OpenGL, un programator în domeniul graficii se poate concentra asupra aplicației dorite și nu mai trebuie să fie preocupat de detaliile de implementare a diferitelor funcții “standard”. Programatorul nu este obligat să scrie algoritmul de generare a liniilor sau a suprafețelor, nu trebuie să calculeze decuparea obiectelor la volumul de vizualizare, etc, toate acestea fiind deja implementate în funcțiile bibliotecii și, cel mai probabil, mult mai eficient și adaptat platformei hardware decât le-ar putea realiza el însuși.

În redarea obiectelor tridimensionale, biblioteca OpenGL folosește un număr redus de primitive geometrice (puncte, linii, poligoane), iar modele complexe ale obiectelor și scenelor tridimensionale se pot dezvolta particularizat pentru fiecare aplicație, pe baza acestor primitive. Dat fiind că OpenGL prevede un set puternic, dar de nivel scăzut, de comenzi de redare a obiectelor, mai sunt folosite și alte biblioteci de nivel mai înalt care utilizează aceste comenzi și preiau o parte din sarcinile de programare grafică.

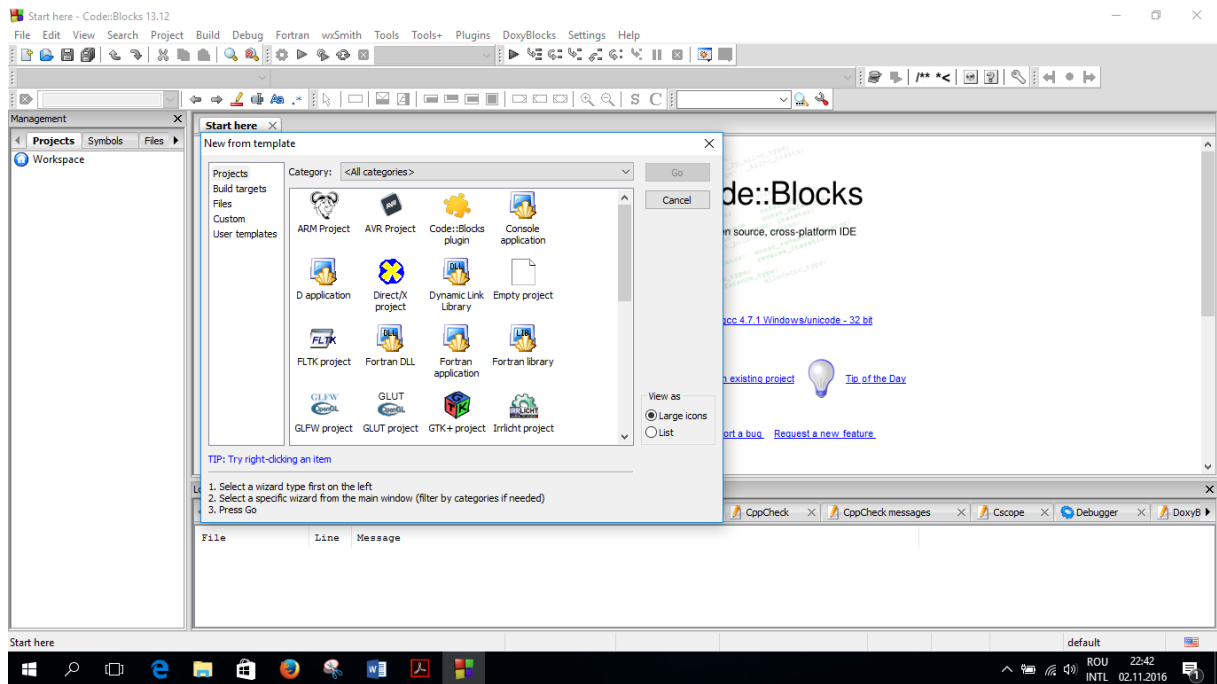
Astfel de biblioteci sunt:

- *Biblioteca de funcții utilitare GLU (OpenGL Utility Library) permite definirea sistemelor de vizualizare, redarea suprafețelor curbe și alte funcții grafice.*
- *Pentru fiecare sistem Windows există o extensie a bibliotecii OpenGL care asigură interfața cu sistemul respectiv: pentru Microsoft Windows, extensia WGL, pentru sisteme care folosesc sistemul X Window, extensia GLX, pentru sisteme IBM OS/2, extensia PGL.*
- *Biblioteca de dezvoltare GLUT (OpenGL Utility Toolkit) este un sistem de dezvoltare independent de sistemul Windows, care ascunde dificultățile interfețelor de aplicații Windows, punând la dispoziție funcții pentru crearea și inițializarea ferestrelor, funcții pentru prelucrarea evenimentelor de intrare și pentru execuția programelor grafice bazate pe biblioteca OpenGL.*

Instalare: se pun glut32.dll și glut.dll în c:\windows\system32 și c:\windows\; folder-ul GLUTMingw32 se pune în c:\, iar când se lansează un nou proiect OpenGL se va alege calea cu librării GLUTMingw32 de pe c:

..		
include		
lib		
glut.dll	221.184	79.368
glut32.dll	221.184	79.072
readme.txt	497	340

Dacă folosim CodeBlocks atunci realizăm un proiect de tip GLUT project și selectăm apoi locația unde am pus librăria GLUTMingw32 (ca în figurele de mai jos).



În orice aplicație OpenGL trebuie să fie incluse fișierele header `gl.h` și `glu.h` sau `glut.h` (dacă este utilizată biblioteca GLUT). Toate funcțiile bibliotecii OpenGL încep cu prefixul `gl`, funcțiile GLU încep cu prefixul `glu`, iar funcțiile GLUT încep cu prefixul `glut`.

Dezvoltarea programelor grafice folosind utilitarul GLUT

Biblioteca grafică OpenGL conține funcții de redare a primitivelor geometrice independente de sistemul de operare, de orice sistem Windows sau de platforma hardware. Ea nu conține nici o funcție pentru a crea sau administra ferestre de afișare pe display (windows) și nici funcții de citire a evenimentelor de intrare (mouse sau tastatură).

Pentru crearea și administrarea ferestrelor de afișare și a evenimentelor de intrare se pot aborda mai multe soluții: utilizarea directă a interfeței Windows (Win32 API), folosirea compilatoarelor sub Windows, care conțin funcții de acces la ferestre și evenimente sau folosirea altor instrumente (utilitare) care înglobează interfața OpenGL în sistemul de operare respectiv.

3.2 Utilizarea funcțiilor din GLUT.h

Un astfel de utilitar este GLUT, care se compilează și instalează pentru fiecare tip de sistem Windows. Header-ul glut.h trebuie să fie inclus în fișierele aplicației, iar biblioteca glut.lib trebuie legată (linkată) cu programul de aplicație.

Sub GLUT, orice aplicație se structurează folosind mai multe funcții callback. O funcție callback este o funcție care aparține programului aplicației și este apelată de un alt program, în acest caz sistemul de operare, la apariția anumitor evenimente. În GLUT sunt predefinite câteva tipuri de funcții callback; acestea sunt scrise în aplicație și pointerii lor sunt transmiși la înregistrare sistemului Windows, care le apelează (prin pointerul primit) în momentele necesare ale execuției.

Funcții de control ale ferestrei de afișare

void glutInit(int* argc, char argv);**

Această funcție inițializează GLUT folosind argumentele din linia de comandă; ea trebuie să fie apelată înaintea oricăror alte funcții GLUT sau OpenGL.

void glutInitDisplayMode(unsigned int mode);

Specifică caracteristicile de afișare a culorilor și a bufferului de adâncime și numărul de buffere de imagine. Parametrul mode se obține prin SAU logic între valorile fiecărei opțiuni.

Exemplu

glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|GLUT_DEPTH)

inițializează afișarea culorilor în modul RGB, cu două buffere de imagine și buffer de adâncime. Alte valori ale parametrului mode sunt: GLUT_SINGLE (un singur buffer de imagine), GLUT_RGBA (modelul RGBA al culorii), GLUT_STENCIL (validare buffer șablon) GLUT_ACCUM (validare buffer de acumulare).

void glutInitWindowPosition(int x, int y);

Specifică locația inițială pe ecran a colțului stânga sus al ferestrei de afișare.

void glutInitWindowSize(int width, int height);

Definește dimensiunea inițială a ferestrei de afișare în număr de pixeli pe lățime (width) și înălțime (height).

int glutCreateWindow(char* string);

Creează fereastra în care se afișează contextul de redare OpenGL și returnează identificatorul ferestrei. Această fereastră este afișată numai după apelul funcției **glutMainLoop()**.

Funcții callback.

Funcțiile callback se definesc în program și se înregistrează în sistem prin intermediul unor funcții GLUT. Ele sunt apelate de sistemul de operare atunci când este necesar, în funcție de evenimentele apărute.

glutDisplayFunc(void(*Display)(void));

Această funcție înregistrează funcția callback Display în care se calculează și se afișează imaginea. Argumentul funcției este un pointer la o funcție fără argumente care nu returnează nici o valoare.

Atentie:

Funcția Display (a aplicației) este apelată oricâte ori este necesară desenarea ferestrei: la inițializare, la modificarea dimensiunilor ferestrei.

glutReshapeFunc(void(*Reshape)(int w, int h));

Înregistrează funcția callback Reshape care este apelată ori de câte ori se modifică dimensiunea ferestrei de afișare. Argumentul este un pointer la funcția cu numele Reshape cu două argumente de tip întreg și care nu returnează nici o valoare. În această funcție, programul de aplicație trebuie să refacă transformarea fereastră-poartă, dat fiind că fereastra de afișare și-a modificat dimensiunile.

**glutKeyboardFunc(void(*Keyboard)(unsigned int key,
int x, int y);**

Înregistrează funcția callback Keyboard care este apelată atunci când se acționează o tastă. Parametrul key este codul tastei, iar x și y sunt coordonatele (relativ la fereastra de afișare) a mouse-ului în momentul acționării tastei.

glutMouseFunc(void(*MouseFunc)(unsigned int button, int state, int x, int y);

Înregistrează funcția callback MouseFunc care este apelată atunci când este apăsat sau eliberat un buton al mouse-ului.

Parametrul button este codul butonului și poate avea una din constantele GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON sau GLUT_RIGHT_BUTTON. Parametrul state indică apăsarea (GLUT_DOWN) sau eliberarea (GLUT_UP) al unui buton al mouse-ului. Parametrii x și y sunt coordonatele relativ la fereastra de afișare a mouse-ului în momentul evenimentului.

Exemplu:

```
void mouse(int buton, int stare, int x, int y)
{
    switch(buton)
    {
        case GLUT_LEFT_BUTTON:
            if(stare == GLUT_DOWN)
                glutIdleFunc(animatieDisplay);
            break;
        case GLUT_RIGHT_BUTTON:
            if(stare == GLUT_DOWN)
                glutIdleFunc(NULL);
            break;
    }
}
```

Funcții de execuție GLUT

Execuția unui program folosind toolkit-ul GLUT se lansează prin apelul funcției glutMainLoop(), după ce au fost efectuate toate inițializările și înregistrările funcțiilor callback. Această buclă de execuție poate fi oprită prin închiderea ferestrei aplicației. În cursul execuției sunt apelate funcțiile callback în momentele apariției diferitelor evenimente.

Generarea obiectelor tridimensionale

Multe programe folosesc modele simple de obiecte tridimensionale pentru a ilustra diferite aspecte ale prelucrărilor grafice. GLUT conține câteva funcții care permit redarea unor astfel de obiecte tridimensionale în modul wireframe sau cu suprafețe pline (*filled*). Fiecare obiect este reprezentat într-un sistem de referință local, iar dimensiunea lui poate fi transmisă ca argument al funcției respective. Poziționarea și orientarea obiectelor în scenă se face în programul de aplicație. Exemple de astfel de funcții sunt:

```
void glutWireCube(GLdouble size);
void glutSolidCube(GLdouble size);
void glutWireSphere(GLdouble radius, GLint slices,
    GLint stacks);
```

```
void glutSolidSphere(GLdouble radius, GLint slices,  
                    GLint stacks);
```

Programele GLUT au un mod specific de organizare, care provine din felul în care sunt definite și apelate funcțiile callback.

Poarta de afișare OpenGL

Poarta de afișare mai este numită context de redare (*rendering context*), și este asociată unei ferestre din sistemul Windows. Dacă se programează folosind biblioteca GLUT, corelarea dintre fereastra Windows și portul OpenGL este asigurată de funcții ale acestei biblioteci. Dacă nu se folosește GLUT, atunci funcțiile bibliotecilor de extensie XGL, WGL sau PGL permit atribuirea unui context de afișare Windows pentru contextul de redare OpenGL și accesul la contextul de afișare Windows (*device context*). Funcția OpenGL care definește transformarea fereastră-poartă este:

```
void glViewport(GLint x, GLint y,  
               GLsizei width, GLsizei height);
```

unde x și y specifică poziția colțului stânga-jos al dreptunghiului porții în fereastra de afișare (window) și au valorile implicite 0, 0. Parametrii width și height specifică lățimea, respectiv înălțimea, porții de afișare, dată ca număr de pixeli.

Exemplu: `glViewport(0, 0, (GLsizei) w, (GLsizei) h);`

ATENȚIE: transformarea fereastră-poartă este componentă a transformării din sistemul de referință normalizat în sistemul de referință ecran 3D

Un pixel este reprezentat în OpenGL printr-un descriptor care definește mai mulți parametri:

numărul de biți/pixel pentru memorarea culorii

numărul de biți/pixel pentru memorarea adâncimii

numărul de buffere de imagine

Bufferul de imagine (*color buffer*) în OpenGL poate conține una sau mai multe secțiuni, în fiecare fiind memorată culoarea pixelilor din poarta de afișare. Redarea imaginilor folosind un singur buffer de imagine este folosită pentru imagini statice, cel mai frecvent în proiectarea grafică (CAD). În generarea interactivă a imaginilor dinamice, un singur buffer de imagine produce efecte nedorite, care diminuează mult calitatea imaginii generate.

Atenție:

Orice cadru de imagine începe cu ștergerea (de fapt, umplerea cu o culoare de fond) a bufferului de imagine. După aceasta sunt generați pe rând pixelii care aparțin tuturor elementelor imaginii (linii, puncte, suprafețe) și intensitățile de culoare ale acestora sunt înscrise în bufferul de imagine. Atenție: Pentru trecerea la cadrul următor, trebuie din nou șters bufferul de imagine și reluată generarea elementelor componente, pentru noua imagine. Chiar dacă ar fi posibilă generarea și înscrierea în buffer a elementelor imaginii cu o viteză foarte mare (ceea ce este greu de realizat), tot ar exista un interval de timp în care bufferul este șters și acest lucru este perceput ca o pâlpâire a imaginii. În grafica interactivă timpul necesar pentru înscrierea datelor în buffer este (în cazul cel mai fericit) foarte apropiat de intervalul de schimbare a unui cadru a imaginii (update rate) și, dacă acest proces are loc simultan cu extragerea datelor din buffer și afișarea lor pe display, atunci ecranul va prezenta un timp foarte scurt imaginea completă a fiecărui cadru, iar cea mai mare parte din timp ecranul va fi șters sau va conține imagini parțiale ale cadrului. Tehnica universal folosită pentru redarea imaginilor dinamice (care se schimbă de la un cadru la altul) este tehnica *dublului buffer de imagine*. Comutarea între bufferele de imagine se poate sincroniza cu cursa de revenire pe verticală a monitorului și atunci imaginile sunt prezentate continuu, fără să se observe imagini fragmentate sau pâlpâiri.

În OpenGL comutarea bufferelor este efectuată de funcția `SwapBuffers()`. Această funcție trebuie să fie apelată la terminarea generării imaginii tuturor obiectelor vizibile în fiecare cadru. Modul în care se execută comutarea bufferelor depinde de platforma hardware.

Operațiile de bază OpenGL

OpenGL desenează primitive geometrice (puncte, linii și poligoane) în diferite moduri selectabile. Primitivele sunt definite printr-un grup de unul sau mai multe vârfuri (*vertices*). Un vârf definește un punct, capătul unei linii sau vârful unui poligon. Fiecare vârf are asociat un set de date:

coordonate,
culoare,
normală,
coordonate de textură.

Aceste date sunt prelucrate independent, în ordine și în același mod pentru toate primitivele geometrice. Singura deosebire care apare este aceea că, dacă o linie sau o suprafață este decupată, atunci grupul de vârfuri care descriu inițial primitiva respectivă este înlocuit cu un alt grup, în care pot apare vârfuri noi rezultate din intersecția laturilor cu planele volumului de decupare (volumul canonic) sau unele vârfuri din cele inițiale pot să dispară.

Comenzile sunt prelucrate în ordinea în care au fost primite în OpenGL, adică orice comandă este complet executată înainte ca să fie executată o nouă comandă.

Biblioteca OpenGL primește o succesiune de primitive geometrice pe care le desenează, adică le convertește în mulțimea de pixeli corespunzătoare, înscrind valorile culorilor acestora într-un buffer de imagine. În continuare sunt detaliate fiecare dintre operațiile grafice prezentate în figură.

Modul în care este executată secvența de operații pentru redarea primitivelor grafice depinde de starea bibliotecii OpenGL, stare care este definită prin mai multe variabile de stare ale acesteia (parametri).

Numărul de variabile de stare ale bibliotecii - *OpenGL Reference Manual*

La inițializare, fiecare variabilă de stare este setată la o valoare implicită. O stare o dată setată își menține valoarea neschimbată până la o nouă setare a acesteia. Variabilele de stare au denumiri sub formă de constante simbolice care pot fi folosite pentru aflarea valorilor acestora.

Primitive geometrice

OpenGL execută secvența de operații grafice asupra fiecărei primitive geometrice, definită printr-o mulțime de vârfuri și tipul acesteia. Coordonatele unui vârf sunt transmise către OpenGL prin apelul unei funcții `glVertex#()`. Aceasta are mai multe variante, după numărul și tipul argumentelor. Iată, de exemplu, numai câteva din prototipurile funcțiilor `glVertex#()`:

```
void glVertex2d(GLdouble x, GLdouble y);  
void glVertex3d(GLdouble x, GLdouble y, GLdouble z);  
void glVertex3f(GLfloat x, GLfloat y, GLfloat z);
```

Vârfurile pot fi specificate în plan, în spațiu (sau în coordonate omogene), folosind apelul funcției corespunzătoare.

O primitivă geometrică se definește printr-o mulțime de vârfuri (care dau descrierea geometrică a primitivei) și printr-unul din tipurile prestabilite, care indică topologia, adică modul în care sunt conectate vârfurile între ele.

Mulțimea de vârfuri este delimitată între funcțiile `glBegin()` și `glEnd()`. Aceeași mulțime de vârfuri ($v_0, v_1, v_2, \dots, v_{n-1}$) poate fi tratată ca puncte izolate, linii, poligon, etc, în funcție de tipul primitivei, care este transmis ca argument al funcției `glBegin()`:

```
void glBegin(GLenum mode);
```

Exista mai multe tipuri de primitive pe care le putem desena folosind OpenGL.

Tipurile de primitive geometrice

Argument	Primitivă geometrică
GL_POINTS	Desenează n puncte
GL_LINES	Desenează segmentele de dreaptă izolate (v_0, v_1), (v_2, v_3),... ș.a.m.d. Dacă n este impar ultimul vârf

	este ignorat
GL_LINE_STRIP	Desenează linia poligonală formată din segmentele $(v_0, v_1), (v_1, v_2), \dots, (v_{n-2}, v_{n-1})$
GL_LINE_LOOP	La fel ca primitiva GL_LINE_STRIP, dar se mai desenează segmentul (v_n, v_0) care închide o buclă.
GL_TRIANGLES	Desenează o serie de triunghiuri folosind vârfurile $(v_0, v_1, v_2), (v_3, v_4, v_5), \dots$ ș.a.m.d. Dacă n nu este multiplu de 3, atunci ultimele 1 sau 2 vârfuri sunt ignorate.
GL_TRIANGLE_STRIP	Desenează o serie de triunghiuri folosind vârfurile $(v_0, v_1, v_2), (v_2, v_1, v_3), \dots$ ș.a.m.d. Ordinea este aleasă astfel ca triunghiurile să aibă aceeași orientare, deci să poată forma o suprafață închisă.
GL_TRIANGLE_FAN	Desenează triunghiurile $(v_0, v_1, v_2), (v_0, v_2, v_3), \dots$ ș.a.m.d.
GL_QUADS	Desenează o serie de patrulatere $(v_0, v_1, v_2, v_3), (v_4, v_5, v_6, v_7), \dots$ ș.a.m.d. Dacă n nu este multiplu de 4, ultimele 1, 2 sau 3 vârfuri sunt ignorate.
GL_QUADS_STRIP	Desenează o serie de patrulatere $(v_0, v_1, v_3, v_2), (v_3, v_2, v_5, v_4), \dots$ ș.a.m.d. Dacă $n < 4$, nu se desenează nimic. Dacă n este impar, ultimul vârf este ignorat.
GL_POLYGON	Desenează un poligon cu n vârfuri, $(v_0, v_1, \dots, v_{n-1})$. Dacă poligonul nu este convex, rezultatul este impredictibil.

Primitivele de tip suprafață (triunghiuri, patrulatere, poligoane) pot fi desenate în modul “cadru de sârmă” (*wireframe*), sau în modul plin (*fill*), prin setarea variabilei de stare GL_POLYGON_MODE folosind funcția:

```
void glPolygonMode(GLenum face, GLenum mode);
```

unde argumentul mode poate lua una din valorile:

GL_POINT : se desenează numai vârfurile primitivei, ca puncte în spațiu, indiferent de tipul acesteia.

GL_LINE: muchiile poligoanelor se desenează ca segmente de dreaptă.

GL_FILL: se desenează poligonul plin.

Argumentul face se referă la tipul primitivei geometrice (din punct de vedere al orientării) căreia i se aplică modul de redare mode. Din punct de vedere al orientării, OpenGL admite primitive orientate direct (frontface) și primitive orientate invers (backface). Argumentul face poate lua una din valorile: GL_FRONT, GL_BACK sau GL_FRONT_AND_BACK, pentru a se specifica primitive orientate direct, primitive orientate invers și, respectiv ambele tipuri de primitive.

În mod implicit, sunt considerate orientate direct suprafețele ale căror vârfuri sunt parcurse în ordinea inversă acelor de ceas. Această setare se poate modifica prin funcția glFrontFace(GLenum mode), unde mode poate lua valoarea GL_CCW, pentru orientare în sens invers acelor de ceas (*counterclockwise*) sau GL_CW, pentru orientare în sensul acelor de ceasornic (*clockwise*).

Reprezentarea culorilor în OpenGL

În biblioteca OpenGL sunt definite două modele de culori: modelul de culori RGBA și modelul de culori indexate. În modelul RGBA sunt memorate componentele de culoare R, G, B și transparența A pentru fiecare primitivă geometrică sau pixel al imaginii. În modelul de culori indexate, culoarea primitivelor geometrice sau a pixelilor este reprezentată printr-un index într-o tabelă de culori (*color map*), care are memorate pentru fiecare intrare (index) componentele corespunzătoare R,G,B,A ale culorii. În modul de culori indexate numărul de culori afișabile simultan este limitat de dimensiunea tabelii culorilor și, în plus, nu se pot

efectua unele dintre prelucrările grafice importante (cum sunt umbrirea, antialiasing, ceața). De aceea, modelul de culori indexate este folosit în principal în aplicații de proiectare grafică (CAD) în care este necesar un număr mic de culori și nu se folosesc umbrirea, ceața, etc.

În aplicațiile de realitate virtuală nu se poate folosi modelul de culori indexate, de aceea în continuare nu vor mai fi prezentate comenzile sau opțiunile care se referă la acest model și toate descrierile consideră numai modelul RGBA.

Culoarea care se atribuie unui pixel dintr-o primitivă geometrică depinde de mai multe condiții, putând fi o culoare constantă a primitivei, o culoare calculată prin interpolare între culorile vârfurilor primitivei, sau o culoare calculată în funcție de iluminare, anti-aliasing și texturare. Presupunând pentru moment culoarea constantă a unei primitive, aceasta se obține prin setarea unei variabile de stare a bibliotecii, variabila de culoare curentă (GL_CURRENT_COLOR). Culoarea curentă se setează folosind una din funcțiile glColor#(), care are mai multe variante, în funcție de tipul și numărul argumentelor. De exemplu, câteva din prototipurile acestei funcții definite în gl.h sunt:

```
void glColor3f(GLfloat red, GLfloat green, GLfloat blue);  
void glColor3ub(GLubyte red, GLubyte green, GLubyte blue);  
void glColor4d(GLdouble red, GLdouble green,  
GLdouble blue, GLdouble alpha);
```

Culoarea se poate specifica prin trei sau patru valori, care corespund componentelor roșu (*red*), verde (*green*), albastru (*blue*), respectiv transparență (*alpha*) ca a patra componentă pentru funcțiile cu 4 argumente. Fiecare componentă a culorii curente este memorată ca un număr în virgulă flotantă cuprins în intervalul [0,1]. Valorile argumentelor de tip întreg fără semn (unsigned int, unsigned char, etc.) sunt convertite liniar în numere în virgulă flotantă, astfel încât valoarea 0 este transformată în 0.0, iar valoarea maximă reprezentabilă este transformată în 1.0 (intensitate maximă). Valorile argumentelor de tip întreg cu semn sunt convertite liniar în numere în virgulă flotantă, astfel încât valoarea negativă maximă este transformată în -1.0, iar valoarea pozitivă maximă este transformată în 1.0 (intensitate maximă).

Valoarea finală a culorii unui pixel, rezultată din toate calculele de umbrire, anti-aliasing, texturare, etc, este memorată în bufferul de imagine, o componentă fiind reprezentată printr-un număr n de biți (nu neapărat același număr de biți pentru fiecare componentă). Deci componentele culorilor pixelilor memorate în bufferul de imagine sunt numere întreg în intervalul $(0, 2^n - 1)$, care se obțin prin conversia numerelor în virgulă mobilă prin care sunt reprezentate și prelucrate culorile primitivelor geometrice, ale materialelor, etc.

Sistemul de vizualizare OpenGL

Sistemul de vizualizare OpenGL definește sistemele de referință, transformările geometrice și relațiile (matriceale) de transformare pentru redarea primitivelor geometrice. Sistemul de vizualizare OpenGL este o particularizare a sistemului PHIGS, în care centrul sistemului de referință de observare (VRP) coincide cu centrul de proiecție (PRP).

Transformări geometrice

Așa după cum s-a mai arătat, nu este eficient să se calculeze produsul dintre matricea de reprezentare a fiecărui punct și matricele de transformări succesive, ci se calculează o matrice de transformare compusă, care se poate aplica unuia sau mai multor obiecte. Pentru calculul matricelor de transformare compuse (prin produs de matrice) se folosește o *matrice de transformare curentă* și operații de acumulare în matricea curentă prin postmultiplicare (înmulțire la dreapta): se înmulțește matricea curentă cu noua matrice (în această ordine) și rezultatul înlocuiește conținutul matricei curente. Pentru început, se consideră o matrice curentă oarecare **C** stabilită în OpenGL. Matricea curentă se poate inițializa cu matricea identitate prin funcția glLoadIdentity(), sau cu o matrice oarecare, dată prin pointer la un vector de 16 valori consecutive, prin funcția glLoadMatrix#():

```
glLoadMatrixd(const GLdouble* m);  
glLoadMatrixf(const GLfloat* m);
```

Valorile din vectorul GLdouble* m (respectiv GLfloat* m) sunt atribuite în ordinea coloană

majoră matricei curente.

Conținutul matricei curente poate fi modificat prin multiplicare (la dreapta) cu o altă matrice, dată printr-un vector de 16 valori de tip double sau float utilizând funcția `glMultMatrix#()`:

```
glMultMatrixd(const GLdouble* m);
```

```
glMultMatrixf(const GLfloat* m);
```

Transformarea de translație cu valorile x , y , z se implementează prin apelul uneia din funcțiile `glTranslated()` sau `glTranslatef()`, după tipul argumentelor de apel:

```
glTranslated(GLdouble x, GLdouble y, GLdouble z);
```

```
glTranslatef(GLfloat x, GLfloat y, GLfloat z);
```

Funcția `glTranslate#()` creează o matrice de translație $T(x,y,z)$, dată de relația 3.2, înmulțește la dreapta matricea curentă C , iar rezultatul înlocuiește conținutul matricei curente C , deci: $C = C T(x,y,z)$.

Transformarea de scalare este efectuată de una din funcțiile `glScaled()` sau `glScalef()`:

```
glScaled(GLdouble x, GLdouble y, GLdouble z);
```

```
glScalef(GLfloat x, GLfloat y, GLfloat z);
```

Funcția `glScale#()` crează o matrice de scalare $S(x,y,z)$, și o înmulțește cu matricea curentă, rezultatul fiind depus în matricea curentă.

Transformarea de rotație se definește printr-o direcție de rotație dată prin vectorul de poziție x,y,z și un unghi $angle$ (specificat în grade). Prototipurile funcțiilor de rotație sunt:

```
glRotated(GLdouble angle, GLdouble x,
```

```
GLdouble y, GLdouble z);
```

```
glRotatef(GLfloat angle, GLfloat x,
```

```
GLfloat y, GLfloat z);
```

Rotațiile în raport cu axele de coordonate sunt cazuri particulare ale funcțiilor `glRotate#()`. De exemplu, rotația cu unghiul $angle$ în raport cu axa x se obține la apelul funcției `glRotated(angle,1,0,0)` sau `glRotatef(angle,1,0,0)`.

3.3 Aplicații rezolvate

Exemplu 1

```
#include <stdio.h>
#include <gl/glut.h>

void Display(void)
{
}

int main(void)
{
    // Create main Window
    glutCreateWindow("This is the window title");

    // Set Call Back Window
    glutDisplayFunc(Display);

    // Window Message Loop
    glutMainLoop();
    return 0;
}
```

Exemplu 2

```
#include <stdio.h>
#include <GL/GLUT.H>

void Display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glFlush();
}

int main(void)
{
    glutCreateWindow("exemplu");
    glutDisplayFunc(Display);
    glClearColor(1.0, 0.0, 0.0);
    glutMainLoop();
    return 0;
}
```

Exemplu 3

```
#include <stdio.h>
#include <GL/GLUT.H>

void Display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    glBegin(GL_POINTS);
        glVertex3f(0.0, 0.0, 0);
    glEnd();

    glFlush();
}

int main(void)
{
    glutCreateWindow("exemplu");
    glutDisplayFunc(Display);
    glutMainLoop();
    return 0;
}
```

Exemplu 4

```
#include <GL/glut.h>
void init()
{
    glClearColor (0.0, 0.0, 0.0, 0.0);    glPointSize(40.0);
    glShadeModel (GL_FLAT);
}
void display()
{
    glClear (GL_COLOR_BUFFER_BIT);

    glColor3f (1.0, 0.0, 0.0);    glBegin(GL_POINTS);
        glVertex2i(100, 300);
        glVertex2i(200,300);        glVertex2i(200,400);
    glVertex2i(0, 0);
    glEnd();    glFlush ();
}
void reshape (int w, int h)
{
    glViewport (0, 0, (GLsizei) w, (GLsizei) h);
    glMatrixMode (GL_PROJECTION);
    glLoadIdentity ();
    gluOrtho2D (0.0, (GLdouble) w, 0.0, (GLdouble) h);
}
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize (400, 400);
    glutInitWindowPosition (0,0);
    glutCreateWindow ("puncte");
    init ();
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutMainLoop();
    return 0;
}
```

Exemplu 5

```
#include <stdio.h>
#include <GL/GLUT.H>

void Display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    glBegin(GL_POINTS);
        glVertex3f(0,0,0);
        glVertex3f(1,1,0);
        glVertex3f(-1,1,0);
        glVertex3f(1,-1,0);
        glVertex3f(-1,-1,0);
    glEnd();

    glFlush();
}

int main(void)
{
    glutCreateWindow("exemplu");
    glutDisplayFunc(Display);

    // Set Point Size to 10 from default 1
    glPointSize(10);

    glutMainLoop();
    return 0;
}
```

Exemplu 6

```
#include <stdio.h>
#include <math.h>
#include <GL/GLUT.H>

void Display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    glBegin(GL_POINTS);
        for(int i=0;i<1000;++i)
        {
            glVertex3f(cos(2*3.14159*i/1000.0),sin(2*3.14159*i/1000.0),0);
        }
    glEnd();

    glFlush();
}

int main(void)
{
    glutCreateWindow("Exemplu");
    glutDisplayFunc(Display);
    glutMainLoop();
    return 0;
}
```

Exemplu 7

```
#include <stdio.h>
#include <math.h>
#include <GL/GLUT.H>

void Display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    glBegin(GL_LINES);
    glVertex3f(-1,0,0);
    glVertex3f(+1,0,0);
    glEnd();

    glFlush();
}

int main(void)
{
    glutCreateWindow("Exemplu");
    glutDisplayFunc(Display);
    glutMainLoop();
    return 0;
}
```

Exemplu 8

```
#include <stdio.h>
#include <math.h>
#include <GL/GLUT.H>

void Display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    glBegin(GL_LINES);
    for(int i=0; i<100; ++i)
    {
        glVertex3f(0,0,0);
        glVertex3f(1-i/100.0, i/100.0, 0);
    }
    glEnd();

    glFlush();
}

int main(void)
{
    glutCreateWindow("Exemplu");
    glutDisplayFunc(Display);
    glColor3f(1,0,0);
    glutMainLoop();
    return 0;
}
```

Exemplu 9

```
#include <stdio.h>
#include <math.h>
#include <GL/GLUT.H>

void Display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    glBegin(GL_LINE_STRIP);
    glVertex3f(-1,0,0);
    glVertex3f(0,0,0);
    glVertex3f(0,1,0);
    glVertex3f(1,0,0);
    glEnd();

    glFlush();
}

int main(void)
{
    glutCreateWindow("Exemplu");
    glutDisplayFunc(Display);
    glutMainLoop();
    return 0;
}
```

Exemplu 10

```
#include <stdio.h>
#include <math.h>
#include <GL/GLUT.H>

void Display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    glBegin(GL_LINE_LOOP);
    glVertex3f(-1,0,0);
    glVertex3f(0,0,0);
    glVertex3f(0,1,0);
    glEnd();

    glFlush();
}

int main(void)
{
    glutCreateWindow("Exemplu");
    glutDisplayFunc(Display);
    glutMainLoop();
    return 0;
}
```

Exemplu 11

```
#include <stdio.h>
#include <math.h>
#include <GL/GLUT.H>

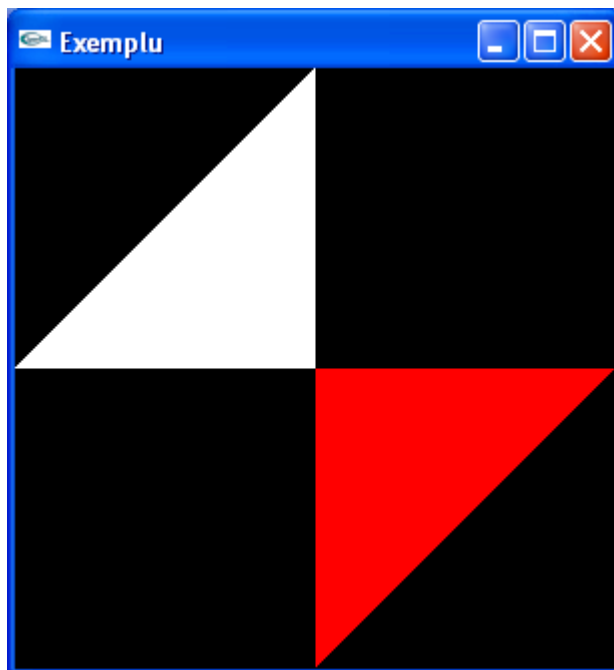
void Display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    glBegin(GL_TRIANGLES);
        glColor3f(1,1,1);
        glVertex3f(0,0,0);
        glVertex3f(-1,0,0);
        glVertex3f(0,1,0);

        glColor3f(1,0,0);
        glVertex3f(0,0,0);
        glVertex3f(1,0,0);
        glVertex3f(0,-1,0);
    glEnd();

    glFlush();
}

int main(void)
{
    glutCreateWindow("Exemplu");
    glutDisplayFunc(Display);
    glutMainLoop();
    return 0;
}
```



Exemplu 12

```
#include <stdio.h>
#include <math.h>
#include <GL/GLUT.H>

void Display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    glBegin(GL_TRIANGLE_STRIP);
    glVertex3f(0,0,0);
    glVertex3f(1,0,0);
    glVertex3f(0,1,0);
    glVertex3f(0.9,0.9,0);
    glVertex3f(1,-1,0);
    glEnd();

    glFlush();
}

int main(void)
{
    glutCreateWindow("Exemplu");
    glutDisplayFunc(Display);
    glutMainLoop();
    return 0;
}
```

Exemplu 13

```
#include <stdio.h>
#include <math.h>
#include <GL/GLUT.H>

void Display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    glBegin(GL_TRIANGLE_STRIP);
    glVertex3f(0,0,0);
    glVertex3f(1,0,0);
    glVertex3f(0,1,0);
    glVertex3f(-1,0,0);
    glVertex3f(-1,-1,0);
    glEnd();

    glFlush();
}

int main(void)
{
    glutCreateWindow("Exemplu");
    glutDisplayFunc(Display);
    glutMainLoop();
    return 0;
}
```

Exemplu 14

```
#include <stdio.h>
#include <math.h>
#include <GL/GLUT.H>

void Display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    glBegin(GL_POLYGON);
    glVertex3f(0,0,0);
    glVertex3f(1,0,0);
    glVertex3f(0,1,0);
    glVertex3f(-1,1/2,0,0);
    glVertex3f(0,-1,0);
    glEnd();

    glFlush();
}

int main(void)
{
    glutCreateWindow("Exemplu");
    glutDisplayFunc(Display);
    glutMainLoop();
    return 0;
}
```

Exemplu 15

```
#include <stdio.h>
#include <math.h>
#include <GL/GLUT.H>

void Display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    glBegin(GL_QUAD_STRIP);
    glVertex3f(0,1,0);
    glVertex3f(0,0,0);

    glVertex3f(0.1,1,0);
    glVertex3f(0.1,0,0);

    glVertex3f(0.2,0.9,0);
    glVertex3f(0.2,-0.1,0);

    glEnd();

    glFlush();
}

int main(void)
{
    glutCreateWindow("Exemplu");
    glutDisplayFunc(Display);
    glutMainLoop();
    return 0;
}
```

Exemplu 16

```
#include <stdio.h>
#include <math.h>
#include <GL/GLUT.H>

void Display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    glBegin(GL_LINES);
    glColor3f(1,0,0);
    glVertex3f(0,0,0);
    glVertex3f(1,0,0);

    glColor3f(0,1,0);
    glVertex3f(0,0,0);
    glVertex3f(0,1,0);

    glColor3f(0,0,1);
    glVertex3f(0,0,0);
    glVertex3f(0,0,1);
    glEnd();

    glFlush();
}

int main(void)
{
    glutCreateWindow("Exemplu");
    glutDisplayFunc(Display);
    glutMainLoop();
    return 0;
}
```

Exemplu 17

```
#include <stdio.h>
#include <math.h>
#include <GL/GLUT.H>
void Display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    glLoadIdentity();
    glTranslated(0.5,-0.5,0);

    glBegin(GL_LINES);
    glColor3f(1,0,0);
    glVertex3f(0,0,0);
    glVertex3f(1,0,0);

    glColor3f(0,1,0);
    glVertex3f(0,0,0);
    glVertex3f(0,1,0);

    glColor3f(0,0,1);
    glVertex3f(0,0,0);
    glVertex3f(0,0,1);
    glEnd();

    glFlush();
}

int main(void)
{
    glutCreateWindow("Exemplu");
    glutDisplayFunc(Display);

    glMatrixMode(GL_MODELVIEW);

    glutMainLoop();
    return 0;
}
```

Exemplu 18

```
#include <stdio.h>
#include <math.h>
#include <GL/GLUT.H>

void Display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    glLoadIdentity();
    glRotated(45,0,0,1);

    glBegin(GL_LINES);
    glColor3f(1,0,0);
    glVertex3f(0,0,0);
    glVertex3f(1,0,0);

    glColor3f(0,1,0);
    glVertex3f(0,0,0);
    glVertex3f(0,1,0);

    glColor3f(0,0,1);
    glVertex3f(0,0,0);
    glVertex3f(0,0,1);
    glEnd();

    glFlush();
}

int main(void)
{
    glutCreateWindow("Exemplu");
    glutDisplayFunc(Display);

    glMatrixMode(GL_MODELVIEW);

    glutMainLoop();
    return 0;
}
```

Exemplu 19

```
#include <stdio.h>
#include <math.h>
#include <GL/GLUT.H>

void Display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    glLoadIdentity();
    glScaled(0.3,0.7,0.4);

    glBegin(GL_LINES);
    glColor3f(1,0,0);
    glVertex3f(0,0,0);
    glVertex3f(1,0,0);

    glColor3f(0,1,0);
    glVertex3f(0,0,0);
    glVertex3f(0,1,0);

    glColor3f(0,0,1);
    glVertex3f(0,0,0);
    glVertex3f(0,0,1);
    glEnd();

    glFlush();
}

int main(void)
{
    glutCreateWindow("Exemplu");
    glutDisplayFunc(Display);

    glMatrixMode(GL_MODELVIEW);

    glutMainLoop();
    return 0;
}
```

Tema: Sa se realizeze aplicatiile de la 1-19 si sa se comenteze functiile OpenGL

Exemplu 20

```
//puncte.cpp
#include <GL/glut.h>
void init()
{
    glClearColor (0.0, 0.0, 0.0, 0.0);
    //glPointSize(40.0);
    glShadeModel (GL_FLAT); }
void display()
{ glClear (GL_COLOR_BUFFER_BIT);
glBegin(GL_POLYGON);//initializare desen poligon
    glVertex2f(0.0,0.0); //stabilire coordonate triunghi
    glVertex2f(200.0,200.0);//stabilire coordonate triunghi
    glVertex2f(0.0,200.0);//stabilire coordonate triunghi
    glEnd();//sfisit desenare
    //executare functie
glFlush (); glPointSize(40.0);
    glColor3f (1.0, 0.0, 0.0);
    glBegin(GL_POINTS);
    glVertex2i(100, 300);
    glVertex2i(200,300);
        glPointSize(40.0);
    glVertex2i(200,400);
        glVertex2i(0, 0);
    glEnd(); glFlush (); }
void reshape (int w, int h)//functia redesenare
{
    glViewport (0, 0, (GLsizei) w, (GLsizei) h);//stabilirea viewportului la dimensiunea ferestrei
    glMatrixMode (GL_PROJECTION);//specificare matrice modificabila la valoare argumentului de modificare
    glLoadIdentity ();//initializarea sistemului de coordonate
    gluOrtho2D (0.0, (GLdouble) w, 0.0, (GLdouble) h);//stabileste volumul de vedere folosind o proiectie ortografica
}
void main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize (400, 400);
    glutInitWindowPosition (150,150);
    glutCreateWindow ("puncte");
    init (); glutDisplayFunc(display); glutReshapeFunc(reshape);
    glutMainLoop(); }
```

Exemplu 21

```
#include <GL/glut.h> //header OpenGL ce include gl.h si glu.h
void init()//functia initiere
{
    // glClearColor (0.0, 0.0, 0.0, 0.0);//stabileste culoarea de sters
    // glShadeModel (GL_FLAT);
}
```

```

void display()//functia de desenare si afisare
{
    glClear (GL_COLOR_BUFFER_BIT);//sterge urmele de desene din fereastra curenta
    glBegin(GL_POLYGON);//initializare desen poligon
    glColor3f (2.0, 0.0, 0.0);//culoarea de desenare
    glVertex2f(200.0,200.0);//stabilire coordonate triunghi
    glVertex2f(400.0,200.0);//stabilire coordonate triunghi
    glVertex2f(400.0,400.0);//stabilire coordonate triunghi
    glEnd();//sfisit desenare
    glFlush();//executare functie
    glColor3f (1.0, 1.0, 0.0);//culoarea de desenare
    glBegin(GL_POLYGON);//initializare desen poligon
    glVertex2f(200.0,400.0);//stabilire coordonate triunghi
    glVertex2f(400.0,400.0);//stabilire coordonate triunghi
    glVertex2f(200.0,200.0);//stabilire coordonate triunghi
    glEnd();//sfisit desenare
    glFlush();//executare functie
    glColor3f (0.0, 1.0, 1.0);//culoarea de desenare
    glBegin(GL_POLYGON);//initializare desen poligon
    glVertex2f(000.0,000.0); //stabilire coordonate triunghi
    glVertex2f(200.0,200.0);//stabilire coordonate triunghi
    glVertex2f(000.0,200.0);//stabilire coordonate triunghi
    glEnd();//sfisit desenare
    glFlush();//executare functie
    glColor3f (1.0, 0.5, 1.0);//culoarea de desenare
    glBegin(GL_POLYGON);//initializare desen poligon
    glVertex2f(000.0,000.0); //stabilire coordonate triunghi
    glVertex2f(200.0,200.0);//stabilire coordonate triunghi
    glVertex2f(000.0,200.0);//stabilire coordonate triunghi
    glEnd();//sfisit desenare
    glFlush();//executare functie
    glColor3f (1.0, 1.0, 0.5);//culoarea de desenare
    glBegin(GL_POLYGON);//initializare desen poligon
    glVertex2f(600.0,000.0); //stabilire coordonate triunghi
    glVertex2f(600.0,200.0);//stabilire coordonate triunghi
    glVertex2f(400.0,200.0);//stabilire coordonate triunghi
    glEnd();//sfisit desenare
    glFlush();//executare functie
    glColor3f (0.0, 1.0, 0.0);//culoarea de desenare
    glBegin(GL_POLYGON);//initializare desen poligon
    glVertex2f(200.0,400.0);//stabilire coordonate triunghi
    glVertex2f(000.0,400.0);//stabilire coordonate triunghi
    glVertex2f(200.0,600.0);//stabilire coordonate triunghi
    glEnd();//sfisit desenare
    glFlush();//executare functie
    glColor3f (0.0, 1.0, 1.0);//culoarea de desenare
    glBegin(GL_POLYGON);//initializare desen poligon
    glVertex2f(400.0,400.0); //stabilire coordonate triunghi
    glVertex2f(600.0,400.0);//stabilire coordonate triunghi
    glVertex2f(600.0,600.0);//stabilire coordonate triunghi
    glEnd();//sfisit desenare
    glFlush();//executare functie
}

```

```

    glColor3f(3.0, 0.0, 0.0); //culoarea de desenare
    glBegin(GL_POLYGON); //initializare desen poligon
    glVertex2f(400.0, 600.0); //stabilire coordonate triunghi
    glVertex2f(400.0, 400.0); //stabilire coordonate triunghi
    glVertex2f(600.0, 600.0); //stabilire coordonate triunghi
    glEnd(); //sfisit desenare
    glFlush(); //executare functie
}
void reshape(int w, int h) //functia redesenare
{
    glViewport(0, 0, (GLsizei) w, (GLsizei) h); //stabilirea viewportului la dimensiunea ferestrei
    glMatrixMode(GL_PROJECTION); //specificare matrice modificabila la valoare argumentului de modificare
    glLoadIdentity(); //initializarea sistemului de coordonate
    gluOrtho2D(0.0, (GLdouble) w, 0.0, (GLdouble) h); //stabileste volumul de vedere folosind o proiectie ortografica
}
int main(int argc, char** argv) //creare fereastră
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB); //se specifica modelul de culoare al ferestrei: un singur buffer si culoare RGB
    glutInitWindowSize(600, 600); //initiaza dimensiunea ferestrei principale 600x600 pixeli
    glutInitWindowPosition(200, 10); //initiaza in fereastră principala fereastră de afisare
    glutCreateWindow("TRIUNGHIURI");
    init();
    glutDisplayFunc(display); //se reimprospăteaza continutul ferestrei
    glutReshapeFunc(reshape); //functia redesenare
    glutMainLoop(); //bucula de procesare a evenimentelor
    return 0; }

```

Exemplu 22

```

#include <GL/glut.h>
#include <stdlib.h>
#include <math.h>
void init() {
    // Stabileste culoarea cu care se va curata interiorul ferestrei, prin parametrii dati stabilindu-se
    // componentele culorii ce au valori reale de la 0.0 la 1.0 (cu pasi aproximativi de 0.00006).
    glClearColor(0.0, 0.0, 0.0, 0.0);
    // select flat or smooth shading.
    glShadeModel(GL_FLAT);
}
void display() {
    // Sterge eventualele desene din fereastră curenta
    glClear(GL_COLOR_BUFFER_BIT);
    // stabileste culoarea (RGB)
    glColor3f(1.0, 0.0, 0.0);
    // Specifies the primitive or primitives that will be created from vertices

```

presented between

```
// glBegin and the subsequent glEnd. Ten symbolic constants are accepted:
GL_POINTS, GL_LINES,
// GL_LINE_STRIP, GL_LINE_LOOP, GL_TRIANGLES, GL_TRIANGLE_STRIP,
GL_TRIANGLE_FAN, GL_QUADS,
// GL_QUAD_STRIP, and GL_POLYGON.
glBegin(GL_POLYGON);
// specify a vertex (x, y coord)
glVertex2f(200.0,200.0);
glVertex2f(400.0,200.0);
glVertex2f(400.0, 400.0);
// glBegin and glEnd delimit the vertices that define a primitive or a group of like
primitives.
```

// glBegin accepts a single argument that specifies in which of ten ways the vertices are interpreted.

```
glEnd();
// force execution of GL commands in finite time
glFlush();
// dreptunghi jos
glColor3f(0.0, 1.0, 0.0);
glBegin(GL_POLYGON);
glVertex2f(0.0,0.0);
glVertex2f(50.0,0.0);
glVertex2f(50.0, 60.0);
glVertex2f(0.0, 60.0);
glEnd();
glFlush();
// patrat sus
glColor3f(0.0, 0.0, 1.0);
glBegin(GL_POLYGON);
glVertex2f(0.0,500.0);
glVertex2f(100.0,500.0);
glVertex2f(100.0, 600.0);
glVertex2f(0.0, 600.0);
glEnd();
glFlush();
// cerc in jurul triunghiului
glColor3f(1, 1, 0); glPointSize(10.0);
glBegin(GL_POINTS);
glVertex2f(300, 300); // un punct in centru
for (int i=0; i<360; ++i) {
    // 141 = raza cercului + 1 pixel
    glVertex2f(300 + sin(i) * 141, 300 + cos(i) * 141);
}
glEnd();
glFlush();
}
```

```
void reshape (int w, int h) {
```

```
// Stabilirea viewportului la dimensiunea ferestrei
glViewport (0, 0, (GLsizei) w, (GLsizei) h);
// specify which matrix is the current matrix
```



```

    glMatrixMode (GL_PROJECTION);
    // replace the current matrix with the identity matrix
    glLoadIdentity ();
    gluOrtho2D (0.0, (GLdouble) w, 0.0, (GLdouble) h);
}
int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
    // seteaza dimensiunea ferestrei
    glutInitWindowSize (600, 600);
    // seteaza pozitia initiala
    glutInitWindowPosition (100,100);
    glutCreateWindow ("TESTE");
    init ();
        glutDisplayFunc(display);
        glutReshapeFunc(reshape);
        glutMainLoop();
    return 0;
}

```

4. Realizarea desenelor 2D / 3D folosind OpenGL

Obiective generale:

- Utilizarea funcțiilor de lucru cu ferestre in OpenGL
- Gestionarea utilizării meniurilor in OpenGL;
- Manipularea obiectelor predefinite 3D in OpenGL;
- Cunoasterea foarte a bine functii de baza OpenGL; Recapitulare Functii;
- Intelegerea si rezolvarea problemelor propuse/rezolvate;

Cuprins:

4.1 Functii pentru lucru cu ferestre;
 4.2 Crearea meniurilor in OpenGL
 4.3 Obiecte predefinite 3D;
 4.4 Recapitulare Functii;
 4.5 Probleme rezolvate
 4.6. Probleme propuse

4.1 Functii pentru lucru cu ferestre

Creare fereastră

```
int glutCreateWindow(char *string);
```

Funcția **glutCreateWindow** creează o fereastră cu un context OpenGL. Ea întoarce un

identificator unic pentru fereastra nou creată. Fereastra nu va fi afișată înainte de apelarea funcției **glutMainLoop**. Valoarea întoarsă de funcție reprezintă identificatorul ferestrei, care este unic.

Creare fereastra copil

```
int glutCreateSubWindow(int win, int x, int y, int width, int height);
```

Funcția creează o fereastră având ca părinte fereastra identificată de **win**, unde :

- **win** - reprezintă identificatorul ferestrei părinte;
- (**x**, **y**) - reprezintă colțul stânga sus al ferestrei (x și y sunt exprimate în pixeli și sunt relative la originea ferestrei părinte);
- **width** - reprezintă lățimea ferestrei (exprimată în pixeli);
- **height** - reprezintă înălțimea ferestrei (exprimată în pixeli);

Fereastra nou creată devine fereastra curentă. Funcția întoarce identificatorul ferestrei create.

Exemplu :

```
int winMain, winSub ;
```

```
winMain= glutCreateWindow("My Main window");  
winSub = glutCreateSubWindow(winMain, xtop, ytop, width,  
height);
```

The first parameter of glutCreatesubWindow is just the index of the parent window. As you know glutCreateWindow returns an integer. You must use that integer in glutCreatesubWindow to tell who is the parent window.

<http://www.opengl.org/resources/libraries/glut/spec3/node17.html>

Distrugere fereastră

```
void glutDestroyWindow(int win) ;
```

Funcția distruge fereastra specificată de **win**. De asemenea, este distrus și contextul OpenGL asociat ferestrei. Orice subfereastră a ferestrei distruse va fi de asemenea distrusă. Dacă **win** identifică fereastra curentă, atunci ea va deveni invalidă.

<http://www.opengl.org/documentation/specs/glut/spec3/node19.html>

```
void glutDestroyWindow(int win);  
win Identifier of GLUT window to destroy.
```

Description

glutDestroyWindow destroys the window specified by win and the window's associated OpenGL context, logical colormap (if the window is color index), and overlay and related state (if an overlay has been established). Any subwindows of destroyed windows are also destroyed by glutDestroyWindow. If win was the *current window*, the *current window* becomes invalid (glutGetWindow will return zero).

Selectarea ferestrei curente

```
void glutSetWindow(int win) ;
```

Funcția selectează fereastra curentă ca fiind cea identificată de parametrul **win**.

Aflarea ferestrei curente

```
int glutGetWindow(void) ;
```

Funcția întoarce identificatorul ferestrei curente. Funcția întoarce 0 dacă nu există nici o fereastră curentă sau fereastra curentă a fost distrusă.

Selectarea cursorului asociat ferestrei curente

```
void glutSetCursor(int cursor) ;
```

exemplu

```
glutSetCursor(GLUT_CURSOR_INHERIT) ;
```

Funcția modifică cursorul asociat ferestrei curente transformându-l în cursorul specificat de parametrul **cursor**, care poate avea una din următoarele valori:

glutSetCursor changes the cursor image of the *current window*.

Usage

```
void glutSetCursor(int cursor);
```

cursor

Name of cursor image to change to.

GLUT_CURSOR_RIGHT_ARROW

Arrow pointing up and to the right.

GLUT_CURSOR_LEFT_ARROW

Arrow pointing up and to the left.

GLUT_CURSOR_INFO

Pointing hand.

GLUT_CURSOR_DESTROY

Skull & cross bones.

GLUT_CURSOR_HELP

Question mark.

GLUT_CURSOR_CYCLE

Arrows rotating in a circle.

GLUT_CURSOR_SPRAY

Spray can.

GLUT_CURSOR_WAIT

Wrist watch.

GLUT_CURSOR_TEXT

Insertion point cursor for text.

GLUT_CURSOR_CROSSHAIR

Simple cross-hair.

GLUT_CURSOR_UP_DOWN

Bi-directional pointing up & down.

GLUT_CURSOR_LEFT_RIGHT

Bi-directional pointing left & right.

GLUT_CURSOR_TOP_SIDE

Arrow pointing to top side.

GLUT_CURSOR_BOTTOM_SIDE

Arrow pointing to bottom side.

GLUT_CURSOR_LEFT_SIDE

Arrow pointing to left side.

GLUT_CURSOR_RIGHT_SIDE

Arrow pointing to right side.

GLUT_CURSOR_TOP_LEFT_CORNER

Arrow pointing to top-left corner.

GLUT_CURSOR_TOP_RIGHT_CORNER

Arrow pointing to top-right corner.

GLUT_CURSOR_BOTTOM_RIGHT_CORNER

Arrow pointing to bottom-right corner.

GLUT_CURSOR_BOTTOM_LEFT_CORNER

Arrow pointing to bottom-right corner.

GLUT_CURSOR_FULL_CROSSHAIR

Full-screen cross-hair cursor (if possible, otherwise GLUT_CURSOR_CROSSHAIR).

GLUT_CURSOR_NONE

Invisible cursor.

GLUT_CURSOR_INHERIT

Use parent's cursor.

4.2 Gestiunea meniurilor

Menu Management

<http://www.opengl.org/resources/libraries/glut/spec3/node35.html>

GLUT supports simple cascading pop-up menus. They are designed to let a user select various modes within a program. The functionality is simple and minimalistic and is meant to be that way. Do not mistake GLUT's pop-up menu facility with an attempt to create a full-featured user interface.

It is illegal to create or destroy menus, or change, add, or remove menu items while a menu (and any cascaded sub-menus) are in use (that is, popped up).

-
- [1 glutCreateMenu](#)
 - [2 glutSetMenu, glutGetMenu](#)
 - [3 glutDestroyMenu](#)
 - [4 glutAddMenuEntry](#)
 - [5 glutAddSubMenu](#)
 - [6 glutChangeToMenuEntry](#)
 - [7 glutChangeToSubMenu](#)
 - [8 glutRemoveMenuItem](#)
 - [9 glutAttachMenu, glutDetachMenu](#)

Crearea meniurilor

Meniurile create cu ajutorul GLUT-ului sunt meniuri simple, de tip pop-up în cascadă.

```
int glutCreateMenu(void (*func)(int value)) ;
```

Funcția creează un nou meniu pop-up și **întoarce identificadorul corespunzător, de tip întreg**. Identificatorii meniurilor **încep de la valoarea 1**. Valorile lor sunt separate de identificadorii ferestrelor.

- parametrul *func* specifică o **funcție callback** a aplicației, care va fi apelată de biblioteca GLUT la selectarea unei opțiuni din meniu.

Parametrul transmis funcției callback (*value*) specifică opțiunea de meniu selectată.

Adăugarea unei opțiuni într-un meniu

```
void glutAddMenuEntry(char* name, int value) ;
```

Funcția adaugă o nouă opțiune meniului curent, la sfârșitul listei de articole a meniului.

- *name* - specifică textul prin care va fi reprezentată opțiunea introdusă
- *value* - reprezintă valoarea transmisă funcției callback asociată meniului la selectarea opțiunii respective

glutAddMenuEntry

glutAddMenuEntry adds a menu entry to the bottom of the *current menu*.

Usage

```
void glutAddMenuEntry(char *name, int value);  
name
```

ASCII character string to display in the menu entry.

```
value
```

Value to return to the menu's callback function if the menu entry is selected.

Description

glutAddMenuEntry adds a menu entry to the bottom of the *current menu*. The string *name* will be displayed for the newly added menu entry. If the menu entry is selected by the user, the menu's callback will be called passing *value* as the callback's parameter.

Adăugarea unui submeniu într-un meniu

```
void glutAddSubMenu(char* name, int menu) ;
```

Funcția adaugă un submeniu la sfârșitul meniului curent.

- *name* - reprezintă numele submeniului introdus (textul prin care va fi afișat în meniu)
- *menu* - reprezintă identificatorul submeniului.

Ștergerea unei opțiuni sau a unui submeniu

```
void glutRemoveMenuItem(int entry) ;
```

Funcția șterge opțiunea sau submeniul identificat de parametrul *entry*. Opțiunile din meniu de sub opțiunea ștearsă sunt renumerotate.

Distrugerea unui meni

```
void glutDestroyMenu(int menu) ;
```

Funcția distruge meniul specificat prin parametru.

Observatie: dacă meniul distrus este cel curent, atunci meniul curent fi deveni invalid.

Setarea meniului curent

```
void glutSetMenu(int menu) ;
```

Funcția setează meniul curent ca fiind cel identificat de parametrul *menu*.

Aflarea meniului curent

```
int glutGetMenu(void) ;
```

Funcția întoarce identificatorul meniului curent. Funcția întoarce 0 dacă nu există meniul curent sau meniul curent a fost distrus.

Atașare / detașare meniul unui buton al mouse-ului

```
void glutAttachMenu(int button) ;  
void glutDetachMenu(int button) ;
```

Funcția **glutAttachMenu** atașează meniul curent butonului mouse-ului specificat de parametrul *button*. Funcția **glutDetachMenu** detașează butonul asociat meniului curent. **Prin atașarea unui buton al mouse-ului meniului curent, meniul va fi derulat la apăsarea butonului respectiv în poziția curentă a cursorului.**

4.3 Afișarea obiectelor 3D predefinite

GLUT conține funcții pentru afișarea următoarelor obiecte 3D:

con	icosaedru	teapot
cub	octaedru	tetraedru
dodecaedru	sfera	tor

Aceste obiecte pot fi afișate prin familii de curbe sau ca obiecte solide.

Exemplu: funcții de desenare cub, sferă și tor prin două familii de curbe și ca solide.

Desenare cub de latură size prin două familii de curbe

```
void glutWireCube(GLdouble size);
```

Desenare cub solid de latură size

```
void glutSolidCube(GLdouble size);
```

Desenare sferă prin două familii de curbe

```
void glutWireSphere(GLdouble radius, GLint slices, GLint stacks);
```

Desenare sferă solidă

```
void glutSolidSphere(GLdouble radius, GLint slices, GLint stacks);
```

Desenare tor prin două familii de curbe

```
void glutWireTorus(GLdouble innerRadius, GLdouble outerRadius,  
GLint nsides, GLint rings);
```

Desenare tor solid

```
void glutSolidTorus(GLdouble innerRadius, GLdouble outerRadius,  
GLint nsides, GLint rings);
```

```
void glutWireTeapot(GLdouble size);  
void glutSolidTeapot(GLdouble size);
```

Alte funcții sunt:

```
void glutWireIcosahedron(void);  
void glutSolidIcosahedron(void);  
  
void glutWireOctahedron(void);  
void glutSolidOctahedron(void);  
  
void glutWireTetrahedron(void);  
void glutSolidTetrahedron(void);  
  
void glutWireDodecahedron(GLdouble radius);  
void glutSolidDodecahedron(GLdouble radius);  
  
void glutWireCone( GLdouble radius, GLdouble height, GLint slices,GLint  
stacks);  
void glutSolidCone(GLdouble radius, GLdouble height, GLint slices,GLint  
stacks);
```

Toate aceste obiecte sunt desenate centrate în originea sistemului de coordonate real.

În momentul în care se fac modificări asupra unui obiect poate apărea efectul de “pâlpâire” a imaginii.

Pentru evitarea acestui efect se asociază ferestrei aplicației un buffer dublu. Astfel, într-un buffer se păstrează imaginea nemodificată (imaginea ce este afișată pe ecran), iar în cel de-al doilea se construiește imaginea modificată. În momentul în care s-a terminat construirea imaginii modificate se interschimbă buffer-ele (lucrul cu două buffere este asemănător lucrului cu mai multe pagini video în DOS). Pentru interschimbarea bufferelor se folosește funcția: **glutSwapBuffers :**

```
void glutSwapBuffers(void) ;
```

glutSwapBuffers

glutSwapBuffers swaps the buffers of the *current window* if double buffered.

Usage

```
void glutSwapBuffers(void);
```

Description

Performs a buffer swap on the *layer in use* for the *current window*. Specifically, glutSwapBuffers promotes the contents of the back buffer of the *layer in use* of the *current window* to become the contents of the front buffer. The contents of the back buffer then become undefined. The update typically takes place during the vertical retrace of the monitor, rather than immediately after glutSwapBuffers is called.

An implicit `glFlush` is done by `glutSwapBuffers` before it returns. Subsequent OpenGL commands can be issued immediately after calling `glutSwapBuffers`, but are not executed until the buffer exchange is completed.

If the *layer in use* is not double buffered, `glutSwapBuffers` has no effect.

glut Function Calls

Beginning Event Processing

- void **glutMainLoop** (void)

Initialization

- void **glutInit** (int *argcp, char **argv)
- void **glutInitDisplayMode** (unsigned int mode)
- void **glutInitWindowPosition** (int x, int y)
- void **glutInitWindowSize** (int width, int height)

Window Management

- int **glutCreateWindow** (char *name)
- int **glutCreateSubWindow** (int win, int x, int y, int width, int height)
- void **glutDestroyWindow** (int win)
- void **glutFullScreen** (void)
- int **glutGetWindow** (void)
- void **glutHideWindow** (void)
- void **glutIconifyWindow** (void)
- void **glutPopWindow** (void)
- void **glutPushWindow** (void)
- void **glutPositionWindow** (int x, int y)
- void **glutPostRedisplay** (void)
- void **glutReshapeWindow** (int width, int height)
- void **glutSetCursor** (int cursor)
- void **glutSetWindow** (int win)
- void **glutSetWindowTitle** (char *name)
- void **glutSetIconTitle** (char *name)
- void **glutShowWindow** (void)
- void **glutSwapBuffers** (void)

Overlay Management

- void **glutEstablishOverlay** (void)
- void **glutHideOverlay** (void)
- void **glutPostOverlayRedisplay** (void)
- void **glutRemoveOverlay** (void)
- void **glutShowOverlay** (void)
- void **glutUseLayer** (GLenum layer)

Menu Management

- void **glutAddMenuEntry** (char *name, int value)
- void **glutAddSubMenu** (char *name, int value)
- void **glutAttachMenu** (int button)
- void **glutChangeToMenuEntry** (int entry, char *name, int value)
- void **glutChangeToSubMenu** (int entry, char *name, int menu);
- int **glutCreateMenu** (void (*func)(int value))
- void **glutDestroyMenu** (int menu)
- void **glutDetachMenu** (int button)
- int **glutGetMenu** (void)
- void **glutRemoveMenuItem** (int entry)
- void **glutSetMenu** (int menu)

Callback Registration

- void **glutButtonBoxFunc** (void (*func)(int button, int state))
- void **glutDialsFunc** (void (*func)(int dial, int value))
- void **glutDisplayFunc** (void (*func)(void))
- void **glutEntryFunc** (void (*func)(int state))
- void **glutIdleFunc** (void (*func)(void))
- void **glutKeyboardFunc** (void (*func)(unsigned char key, int x, int y))
- void **glutMenuStatusFunc** (void (*func)(int status, int x, int y))
- void **glutMenuStateFunc** (void (*func)(int status))
- void **glutMotionFunc** (void (*func)(int x, int y))
- void **glutMouseFunc** (void (*func)(int button, int state, int x, int y))
- void **glutOverlayDisplayFunc** (void (*func)(void))
- void **glutPassiveMotionFunc** (void (*func)(int x, int y))
- void **glutReshapeFunc** (void (*func)(int width, int height))
- void **glutSpaceballButtonFunc** (void (*func)(int button, int state))
- void **glutSpaceballMotionFunc** (void (*func)(int x, int y, int z))
- void **glutSpaceballRotateFunc** (void (*func)(int x, int y, int z))
- void **glutSpecialFunc** (void (*func)(int key, int x, int y))
- void **glutTabletButtonFunc** (void (*func)(int button, int state, int x, int y))
- void **glutTabletMotionFunc** (void (*func)(int x, int y))
- void **glutTimerFunc** (unsigned int msec, void (*func)(int value), value)
- void **glutVisibilityFunc** (void (*func)(int state))

Color Index Colormap Management

- void **glutCopyColormap** (int win)
- GLfloat **glutGetColor** (int cell, int component)
- void **glutSetColor** (int cell, GLfloat red, GLfloat green, GLfloat blue)

State Retrieval

- int **glutDeviceGet** (GLenum info)
- int **glutExtensionSupported** (char *extension)
- int **glutGet** (GLenum state)
- int **glutGetModifiers** (void)
- int **glutLayerGet** (GLenum info)

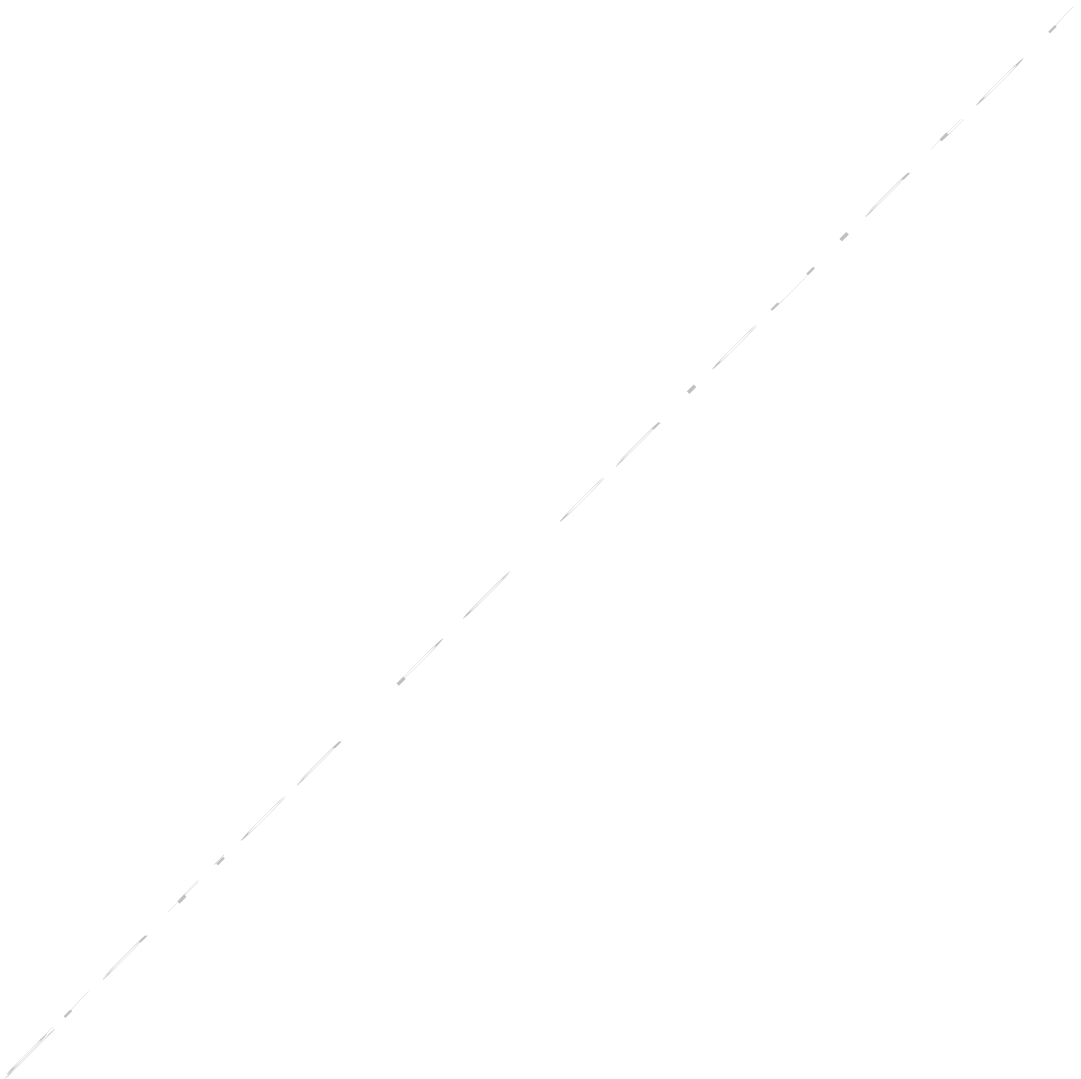
Font Rendering

- void **glutBitmapCharacter** (void *font, int character)
- int **glutBitmapWidth** (GLUTbitmapFont font, int character)
- void **glutStrokeCharacter** (void *font, int character)
- int **glutStrokeWidth** (GLUTstrokeFont font, int character)

Geometric Object Rendering

- void **glutSolidCone** (GLdouble base, GLdouble height, GLint slices, GLint stacks)
- void **glutSolidCube** (GLdouble size)
- void **glutSolidDodecahedron** (void)
- void **glutSolidIcosahedron** (void)
- void **glutSolidOctahedron** (void)
- void **glutSolidSphere** (GLdouble radius, GLint slices, GLint stacks)
- void **glutSolidTeapot** (GLdouble size)
- void **glutSolidTetrahedron** (void)
- void **glutSolidTorus** (GLdouble innerRadius, GLdouble outerRadius, GLint nsides, GLint rings)
- void **glutWireCone** (GLdouble base, GLdouble height, GLint slices, GLint stacks)
- void **glutWireCube** (GLdouble size)
- void **glutWireDodecahedron** (void)

- void **glutWireIcosahedron** (void)
- void **glutWireOctahedron** (void)
- void **glutWireSphere** (GLdouble radius, GLint, slices, GLint stacks)
- void **glutWireTeapot** (GLdouble size)
- void **glutWireTetrahedron** (void)
- void **glutWireTorus** (GLdouble innerRadius, GLdouble outerRadius, GLint nsides, GLint rings)



Exemplu utilizare obiecte 3d :

```
#include <GL/gl.h>
#include <GL/glut.h>
void init(void)
{
    glClearColor(1.0, 1.0, 1.0, 1.0);
    glEnable(GL_DEPTH_TEST);
}
void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glColor3f(0.0, 0.0, 1.0);
    glPushMatrix();
    glutWireTorus(2.0, 5.0, 20, 20);
    glPopMatrix();
    glutSwapBuffers();
    //glFlush();
}
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(400, 400);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("Exemplu");
    init();
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```

Exemplu complex. Utilizare meniuri pentru obiecte 3D predefinite in OpenGL /TeaPot

```
#include <gl/freeglut.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
static int window;
static int menu_id;
static int submenu_id;
static int value = 0;
void menu(int num) {
    if (num == 0) {
        glutDestroyWindow(window);
        exit(0);
    }
    else {
        value = num;
    }
    glutPostRedisplay();
}
void createMenu(void) {
    submenu_id = glutCreateMenu(menu);
    glutAddMenuEntry("Sphere", 2);
    glutAddMenuEntry("Torus", 4);
    glutAddMenuEntry("Teapot", 5);
    glutAddMenuEntry("Cube", 6);
    glutAddMenuEntry("Cone", 7);
}
```

```

glutAddMenuEntry("Tetrahedron", 8);
menu_id = glutCreateMenu(menu);
glutAddSubMenu("Draw", submenu_id);
glutAddMenuEntry("Clear", 1);
glutAddMenuEntry("Exit", 0);      glutAttachMenu(GLUT_RIGHT_BUTTON);
}
int GAngle = 0;
void display(void) {
    glClear(GL_COLOR_BUFFER_BIT);    if (value == 1) {
        return; //glutPostRedisplay();
    }
    else if (value == 2) {
        static float alpha = 20;
        glPushMatrix();
        glColor3d(1.0, 1.0, 0.0);
        glRotated(alpha, -1.0, 0.0, 0.0);
        glutWireSphere(0.5, 50, 50);
        glPopMatrix();
        alpha = alpha + 0.5;
    }
    else if (value == 4) {
        static float alpha = 20;

        glPushMatrix();
        glColor3d(0.0, 1.0, 1.0);
        glRotatef(alpha, 1.9, 0.6, 0);
        glutSolidTorus(0.3, 0.6, 100, 100);
        glPopMatrix();
        alpha = alpha + 0.5;
    }
    else if (value == 5) {
        static float alpha = 20;
        glPushMatrix();
        glColor3d(0.5, 1.0, 0.5);
        glRotatef(alpha, 1.9, 0.6, 0);
        glutSolidTeapot(0.5);
        glPopMatrix();
        alpha = alpha + 0.5;
    }
    else if (value == 6) {
        glColor3f(0.0, 0.5, 0.0);
        glLoadIdentity();
        glRotated(GAngle, 1, 1, 0);
        glutWireCube(0.5);
        GAngle = GAngle + 1;
    }
    else if (value == 7) {
        static float alpha = 20;
        glPushMatrix();
        glColor3d(1.0, 1.0, 1.0);
        glRotated(alpha, -1.0, 0.0, 0.0);
        glutWireCone(0.5, 1.0, 50, 50);
        glPopMatrix();
        alpha = alpha + 0.5;
    }
    else if (value == 8) {
        static float alpha = 20;
        glPushMatrix();
        glColor3d(0.0, 1.0, 0.0);
        glRotated(alpha, 0.0, 1.0, 0.0);
        glutWireTetrahedron();
        glPopMatrix();
        alpha = alpha + 0.5;
    }
    glFlush();
}

```

```

}
void Timer(int extra) {
    glutPostRedisplay();
    glutTimerFunc(40, Timer, 0);
}
int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGBA | GLUT_SINGLE);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(100, 100);
    window = glutCreateWindow("Aplicatie 2022");
    createMenu();
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glutDisplayFunc(display);
    glutTimerFunc(0, Timer, 0); glutMainLoop();

    return 0;
}

```

4.4 Recapitulare functii principale in OpenGL

glViewport(0, 0, w, h); // Stabilirea viewportului la dimensiunea ferestrei

```

void glViewport(
    GLint x,
    GLint y,
    GLsizei width,
    GLsizei height)

```

Parameters

x, y

Specify the lower left corner of the viewport rectangle, in pixels. The initial value is (0, 0).

width, height

Specify the width and height of the viewport. When a GL context is first attached to a window, *width* and *height* are set to the dimensions of that window.

glViewport specifies the affine transformation of *x* and *y* from normalized device coordinates to window coordinates. Let (*x_{nd}*, *y_{nd}*) be normalized device coordinates. Then the window coordinates (*x_w*, *y_w*) are computed as follows:

$$\begin{aligned}
 x_w &= (x_{nd} + 1) \cdot (width / 2) + x \\
 y_w &= (y_{nd} + 1) \cdot (height / 2) + y
 \end{aligned}$$

glLoadIdentity(); // Initializeaza sistemul de coordonate

glClearColor(0.0f, 0.0f, 1.0f, 1.0f); // Culoarea de stergere (albastru)

```

void glClearColor(    GLclampf    red,
                    GLclampf    green,

```

```
GLclampf    blue,  
GLclampf    alpha);
```

red, green, blue, alpha

Specify the red, green, blue, and alpha values used when the color buffers are cleared. The initial values are all 0.

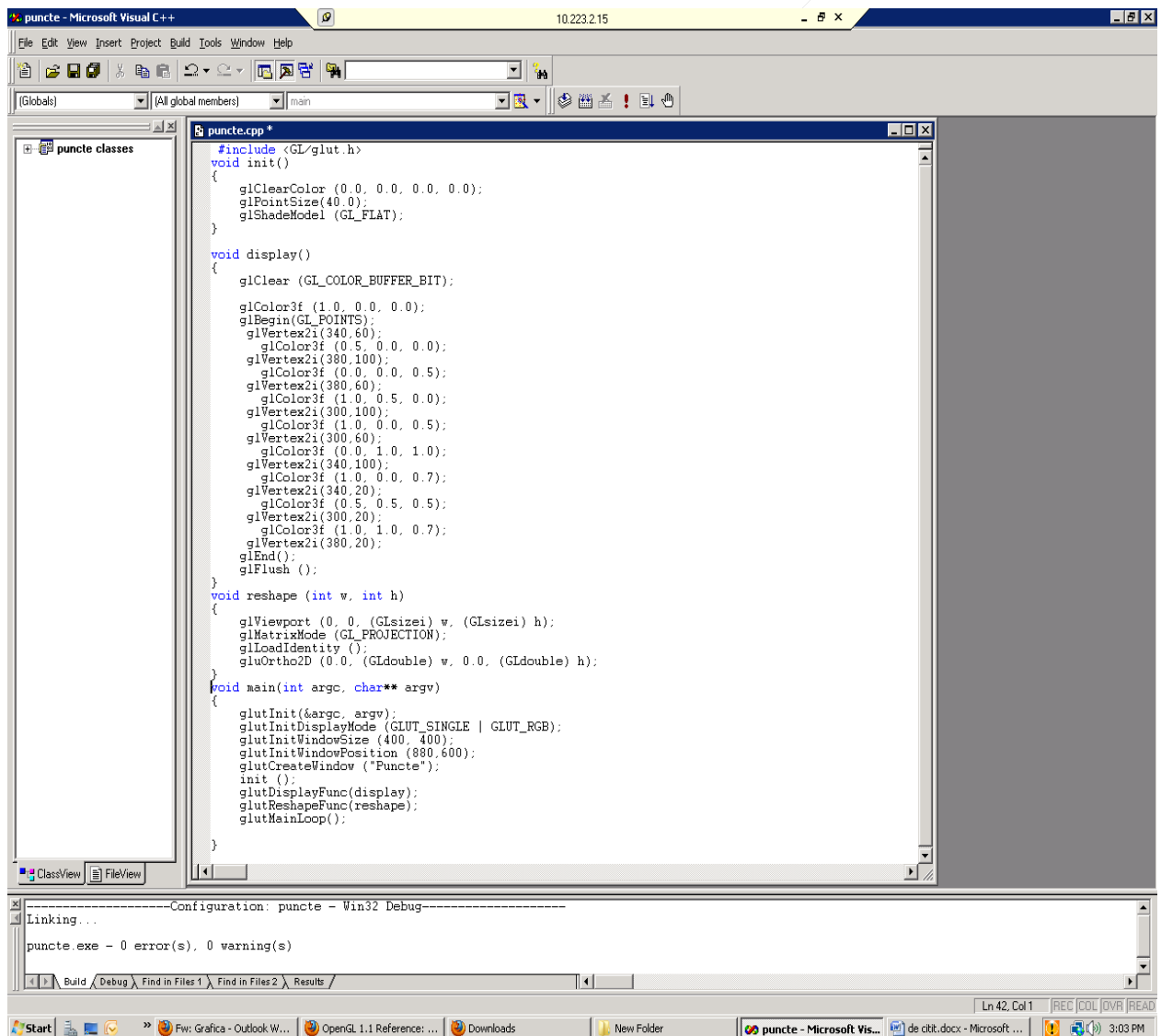
```
glClear(GL_COLOR_BUFFER_BIT); // Ștergerea ferestrei glColor3f(1.0f, 0.0f, 0.0f);
```

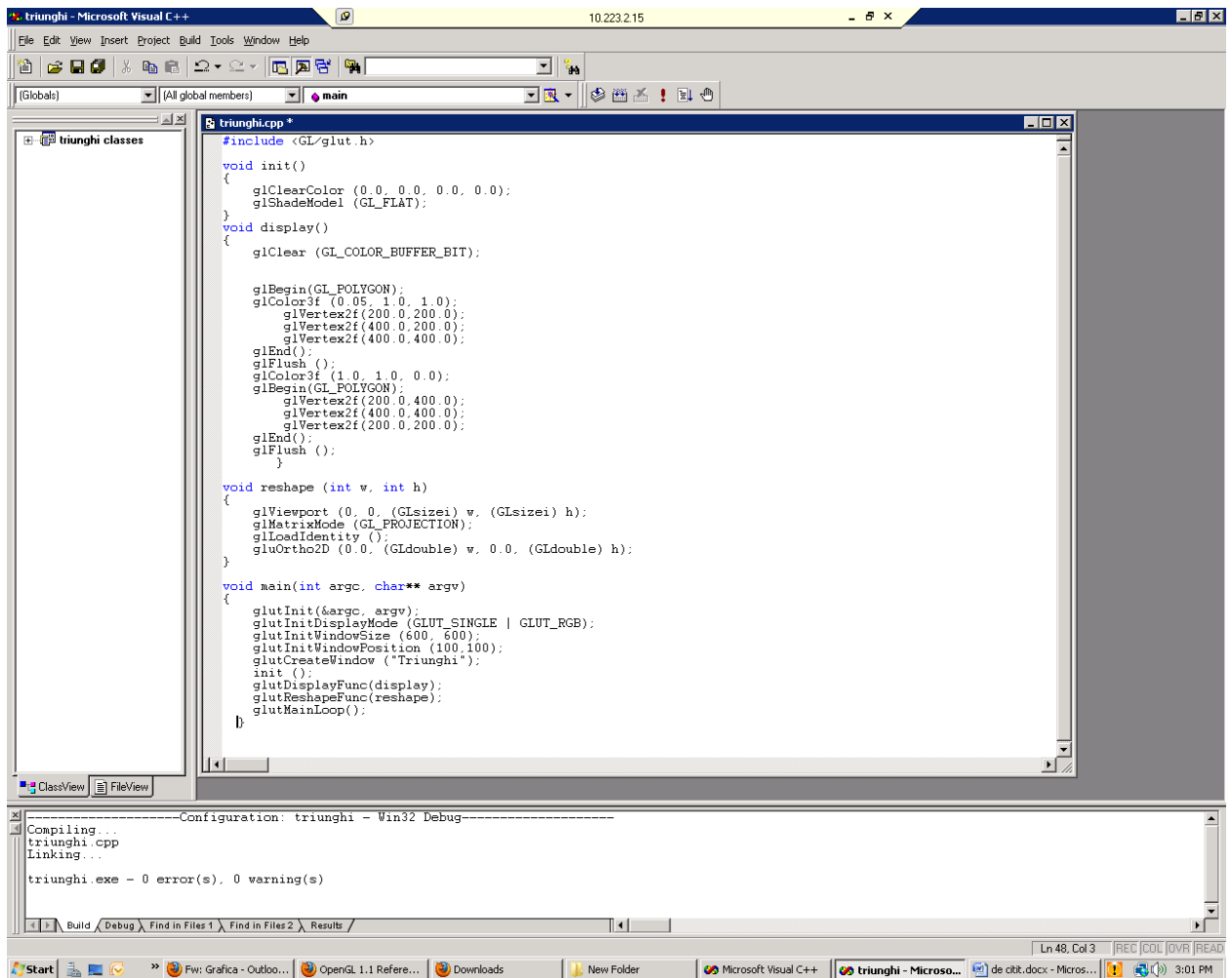
void **glPointSize**(GLfloat size);

Funcția setează lățimea în pixeli a punctelor ce vor fi afișate. Parametrul *size* reprezintă dimensiunea punctului exprimată în pixeli ecran. Ea trebuie să fie mai mare ca 0.0, iar valoarea sa implicită este 1.0.

void **glLineWidth**(GLfloat width);

Funcția setează lățimea în pixeli a liniilor ce vor fi afișate; *width* trebuie să fie mai mare ca 0.0, iar valoarea implicită este 1.0.





Utilizarea ferestrelor in OpenGL

void **glutInit**(int *argc, char **argv);

Funcția **glutInit()** inițializează variabilele interne ale pachetului de funcții GLUT și procesează argumentele din linia de comandă. Ea trebuie sa fie apelată înaintea oricărei alte comenzi GLUT.

void **glutInitDisplayMode**(unsigned int mode);

Folosirea modelului RGBA (culoarea se specifică prin componentele sale roșu, verde, albastru și transparența sau opacitatea) sau a modelului de culoare bazat pe indecși de culoare. În general se recomandă folosirea modelului RGBA.

Folosirea unei ferestre cu un singur buffer sau cu buffer dublu, pentru realizarea animației. Folosirea bufferului de adâncime pentru algoritmul z-buffer.

void **glutInitWindowPosition**(int x, int y) ;

Funcția **glutInitWindowPosition** specifică colțul stânga sus al ferestrei în coordonate relative la colțul stânga sus al ecranului.

void **glutInitWindowSize**(int width, int height) ;

Funcția **glutInitWindowSize** specifică dimensiunea în pixeli a ferestrei : lățimea (width) și înălțimea (height).

int **glutCreateWindow**(char *string) ;

Funcția **glutCreateWindow** creează o fereastră cu un context OpenGL. Ea întoarce un

identificator unic pentru fereastra nou creată. Fereastra nu va fi afișată înainte de apelarea funcției **glutMainLoop**. Valoarea întoarsă de funcție reprezintă identificatorul ferestrei, care este unic.

void glutKeyboardFunc(void (*func)(unsigned char key, int x, int y)) :

Parametrul *func* reprezintă funcția callback care va fi apelată la apăsarea / eliberarea unei taste care generează un caracter ASCII. Parametrul *key* al funcției callback reprezintă valoarea ASCII. Parametrii *x* și *y* indică poziția mouse-ului la apăsarea tastei (poziție specificată în coordonate fereastră).

void glutMouseFunc(void (*func)(int button, int state, int x, int y)) :

Parametrul *func* specifică funcția callback care va fi apelată la apăsarea / eliberarea unui buton al mouseului. Parametrul *button* al funcției callback poate avea una din următoarele valori: GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON sau GLUT_RIGHT_BUTTON. Parametrul *state* poate fi GLUT_UP sau GLUT_DOWN după cum butonul mouse-ului a fost apăsător sau eliberat. Parametrii *x* și *y* reprezintă poziția mouse-ului la apariția evenimentului. Valorile *x* și *y* sunt relative la colțul stânga sus al ferestrei aplicației.

void glVertex{234} {sifd} [v] (TYPE coords) :

Funcția poate avea:

- 2 parametri – vârful va fi specificat în 2D prin coordonatele sale (*x*, *y*)
- 3 parametri – vârful va fi specificat în 3D prin coordonatele sale (*x*, *y*, *z*)
- 4 parametri – vârful va fi specificat în 3D prin coordonatele sale omogene (*x*, *y*, *z*, *w*)

Și în acest caz *TYPE* reprezintă tipul coordonatelor vârfului (*s* - short, *i* - int, *f* - float, *d* - double). Apelul funcției **glVertex** trebuie să fie făcut între perechea de comenzi **glBegin** și **glEnd**.

Funcția de afișare callback

```
void glutDisplayFunc(void (*func)(void)) ;
```

Parametrul funcției **glutDisplayFunc** specifică funcția callback a aplicației care va fi apelată de GLUT pentru afișarea conținutului inițial al ferestrei aplicației precum și ori de câte ori trebuie refăcut conținutul ferestrei ca urmare a cererii explicite a aplicației, prin apelul funcției **glutPostRedisplay()**.

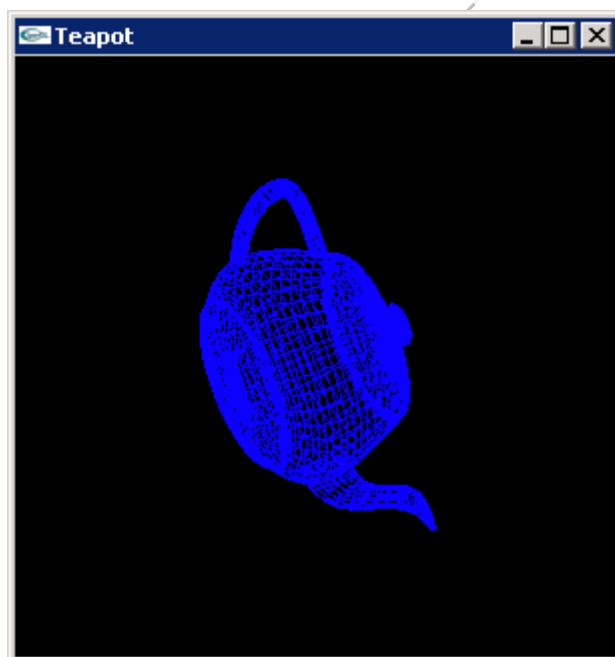
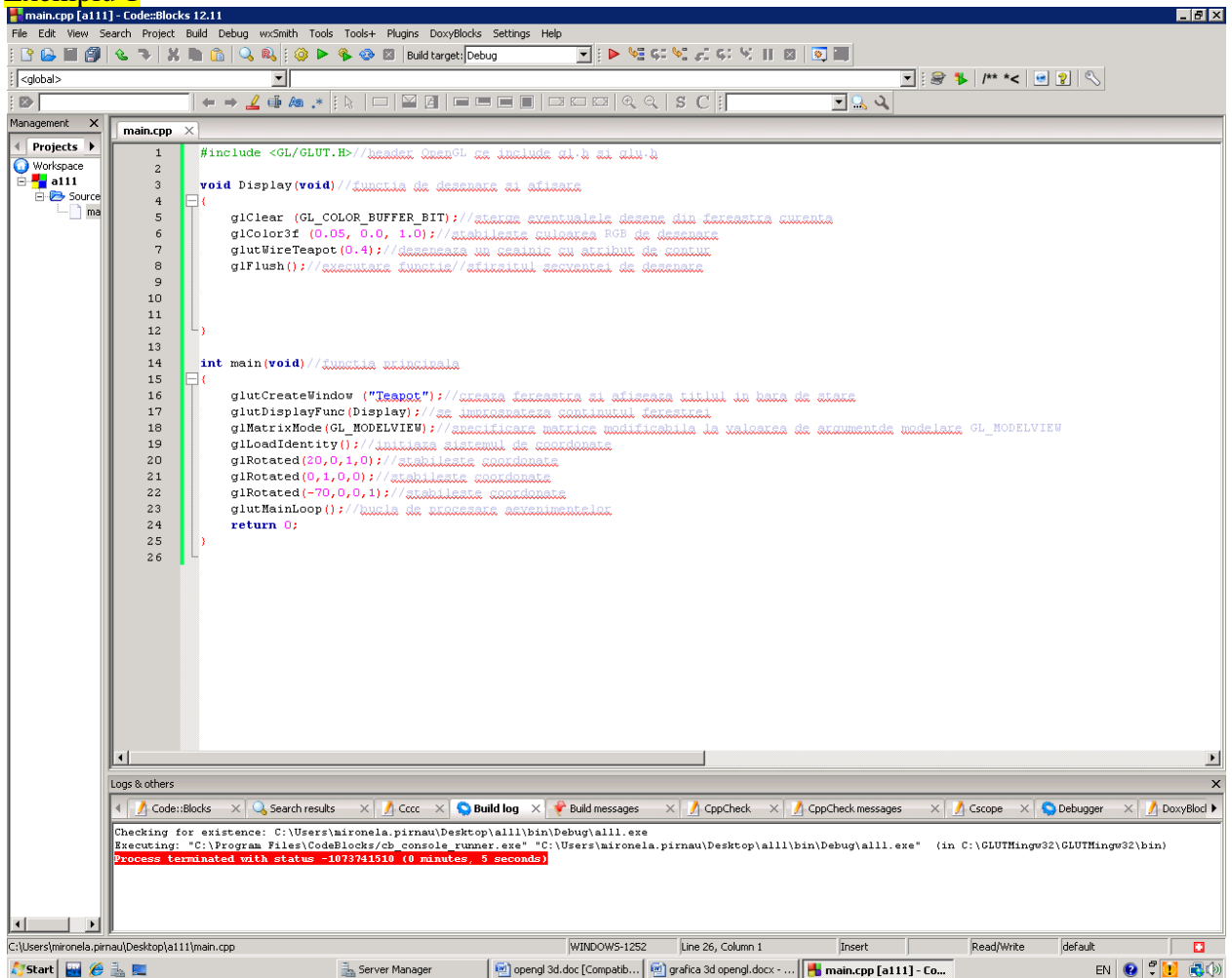
Bucula de execuție a aplicației

```
void glutMainLoop(void) ;
```

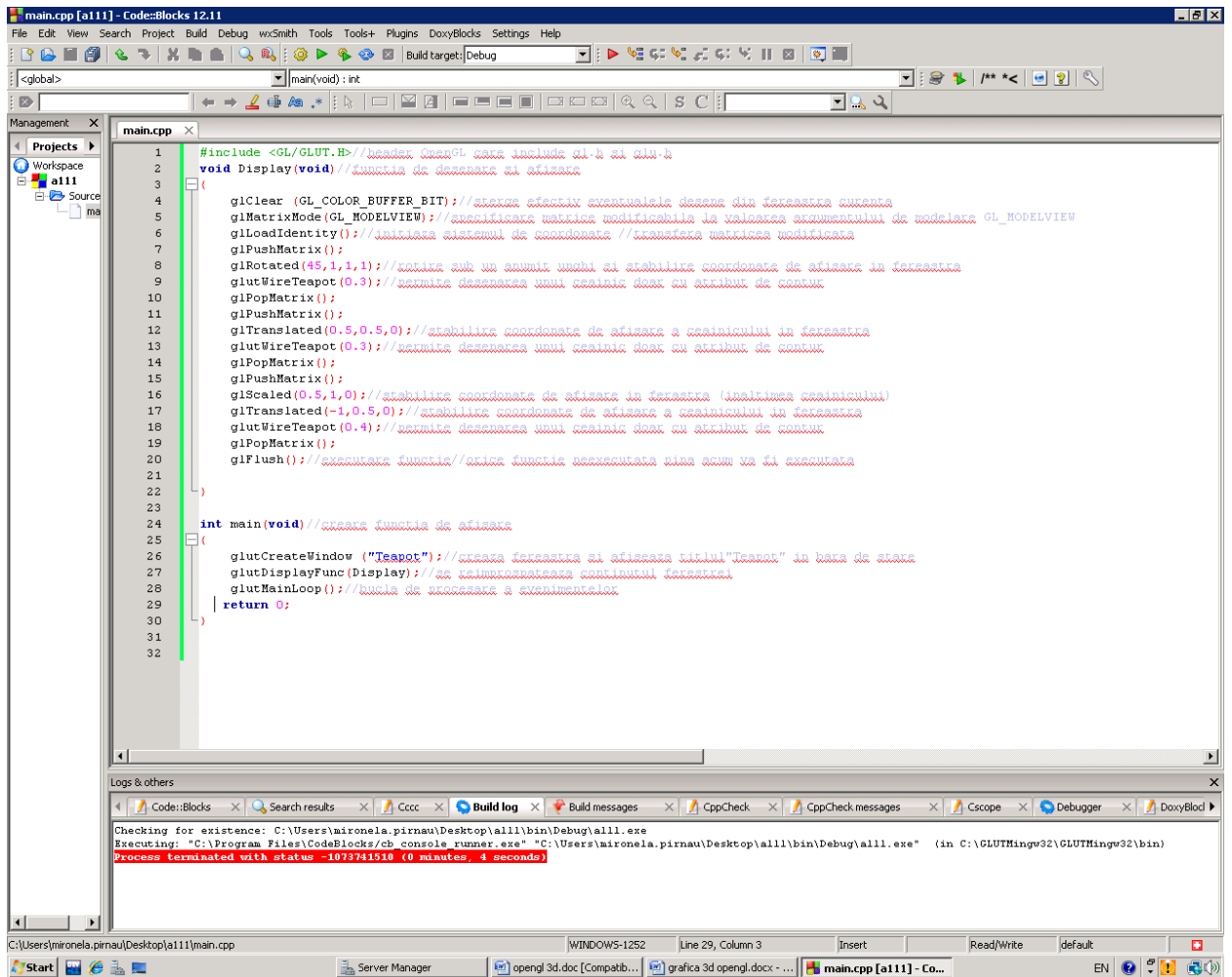
Aceasta este ultima funcție care trebuie apelată în funcția **main** a aplicației. Ea conține bucla de execuție (infinită) a aplicației în care aplicația așteaptă evenimente. Orice eveniment este tratat prin rutina callback specificată anterior în funcția **main**, prin apelul funcției GLUT specifice tipului de eveniment.

Aplicatii propuse

Exemplu 1



Exemplu 2.

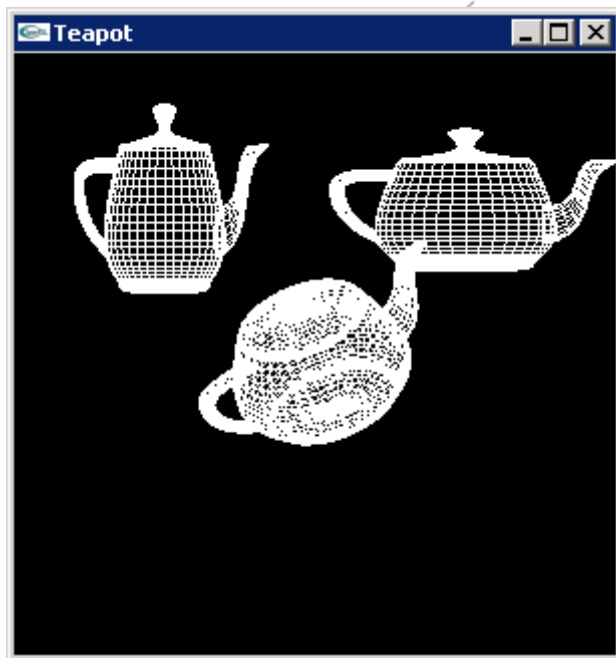


The screenshot shows the Code::Blocks IDE interface. The main window displays the `main.cpp` file with the following code:

```
1 #include <GL/GLUT.H> //include OpenGL care include GL si GLU
2 void Display(void) //functie de desenaire si afisare
3 {
4     glClear (GL_COLOR_BUFFER_BIT); //sterge efectiv ecranul din fereastra curenta
5     glMatrixMode(GL_MODELVIEW); //specificare matrice modificabila la valoarea actualizarii de modelare GL_MODELVIEW
6     glLoadIdentity(); //initializa sistemul de coordonate //transfata matricea modificata
7     glPushMatrix();
8     glRotated(45,1,1,1); //rotire sub un anumit unghi si stabilirea coordonate de afisare in fereastra
9     glutWireTeapot(0.3); //permite desenairea unui ceainic doar cu atribut de contur
10    glPopMatrix();
11    glPushMatrix();
12    glTranslated(0.5,0.5,0); //stabilirea coordonate de afisare a ceainicului in fereastra
13    glutWireTeapot(0.3); //permite desenairea unui ceainic doar cu atribut de contur
14    glPopMatrix();
15    glPushMatrix();
16    glScaled(0.5,1,0); //stabilirea coordonate de afisare in fereastra (inaltimea ceainicului)
17    glTranslated(-1,0.5,0); //stabilirea coordonate de afisare a ceainicului in fereastra
18    glutWireTeapot(0.4); //permite desenairea unui ceainic doar cu atribut de contur
19    glPopMatrix();
20    glFlush(); //executare functie/orice functie reexecutata pana acum va fi executata
21 }
22
23
24 int main(void) //creaza functie de afisare
25 {
26     glutCreateWindow ("Teapot"); //creaza fereastra si afisarea titlului "Teapot" in bara de stare
27     glutDisplayFunc(Display); //sa se actualizeze continutul ferestrei
28     glutMainLoop(); //functie de intrerupere a executiei
29     return 0;
30 }
31
32
```

The bottom panel shows the "Logs & others" window with the following output:

```
Checking for existence: C:\Users\mironela.pirna\Desktop\all1\bin\Debug\all1.exe
Executing: "C:\Program Files\CodeBlocks\cb_console_runner.exe" "C:\Users\mironela.pirna\Desktop\all1\bin\Debug\all1.exe" (in C:\GLUTMingw32\GLUTMingw32\bin)
Process terminated with status -1073741510 (0 minutes, 4 seconds)
```

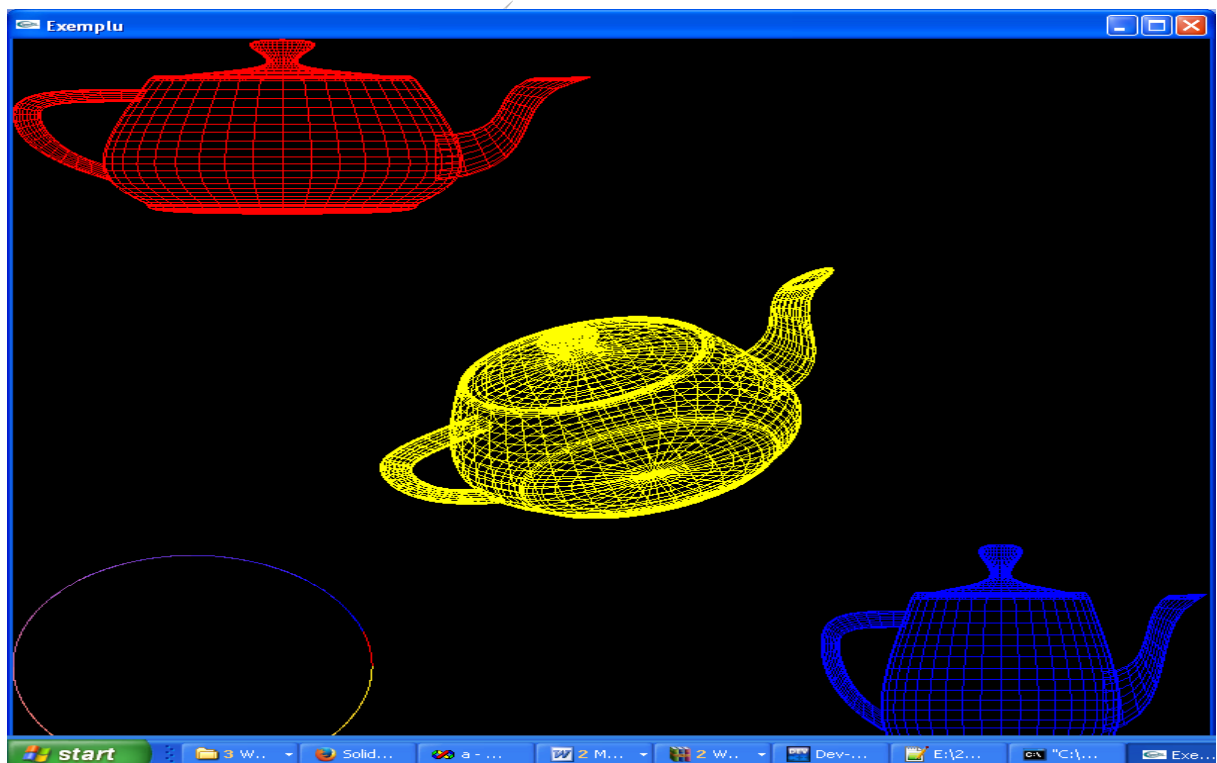


Exemplu3. Sa se realizeze aplicatia:

```

*main.cpp [a111] - Code::Blocks 12.11
File Edit View Search Project Build Debug wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
Build target: Debug
<global>
main(void): int
Management
*main.cpp
1 #include <stdio.h>
2 #include <math.h>
3 #include <gl/glut.h>
4 void Display(void)
5 {
6     glClear(GL_COLOR_BUFFER_BIT);
7     glMatrixMode(GL_MODELVIEW);
8     glLoadIdentity();
9     glPushMatrix(); glColor3f(1,1,0);
10    glRotated(45,1,1,1); //rotate the teapot
11    glutWireTeapot(0.3); //create the teapot
12    glPopMatrix();
13    glPushMatrix();
14    glColor3f(1,0,0);
15    glTranslated(-0.55,0.75,0); //
16    glutWireTeapot(0.3); //
17    glPopMatrix(); glPushMatrix();
18    glColor3f(0,0,1);
19    glScaled(0.5,1,0);
20    glTranslated(1.3,-0.7,1); // glutWireTeapot(0.4);
21    glPopMatrix(); glFlush();
22    glColor3f(1.0,0.0,0.0); glTranslated(-0.7,-0.7,1);
23    glScaled(0.3,0.3,0.3); glBegin(GL_POINTS);float r=0.01,g=0.01,b=1;
24    for(int i=0,k=1;i<1000;i++)
25    {
26        glVertex3f(cos(2*3.14159*i/1000.0),sin(2*3.14159*i/1000.0),0);
27        if(i>k*50)
28        {
29            k++;
30            glColor3f(r,g,b);
31            r+=0.1;g+=0.05;b-=0.05;
32        }
33    } glEnd(); glFlush();
34    int main(void)
35    {
36        glutInitWindowSize(800,800);
37        glutCreateWindow("Exemplu"); glutDisplayFunc(Display);
38        glutMainLoop();return 0;
39    }
Logs & others
Code::Blocks Search results Cccc Build log Build messages CppCheck CppCheck messages Cscope Debugger DoxyBlod
C:\Users\mironela.pirnaul\Desktop[a111]main.cpp WINDOWS-1252 Line 38, Column 29 Insert Modified Read/Write default
Start Server Manager *main.cpp [a111] 2 Windows Explo... opengl 3d.doc [Co... grafica 3d opengl.... GRAFICA PE CALC... EN

```



Exemplu 4:

Meniu atasat mouse-ului:

```
int meniu_1,meniu_2,meniu_3,meniu_main;
```

```
//meniu principal
```

```
void meniu_principal(int key)
```

```
{
    if(key == 0)
    {
        exit(0);
    }
}
```

```
//Meniu 1
```

```
void callback_1(int key)
```

```
{
    switch(key)
    {
        case 0:
            printf("Selectie\n");
            break;
        case 1:
            printf("Editare\n");
            break;
    }
}
```

```
//Meniu 2
```

```
void callback_2(int key)
```

```
{
    switch(key)
    {
        case 0:
            printf("Translatie\n");
            break;
        case 1:
            printf("Scalare\n");
            break;
        case 2:
            printf("Rotatie\n");
            break;
    }
}
```

```
//Meniu 3
```

```
void callback_3(int key)
```

```
{
    switch(key)
    {
        case 0:
            printf("OX\n");
            break;
        case 1:
            printf("OY\n");
            break;
        case 2:
            printf("OZ\n");
            break;
    }
}
```

```
}
```

```
int main(void)
```

```
{
    //Meniuri
    meniu_1=glutCreateMenu(callback_1);
    glutAddMenuEntry("Selectie",0);
    glutAddMenuEntry("Editare",1);
    meniu_2=glutCreateMenu(callback_2);
```

```
    glutAddMenuEntry("Translatie",0);
    glutAddMenuEntry("Scalare",1);
    glutAddMenuEntry("Rotatie",2);
    meniu_3=glutCreateMenu(callback_3);
    glutAddMenuEntry("OX",0);
    glutAddMenuEntry("OY",1);
    glutAddMenuEntry("OZ",2);
    meniu_main=glutCreateMenu(meniu_principal);
    glutAddSubMenu("Stare Program",meniu_1);
    glutAddSubMenu("Operatie",meniu_2);
    glutAddSubMenu("Axa",meniu_3);
    glutAddMenuEntry("Exit",0);
    glutAttachMenu(GLUT_RIGHT_BUTTON);
    glutMainLoop();
    return 0;
}
```

4.5 Probleme rezolvate

Tema de lucru: Toate aplicatiile urmatoare se vor realiza si vor fi utilizate/apelate intr-un program principal, avand la baza meniuri.

Aplicatia 1

```
#include <stdio.h>
#include <math.h>
#include <GL/GLUT.H>

void Display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

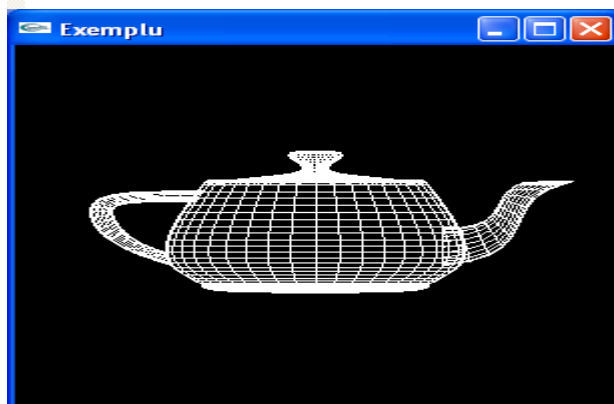
    glutWireTeapot(0.5);

    glFlush();
}

int main(void)
{
    glutCreateWindow("Exemplu");
    glutDisplayFunc(Display);

    glMatrixMode(GL_MODELVIEW);

    glutMainLoop();
    return 0;
}
```



Aplicatia 2

```

#include <stdio.h>
#include <math.h>
#include <GL/GLUT.H>

void Display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    glutWireTeapot(0.5);

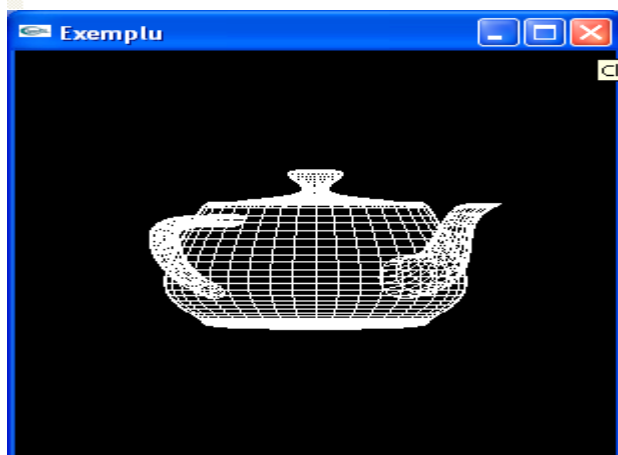
    glFlush();
}

int main(void)
{
    glutCreateWindow("Exemplu");
    glutDisplayFunc(Display);

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glRotated(45,0,1,0);

    glutMainLoop();
    return 0;
}

```



Aplicatia 3

```

#include <stdio.h>
#include <math.h>
#include <GL/GLUT.H>

void Display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    glutWireTeapot(0.5);

    glFlush();
}

int main(void)
{
    glutCreateWindow("Exemplu");
    glutDisplayFunc(Display);

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glRotated(45,1,1,1);

    glutMainLoop();
    return 0;
}

```

Aplicatia 4

```
#include <stdio.h>
#include <math.h>
#include <GL/GLUT.H>

void Display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    glPushMatrix();
    glRotated(45,1,1,1);
    glutWireTeapot(0.3);
    glPopMatrix();

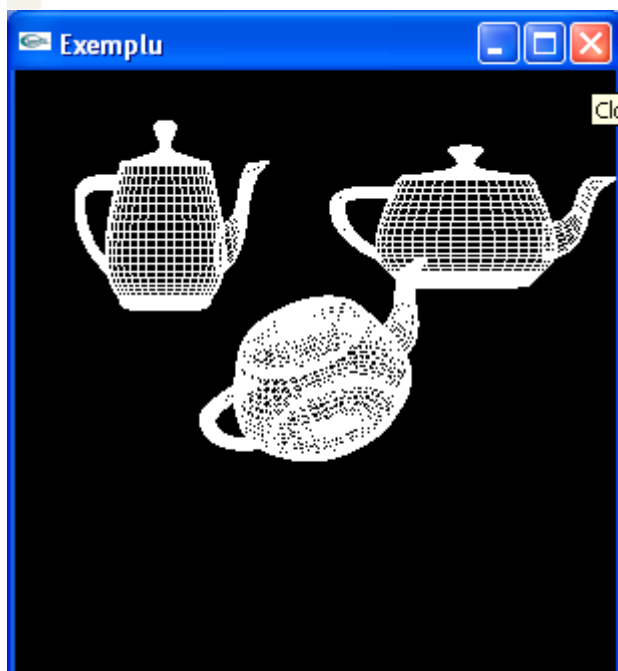
    glPushMatrix();
    glTranslated(0.5,0.5,0);
    glutWireTeapot(0.3);
    glPopMatrix();

    glPushMatrix();
    glScaled(0.5,1,0);
    glTranslated(-1,0.5,0);
    glutWireTeapot(0.4);
    glPopMatrix();

    glFlush();
}

int main(void)
{
    glutCreateWindow("Exemplu");
    glutDisplayFunc(Display);

    glutMainLoop();
    return 0;
}
```



Aplicatia 5

```
#include <stdio.h>
#include <math.h>
#include <GL/GLUT.H>

void Display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    glutWireTeapot(0.5);

    glFlush();
}

int main(void)
{
    glutCreateWindow("Exemplu");
    glutDisplayFunc(Display);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(-2,2,-2,2,-2,2);

    glutMainLoop();
    return 0;
}
```

Aplicatia 6

```
#include <stdio.h>
#include <math.h>
#include <GL/GLUT.H>

void Display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glutWireTeapot(0.9);
    glFlush();
}

int main(void)
{
    glutCreateWindow("Exemplu");
    glutDisplayFunc(Display);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glFrustum(-1,1,-1,1,1,3);
    glTranslated(0,0,-2);

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glRotated(45,0,1,0);

    glutMainLoop();
    return 0;
}
```

Aplicatia 7

```

#include <stdio.h>
#include <windows.h>
#include <GL/GLUT.H>

void Display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    glutWireTeapot(0.5);

    glFlush();
}

void Timer(int extra)
{
    Beep(1000,100);
    glutTimerFunc(1000,Timer,0);
}

int main(void)
{
    glutCreateWindow("Exemplu");
    glutDisplayFunc(Display);
    glutTimerFunc(1000,Timer,0);
    glutMainLoop();
    return 0;
}

```

Aplicatia 8

```

#include <stdio.h>
#include <GL/GLUT.H>

int GAngle=0;

void Display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    glLoadIdentity();
    glRotated(GAngle,0,1,0);

    glutWireTeapot(0.5);

    GAngle = GAngle +1;

    glFlush();
}

void Timer(int extra)
{
    glutPostRedisplay();
    glutTimerFunc(30,Timer,0);
}

int main(void)
{
    glutCreateWindow("Exemplu");
    glutDisplayFunc(Display);
    glutTimerFunc(0,Timer,0);

    glutMainLoop();
    return 0;
}

```

Aplicatia 9

```

#include <stdio.h>
#include <GL/GLUT.H>
int GAngle=0;
void Display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glLoadIdentity();
    glRotated(GAngle,0,1,0);
    glutWireTeapot(0.9);
    GAngle = GAngle +1;
    glFlush();
    glutSwapBuffers();
}
void Timer(int extra)
{
    glutPostRedisplay();
    glutTimerFunc(30,Timer,0);
}
int main(void)
{
    glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB);
    glutCreateWindow("Exemplu");
    glutDisplayFunc(Display);
    glutTimerFunc(0,Timer,0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glFrustum(-1,1,-1,1,1,3);
    glTranslated(0,0,-2);
    glMatrixMode(GL_MODELVIEW);

    glutMainLoop();
    return 0;
}

```

Aplicatia 10

```

#include <stdio.h>
#include <GL/GLUT.H>
int GAngle=0;
void DrawPlanetMoon(float radius,int angle)
{
    glPushMatrix();
    glRotated(angle,0,1,0);
    glPushMatrix();
    glTranslated(radius,0,0);
    glutWireSphere(radius,10,10);
    glPopMatrix();
    glPushMatrix();
    glTranslated(-2*radius,0,0);
    glutWireSphere(radius/2,10,10);
    glPopMatrix();
    glPopMatrix();
}
void Display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glLoadIdentity();
    DrawPlanetMoon(0.4,GAngle++);
    glFlush(); glutSwapBuffers();
}
void Timer(int extra)
{
    glutPostRedisplay(); glutTimerFunc(30,Timer,0);
}
int main(void)
{
    glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB);
    glutCreateWindow("Exemplu"); glutDisplayFunc(Display);
    glutTimerFunc(0,Timer,0);
    glMatrixMode(GL_PROJECTION); glLoadIdentity();
    glFrustum(-1,1,-1,1,1,3); glTranslated(0,0,-2);
    glMatrixMode(GL_MODELVIEW);
    glutMainLoop();
    return 0; }

```

Aplicatia 11

```

#include <stdio.h>
#include <GL/GLUT.H>
int GAngle=0;
void Display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glLoadIdentity();
    glRotated(GAngle,0,1,0);
    glBegin(GL_TRIANGLES);
        glVertex3f(-1,0,0);
        glVertex3f(1,0,0);
        glVertex3f(0,1,0);
    glEnd();
    GAngle = GAngle + 1;
    glFlush();
    glutSwapBuffers();
}
void Timer(int extra)
{
    glutPostRedisplay();
    glutTimerFunc(30,Timer,0);
}
int main(void)
{
    glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB);
    glutCreateWindow("Exemplu");
    glutDisplayFunc(Display);
    glutTimerFunc(0,Timer,0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glFrustum(-1,1,-1,1,1,3);
    glTranslated(0,0,-2);
    glMatrixMode(GL_MODELVIEW);
    glutMainLoop();
    return 0;
}

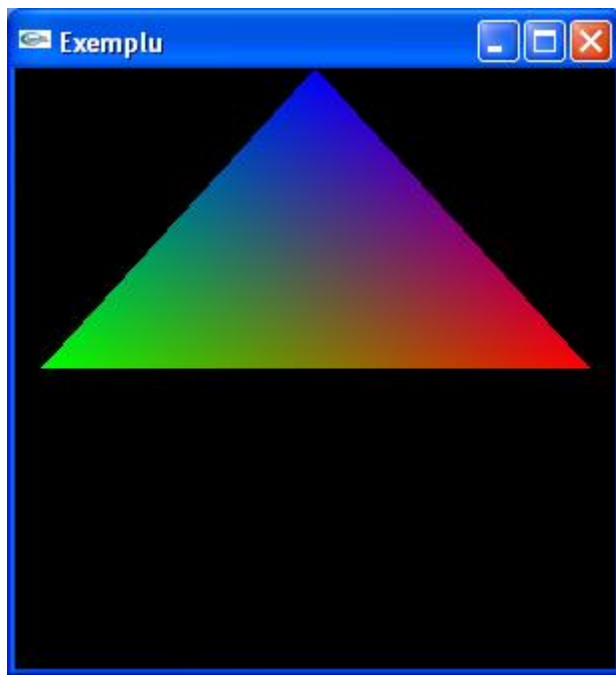
```

Aplicatia 12

```

#include <stdio.h>
#include <GL/GLUT.H>
int GAngle=0;
void Display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glLoadIdentity();
    glRotated(GAngle,0,1,0);
    glBegin(GL_TRIANGLES);
        glColor3f(1,0,0);
        glVertex3f(-1,0,0);
        glColor3f(0,1,0);
        glVertex3f(1,0,0);
        glColor3f(0,0,1);
        glVertex3f(0,1,0);
    glEnd();
    GAngle = GAngle + 1;
    glFlush();
    glutSwapBuffers();
}
void Timer(int extra)
{
    glutPostRedisplay();
    glutTimerFunc(30,Timer,0);
}
int main(void)
{
    glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB);
    glutCreateWindow("Exemplu");
    glutDisplayFunc(Display);
    glutTimerFunc(0,Timer,0);
    glShadeModel(GL_SMOOTH);
    glutMainLoop();
    return 0;
}

```



Aplicatia 13

```
#include <stdio.h>
#include <GL/GLUT.H>
int GAngle=0;
void Display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glLoadIdentity();
    glRotated(GAngle,0,1,0);

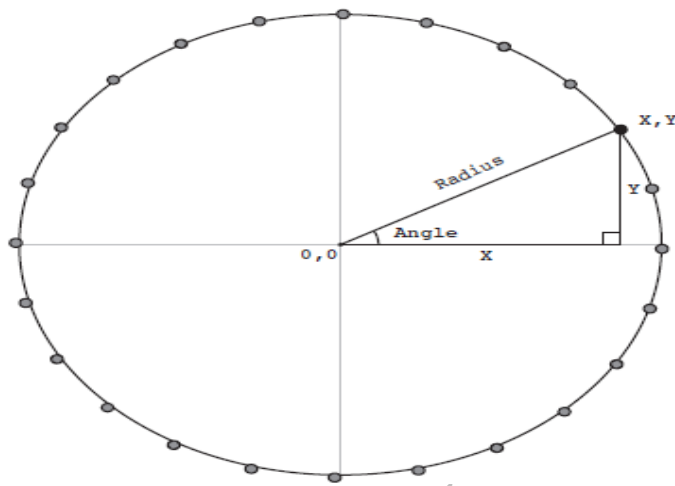
    glBegin(GL_TRIANGLES);
    glVertex3f(1,0,0);
    glVertex3f(0,1,0);
    glVertex3f(-1,0,0);
    glEnd();
    GAngle = GAngle + 1;
    glFlush();
    glutSwapBuffers();
}
void Timer(int extra)
{
    glutPostRedisplay();
    glutTimerFunc(30,Timer,0);
}

int main(void)
{
    glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB);
    glutCreateWindow("Exemplu");
    glutDisplayFunc(Display);
    glutTimerFunc(0,Timer,0);
    glEnable(GL_LIGHTING);
    GLfloat ambient[]={1,1,1,1};
    glLightModelfv(GL_LIGHT_MODEL_AMBIENT,ambient);
    GLfloat material[]={0.5,0.5,0.5,1};
    glMaterialfv(GL_FRONT,GL_AMBIENT_AND_DIFFUSE,material);
    glEnable(GL_CULL_FACE);
    glutMainLoop();
    return 0;
}
```

Drawing Circles with OpenGL

```
void drawCircle( float Radius, int numPoints )  
{  
    glBegin( GL_LINE_STRIP );  
    for( int i=0; i<numPoints; i++ )  
    {  
        float Angle = i * (2.0*PI/numPoints); // use 360 instead of 2.0*PI if  
        float X = cos( Angle )*Radius; // you use d_cos and d_sin  
        float Y = sin( Angle )*Radius;  
        glVertex2f( X, Y );  
    }  
    glEnd();  
}
```

The Angle variable in this code is in radians, since the built-in cos and sin functions take radians as inputs. To use degrees instead, implement helper functions d_cos and d_sin that convert the angle to radians from degrees, and then return the cosine or sine.



```
X = cos(Angle)*Radius;  
Y = sin(Angle)*Radius;
```

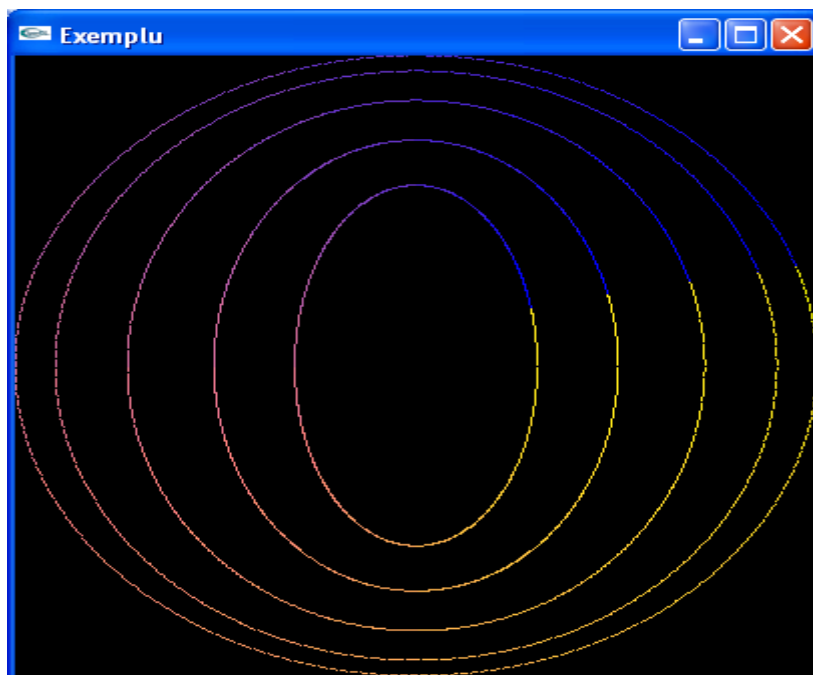
```
Pythagorean Theorem:  
Radius = sqrt(X*X + Y*Y);
```

This circle is shown with 24 points.

Aplicatia 14

```
#include <stdio.h>
#include <math.h>
#include <gl/glut.h>
void Display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0,1.0,0.0);
    float pas_x=1,pas_y=1;
    int i=0, ii=1; int k=1;
    for(ii=1;ii<=5;ii++)
    {
        glScaled(pas_x,pas_y,0.0);pas_x-=.1;pas_y-=.05;
glBegin(GL_POINTS);float r=0.01,g=0.01,b=1;
        for(i=0,k=1;i<1000;i++)
        {
            glVertex3f(cos(2*3.14*i/1000.0),sin(2*3.14*i/1000.0),0);
            if(i>k*50)
            {
                k++;
                glColor3f(r,g,b);
                r+=0.1;g+=0.05;b-=0.05;
            }
        }
        glEnd();
        glFlush();
    }

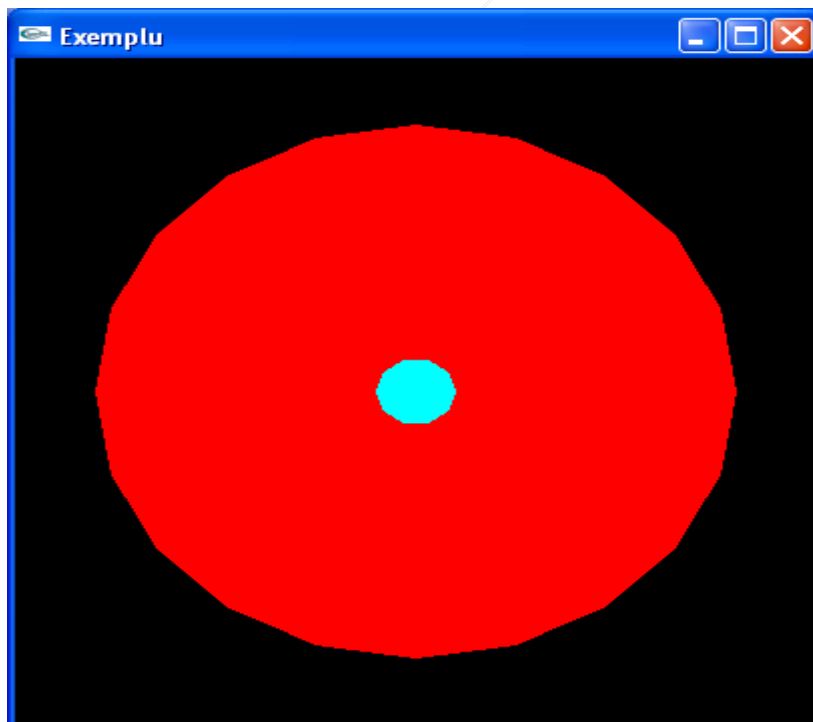
    //glScaled(0.75,0.75,0.0);
}
void main(void)
{
    glutInitWindowSize(400,400);
    glutCreateWindow("Exemplu");
    glutDisplayFunc(Display);
    glutMainLoop();
}
```



Aplicatia 15

```
#include<stdio.h>
#include<math.h>
#include<GL/GLUT.h>
void Display() {
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0f, 0.0f, 0.0f);
    int i;
    int sectiuni = 20;
    GLfloat raza1 = 0.8f;
    GLfloat doiPi = 2.0f * 3.14159f;

    glBegin(GL_TRIANGLE_FAN);
        glVertex2f(0.0, 0.0); // originea
        for(i = 0; i <= sectiuni; i++) { //numar sectiuni
            glVertex2f(raza1 * cos(i * doiPi / sectiuni),
                raza1 * sin(i * doiPi / sectiuni));
        }
    glEnd();glFlush();
    glColor3f(0.0f, 1.0f, 1.0f);
    GLfloat rad = 0.1; //raza
    GLfloat Pi = 2*3.14;
    int sectiuni_i = 10;
    glBegin(GL_TRIANGLE_FAN);
        glVertex2f(0.0, 0.0); // originea
        for(i = 0; i <= sectiuni_i; i++) {
            glVertex2f(rad * cos(i * Pi / sectiuni_i), rad * sin(i * Pi / sectiuni_i));
        }
    glEnd();glFlush();
}
void main(void)
{
    glutInitWindowSize(400,400);
    glutCreateWindow("Exemplu");
    glutDisplayFunc(Display);
    glutMainLoop();
}
```



Aplicatia 16

```
#include<stdio.h>
#include<math.h>
#include<GL/GLUT.h>
void Display() {
    // cerc prin puncte
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_POINTS);
    for(int i=0;i<1000;++i)
    {
        glVertex2f(cos(2*3.14159*i/1000.0),sin(2*3.14159*i/1000.0));
    }
    glEnd();
    glFlush();

    glColor3f(0.0f, 1.0f, 0.0f); //
    glBegin(GL_TRIANGLE_STRIP); // deseneaza triangle strips
    glVertex2f(0.0f, 0.75f); // top
    glVertex2f(-0.5f, 0.25f); // left
    glVertex2f(0.5f, 0.25f); // right f
    glVertex2f(-0.5f, -0.5f); // bottom left corner
    glVertex2f(0.5f, -0.5f); //bottom right corner
    glColor3f(1.0f, 1.0f, 0.0f); //
    glBegin(GL_QUADS);

        glVertex2f(-0.25f, 0.25f); // vertex 1
        glVertex2f(-0.5f, -0.25f); // vertex 2
        glVertex2f(0.5f, -0.25f); // vertex 3
        glVertex2f(0.25f, 0.25f); // vertex 4
    glEnd();glEnd(); glFlush(); }

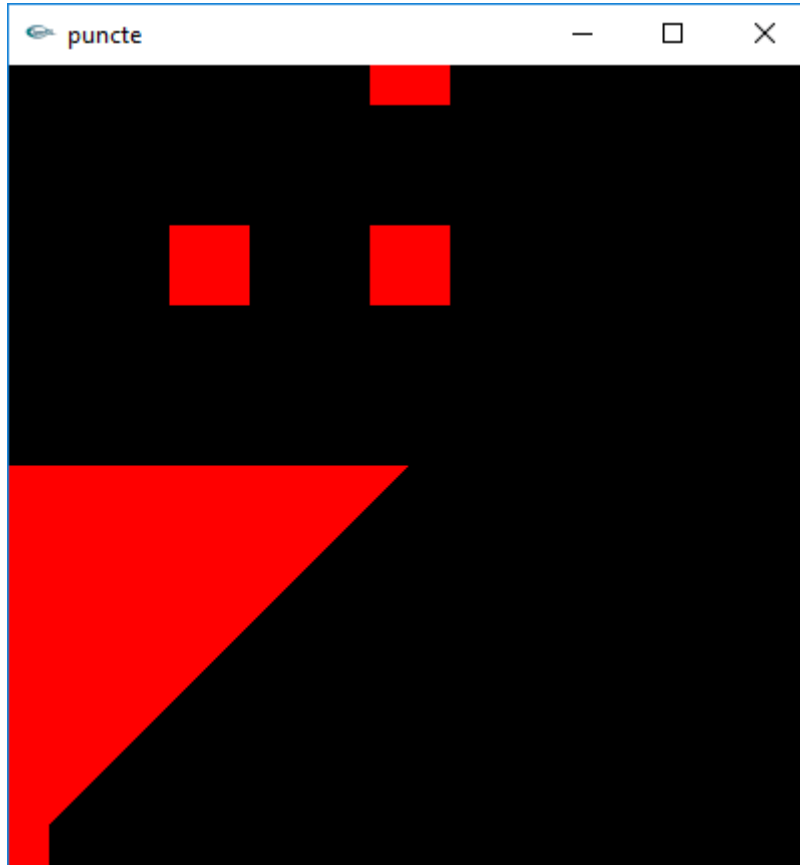
void main(void)
{
    glutInitWindowSize(400,400);
    glutCreateWindow("Exemplu");
    glutDisplayFunc(Display);
    glutMainLoop();
}
```

4.6. Probleme propuse

Probleme Propuse

Sa se scrie codul corespunzator, folosind functii OpenGL pentru a rezolva aplicatiile 1-3

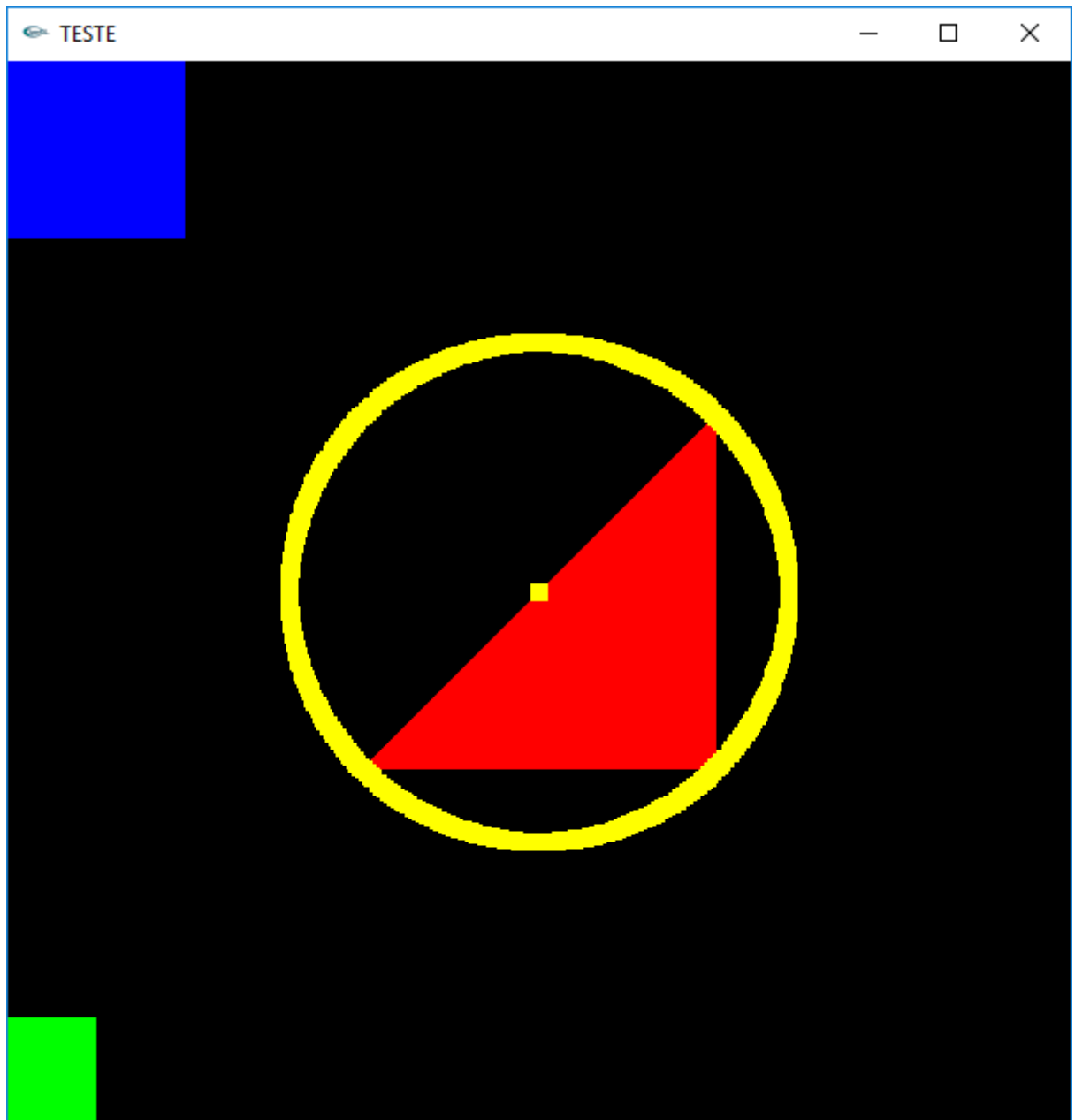
Aplicatia 1



Aplicatia 2



Aplicatia 3



5. Unity Game Engine

OBIECTIVE GENERALE

- Cunoasterea interfeței grafice pentru Unity Game Engine;
- Înțelegerea principalelor tutoriale de la <https://unity3d.com/learn>
- Înțelegerea celor mai utilizate funcții din clasa MonoBehaviour

Keywords: Unity 3D, clasa MonoBehaviour

Cuprins:

5.1 Unity Game Engine;

5.2 Funcții din clasa MonoBehaviour;

5.3 Parcurgerea unui tutorial -
<https://unity3d.com/learn/tutorials/modules/beginner/live-training-archive/creating-a-scene-menu?playlist=17111>

5.1 Unity Game Engine

Unity este un game engine cu ajutorul căruia putem realiza jocuri 2D cât și 3D pentru PC, console, telefoane mobile și website. Acesta a fost lansat în 2005 și este scris în C, C++ și C#. Inițial acest game engine a fost lansat doar pentru Sistemele de operare Mac, iar în versiunea lansată inițial conținea shadere orientate OpenGL. În următoarele versiuni de Unity 1.x s-a inclus suport pentru rularea jocurilor pe sistemul de operare Windows.

Link de descărcare : <https://unity3d.com/get-unity>

Unity este un game engine care a evoluat mult de-a lungul anilor, reușind să își extindă aria de dispozitive target. A devenit mult mai ușor de utilizat, oferind programatorului cât și amatorului programator posibilitatea de a crea jocuri 2D cât și 3D cu un bagaj mic de cunoștințe. Cu timpul a devenit tot mai popular, tot mai cunoscut prin jocurile create și s-a format o întreagă comunitate de developeri amatori și profesioniști care pot pune la dispoziție răspunsuri la întrebările utilizatorilor. De asemenea pe site-ul oficial (www.unity3d.com) se găsesc o mulțime de tutoriale inclusive live, cu scopul de a demonstra cât de practic și puternic poate fi acest game engine, iar ca un plus, noua versiune lansată în acest an Unity 5, este gratuită. Unity suportă 3 limbaje diferite (C#, JavaScript și Boo), oferind programatorilor

posibilitatea sa lucreze in limbajul preferat. In acest moment C# este limbajul cel mai utilizat, Boo insa este un limbaj de programare aflat inca in umbra folosit de o parte mica a programatorilor in Unity.

In Unity se lucreaza in Monodevelop, ceea ce este o versiune customizata pentru Unity. Monodevelop este un mediu de lucru open source pentru Linux si Windows. Contine functii similare cu Microsoft Visual Studio si NetBeans precum completarea automata a codului sau crearea interfetelor grafice (GUI), suporta Boo, C#, JavaScript etc.

Unity game engine are o interfata simpla si intuitiva, ferestrele cu care interactionezi cel mai des sunt urmatoarele:

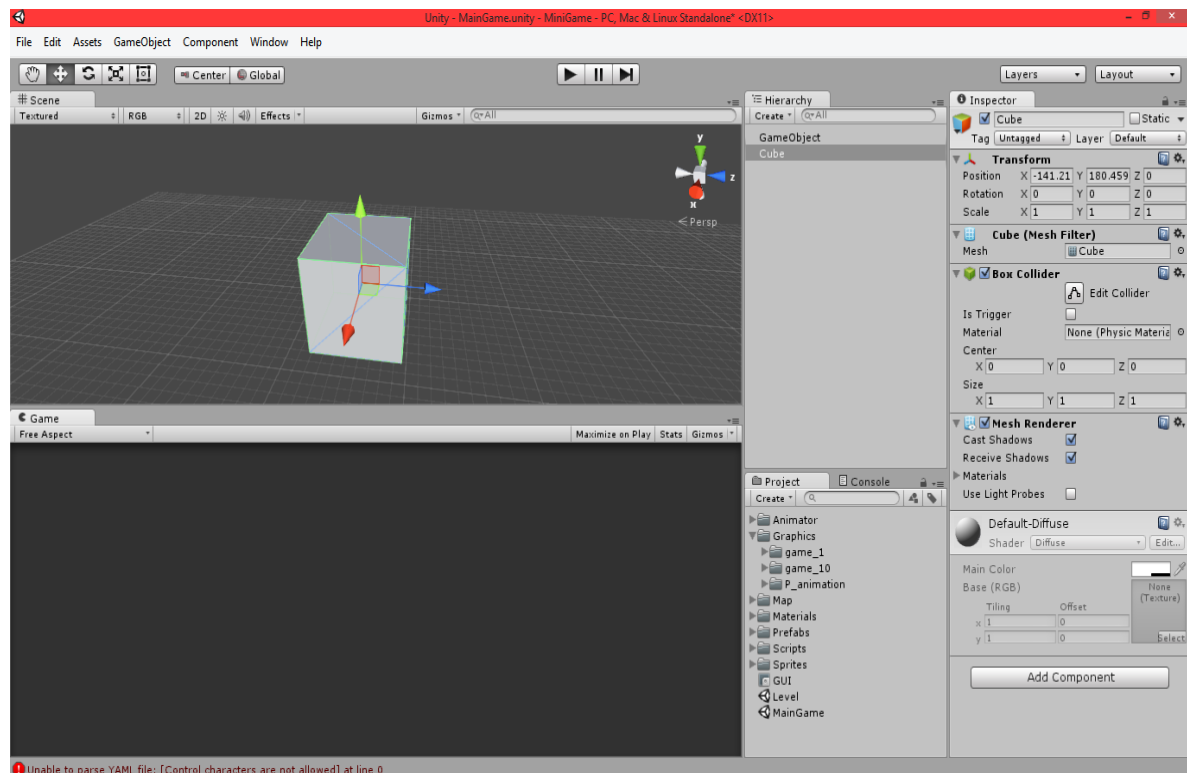
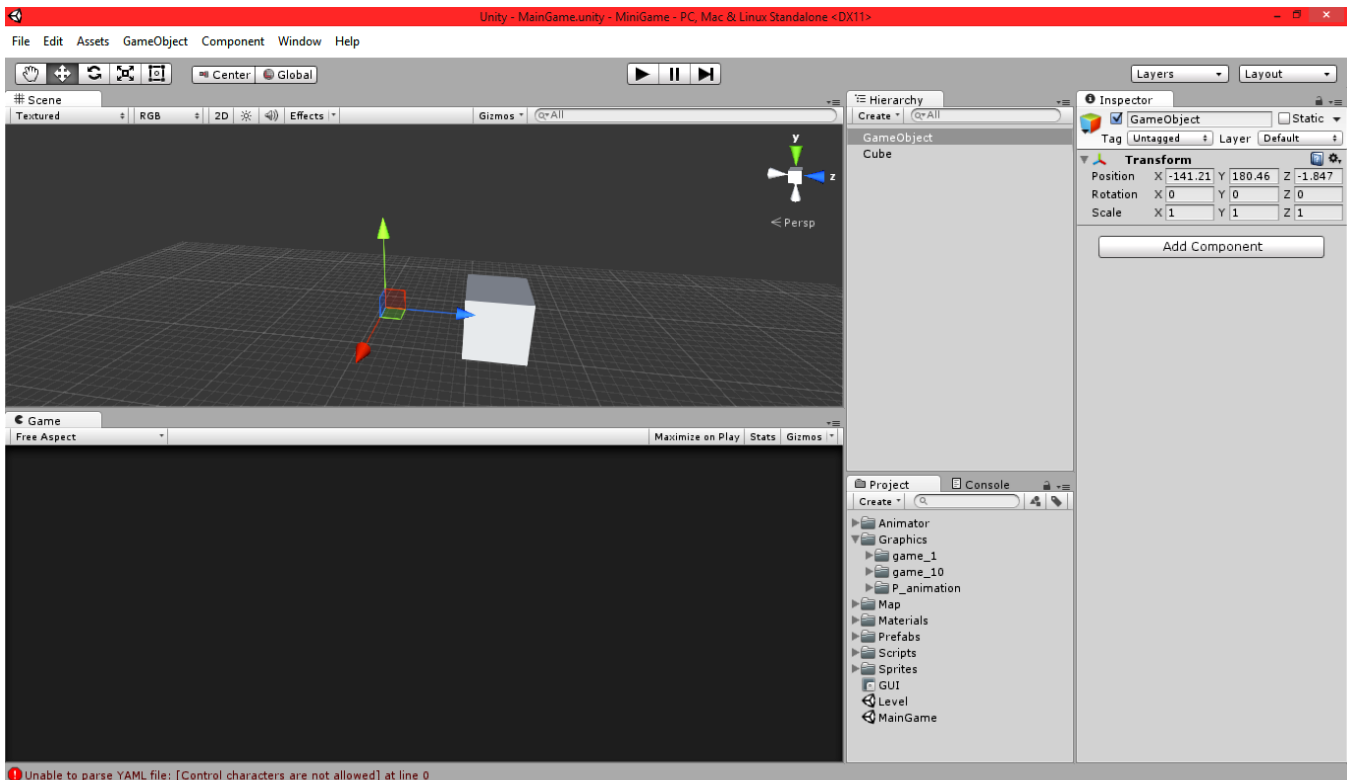
- 1. Scena - reprezinta toate componentele pe care le ai in joc (fiecare scena contine o camera);**
- 2. Game - reprezinta fereastra cu jocul si este ceea ce camera ta vede in scena;**
- 3. Ierarhie – contine structurate obiectele din scena;**
- 4. Proiect - reprezinta proiectul utilizatorului si contine scripturile, prefaburi, materiale, texture, assets etc.**
- 5. Consola – pentru a obtine informatii legate de proiectul curent, daca sunt anumite erori de compilare, de sintaxa in cod precum si zona in care se va putea urmari comportamentul variabilelor , adica un debug.**
- 6. Inspectorul - aici apare fiecare componenta a obiectului tau (fiecare obiect este considerat un GameObject), se poate modifica fiecare parametru droit.**

Foarte important este faptul ca forma, obiect contine 3 componente fara de care nu am putea randat obiectul. Cele trei componente sunt:

- -Material
- -Mesh Filter
- -Mesh Renderer

Ca prim exemplu al unei componente care poate fi adaugat il reprezinta componenta Rigidbody, ce permite obiectului sa interactioneze cu mediul inconjurator.

Unity este un game engine ce iti ofera posibilitatea de a iti porta jocul pe o platforma dorita de tine, iti pune la dispozitie o multime de pachete ce acopera totul incepand de la texture si animatii pana la intregi proiecte drept model. Aceste pachete sunt produse de catre Unity Technologies dar si de catre membrii ai comunitatii, care le pot posta gratuit sau vinde.



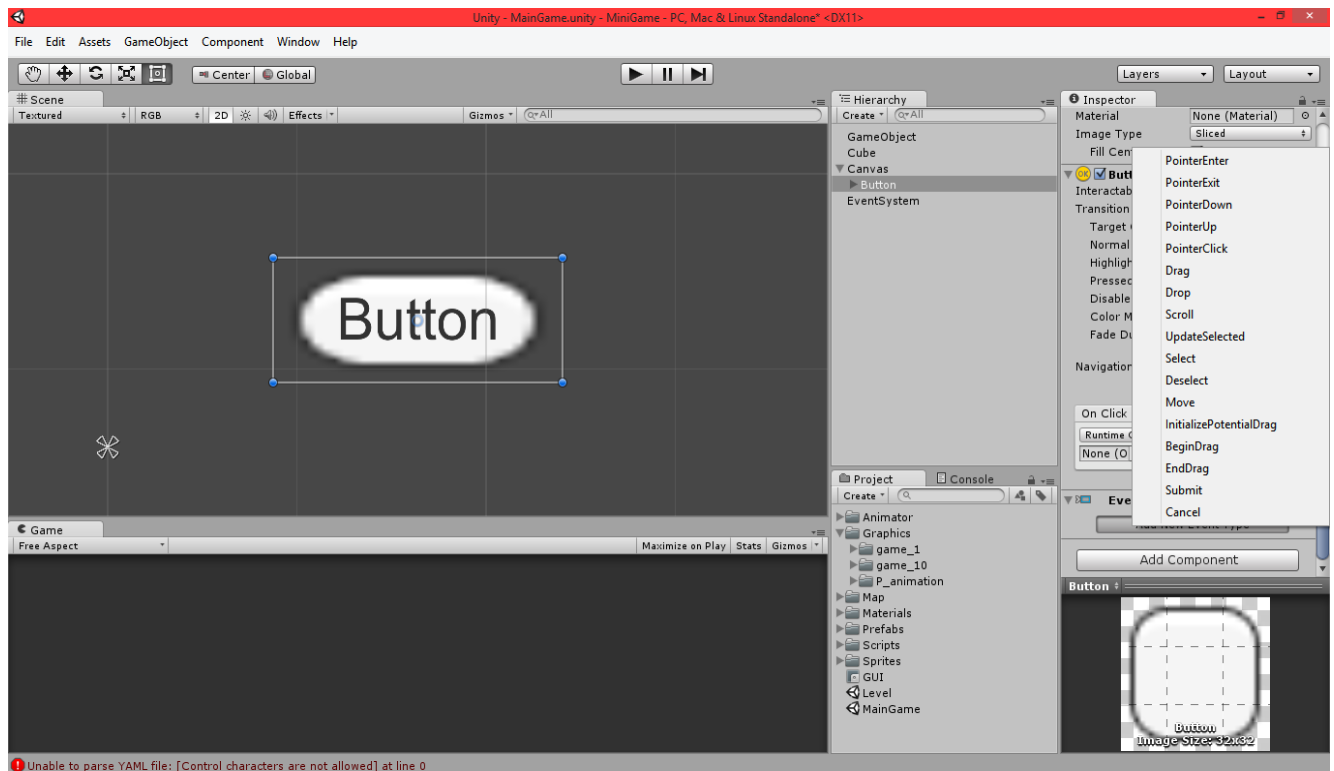
5.2 Functii din clasa MonoBehaviour

Unele din cele mai des utilizate functii din clasa MonoBehaviour sunt :

```
Awake (){
// Fiecare instructiune este rulata inca de la incarcarea scriptului, chiar daca componenta de
script nu este activata
}
Start(){
// Functia Start este apelata dupa functia Awake si doar atunci cand scriptul este activat.
}
Update(){
// Functie care este apelata la fiecare frame, si in aceasta functie sunt introduse comenzile de
control.
}
FixedUpdate(){
// Functie care este apelata la un numar de frameuri constante, pot fi folosite pentru a adauga o
fora unui obiect, la fiecare miscare a obiectului functia FixedUpdate este apelata.
}
LateUpdate(){
Functie apelata la finalul rularii instructiunilor din Update()
}

IEnumerator functie(){
    Instructiuni
    Yield return new WaitForSeconds(1.4f);
    Instructiuni
    ...
}
astfel ai controlul asupra derularii unor actiuni.
*/
}
OnGui(){
// Putem crea butoane, ferestre, o interfata cu care utilizatorul sa interactioneze. Aceasta functie
este apelata la fel ca si functia Update(), la fiecare frame.
}
```

Începând cu Unity 4.6 dezvoltatorii de jocuri au accesul la un UI Tool care nu necesită scrierea de foarte mult cod. Crearea unei interfețe cu care utilizatorul să interacționeze este foarte ușoară, deoarece avem la dispoziție un Animator Controller care ne ajută să menținem anumite animații pe un obiect și să le controlăm în funcție de anumite condiții.



<https://unity3d.com/learn/tutorials/modules/beginner/live-training-archive/the-new-ui>

Unity folosește pentru programarea JavaScript o versiune modificată a acesteia numită UnityScript. Chiar dacă în comunitatea Unity, până și în documentația oficială de pe site-ul programului, lumea vorbește despre cele două limbaje ca și cum ar fi echivalente și interschimbabile ele sunt două limbaje foarte diferite. Se aseamănă din punct de vedere sintactic dar au semantici foarte diferite.

În comunitatea Unity lumea vorbește despre UnityScript folosind numele JavaScript ceea ce nu este tocmai corect. Chiar dacă JavaScript este un nume generic și poate fi folosit pentru o implementare a standardului ECMAScript, UnityScript nu se apropie de acest standard dar nici nu încearcă acest lucru. UnityScript este un limbaj proprietar și nu urmărește niciun standard specific, fiind modificat constant de dezvoltatorii Unity.

Majoritatea bibliotecilor JavaScript nu vor merge dacă sunt doar copiate în Unity.

Unii oameni cred că nu există diferențe între aceste două limbaje și asta creează o problemă atunci când cineva, având o problemă, caută pe internet o soluție și în loc să caute rezolvarea pentru limbajul folosit de Unity caută rezolvarea pentru limbajul JavaScript ceea ce face rezolvarea problemei mult mai grea.

De asemenea este destul de problematic să specifice mereu când cauți rezolvarea unei probleme dacă te referi la limbajul real “JavaScript” sau “UnityScript”.

În comparație cu JavaScript, UnityScript folosește clase. De asemenea în UnityScript o clasă odată definită este mai mult sau mai puțin fixă pe durata rulării programului. În orice caz,

sistemul de clase are avantajul de a fi un limbaj mai familiar si usor de citit.

In UnityScript numele fisierului conteaza. Limbajul incearca sa reduca codul care trebuie scris. Majoritatea fisierelor UnityScript reprezinta clase deci automat numele fisierului UnityScript defineste clasa pe care continutul scriptului o implementeaza.

Spre deosebire de JavaScript, UnityScript necesita inserarea manuala de “;”. JavaScript incearca sa faca inserarea de “;” inutila, dar acest lucru afecteaza calitatea si functionalitatea codului. UnityScript evita aceasta metoda si obliga programatorul sa puna “;” dupa aproape fiecare linie de cod.

UnityScript nu suporta declararea mai multor variabile in aceasi linie de cod, in timp ce JavaScript permite acest lucru.

C# vs JavaScript

Unity este cu siguranta cel mai flexibil program de creat jocuri 2D si 3D pe o multitudine de platforme printre care iPhone, iPad, Android si PC. Comunitatea plina de resurse, suportul pentru o varietate de limbaje de programare si motorul avansat de redare(render) ofera dezvoltatorilor de jocuri video flexibilitatea de care au nevoie pentru a creea jocuri inovative.

Unity permite codarea in 3 limbaje de programare si anume C#, UnityScript (o versiune a JavaScript modificata de Unity) si Boo. Cel mai popular dintre acestea 3 este limbajul de programare C# deoarece acesta este o ramura a limbajelor de programare C si C++ cu care majoritatea programatorilor sunt familiari, iar aceasta familiaritate ajuta la scurtarea timpului de acomodare pentru a scrie cod in Unity.

Unity nu ofera suport direct pentru traditionalul JavaScript. El foloseste o versiune modificata de JavaScript numita UnityScript. Chiar daca sintaxa ramane aceeasi, unele modificari cum ar fi folosirea de obiecte statice si omiterea anumitor dinamici JavaScript vor incurca un programator familiarizat cu JavaScript. Asta inseamna ca pentru a folosi UnityScript trebuie invatat practic un alt limbaj de programare.

Unity ofera suport pentru folosirea limbajului de programare C Sharp nativ, deci nu trebuie invatat un alt limbaj pentru a incepe codarea de jocuri video in acest game engine. Asta inseamna ca poti lucra cu orice alt programator care cunoaste acest limbaj de programare fara a intampina probleme dealungul timpului. Deasemenea acest lucru ajuta la cautarea si rezolvarea problemelor ce pot aparea, in mediul online.

Un avantaj al codarii in limbajul C Sharp este magazinul de bunuri Unity Asset Store. Majoritatea bunurilor din acesta ruleaza in C#. Desi Unity permite codarea in mai multe limbaje in acelasi proiect este recomandat codarea intr-un singur limbaj pentru a asigura o mai buna functionare a proiectului.

Este mult mai avantajos sa folosesti un limbaj nativ cand codezi pentru prima oara in

Unity deoarece C# este mult mai cuprinzator si puternic decat versiunea de JavaScript modificata in UnityScript.

Link pentru galeria cu jocurile create in Unity :

- <https://unity3d.com/showcase/gallery/games>
- <http://www.unitygamesbox.com/page/11>

5.3 Parcurgerea unui tutorial

<https://unity3d.com/learn/tutorials/modules/beginner/live-training-archive/the-new-ui>

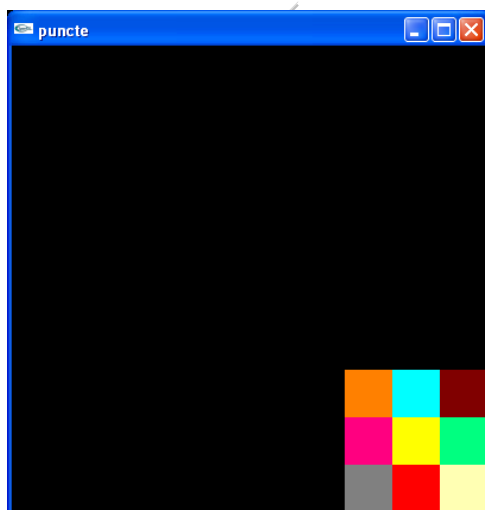
Model bilet 1

1. Caracterizati grafica digitală (2 pct).
2. Caracterizati grafica vectorială (2 pct).
3. Explicati modelele de culori RGB si CMYK (2 pct).
4. Sa se explice fiecare functie *OpenGL* utilizata in programul urmatoar (1.5 pct).

```
1.  #include <GL/glut.h>
2.  void init()
3.  {
4.    glClearColor (0.0, 0.0, 0.0, 0.0);
5.    glShadeModel (GL_FLAT);
6.  }
7.  void display()
8.  {
9.    glClear
    (GL_COLOR_BUFFER_BIT);
10.   glColor3f (1.0, 0.0, 0.0);
11.   glBegin(GL_POLYGON);
12.   glVertex2f(200.0,200.0);
13.   glVertex2f(400.0,200.0);
14.   glVertex2f(400.0, 400.0);
15.   glEnd();
16.   glFlush ();
17. }
18. void reshape (int w, int h)
19. {
```

```
20.  glViewport (0, 0, (GLsizei) w,
    (GLsizei) h);
21.  glMatrixMode
    (GL_PROJECTION);
22.  glLoadIdentity ();
23.  gluOrtho2D (0.0, (GLdouble) w,
    0.0, (GLdouble) h);
24.  }
25.  int main(int argc, char** argv)
26.  {
27.    glutInit(&argc, argv);
28.    glutInitDisplayMode
    (GLUT_SINGLE | GLUT_RGB);
29.    glutInitWindowSize (600, 600);
30.    glutInitWindowPosition
    (100,100);
31.    glutCreateWindow (argv[0]);
32.    init ();
33.    glutDisplayFunc(display);
34.    glutReshapeFunc(reshape);
35.    glutMainLoop();
36.    return 0;
37. }
```

5. Sa se scrie setul de comenzi (utilizand *OpenGL*) care, in urma executiei, vor conduce la fereastra din figura alaturata (1.5 pct).



Model bilet 2

1. Prezentați cel puțin 4 idei principale din referatul întocmit (2 pct).
2. Caracterizați grafica vectorială (2 pct).
3. Explicați modelele de culori RGB și CMYK (1 pct).
4. Să se explice fiecare funcție *OpenGL* utilizată în programul următor (2 pct).

<pre>1. #include <GL/glut.h> 2. void init() 3. { 4. glClearColor (0.0, 0.0, 0.0, 0.0); 5. glShadeModel (GL_FLAT); 6. } 7. void display() 8. { 9. glClear (GL_COLOR_BUFFER_BIT); 10. glColor3f (1.0, 0.0, 0.0); 11. glBegin(GL_POLYGON); 12. glVertex2f(200.0,200.0); 13. glVertex2f(400.0,200.0); 14. glVertex2f(400.0, 400.0); 15. glEnd(); 16. glFlush (); 17. } 18. void reshape (int w, int h) 19. { 20. glViewport (0, 0, (GLsizei) w, (GLsizei) h);</pre>	<pre>21. glMatrixMode (GL_PROJECTION); 22. glLoadIdentity (); 23. gluOrtho2D (0.0, (GLdouble) w, 0.0, (GLdouble) h); 24. } 25. int main(int argc, char** argv) 26. { 27. glutInit(&argc, argv); 28. glutInitDisplayMode (GLUT_SINGLE GLUT_RGB); 29. glutInitWindowSize (600, 600); 30. glutInitWindowPosition (100,100); 31. glutCreateWindow (argv[0]); 32. init (); 33. glutDisplayFunc(display); 34. glutReshapeFunc(reshape); 35. glutMainLoop(); 36. return 0; 37. }</pre>
--	---

5. Explicați ce rezultă în urma testării setului de comenzi (2 pct).

<pre>1. #include <GL/glut.h> 2. #include <stdlib.h> 3. #include <math.h> 4. void Display(void) a. { b. glClear(GL_COLOR_BUFFER_BIT); c. glBegin(GL_LINES); d. for(int i=0;i<100;++i) i. { ii. glVertex3f(0,0,0); iii. glVertex3f(1- i/100.0,i/100.0,0);</pre>	<pre>iv. } e. glEnd(); f. glFlush(); g. } 5. int main(void) a. { b. glutCreateWindow("model"); c. glutDisplayFunc(Display); d. glColor3f(1,1,0); e. glutMainLoop(); f. return 0; g. }</pre>
---	--

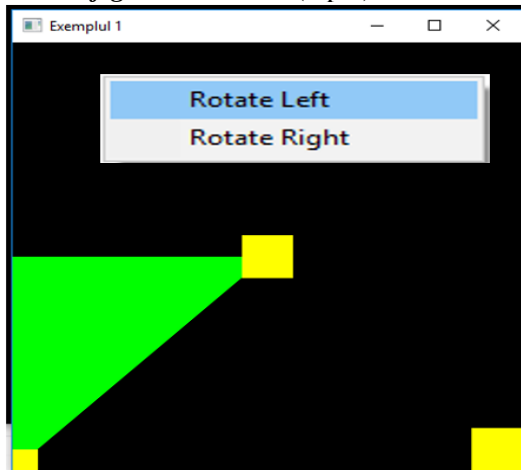
Model bilet 3 /Partea I.

1. Sa se explice functiile *OpenGL* utilizate in programul urmator, cat si ce rezulta dupa testarea aplicatiei (3 pct).

```
#include "stdafx.h"
#include <gl/freeglut.h>
#include<math.h>
#include<stdio.h>
#include <stdlib.h>
void display()
{ glClear(GL_COLOR_BUFFER_BIT);
glBegin(GL_TRIANGLES);
glColor3f(1.0, 0.0, 0.0);
glVertex2i(1, 0); glVertex2i(0, 0);
glVertex2i(0, 1);
glEnd(); glPointSize(1.0);
glColor3f(1, 1, 1);
glBegin(GL_POINTS);
for (int i = 0; i<10; i++) {
```

```
glVertex3f(cos(2 * 3.14*i / 10.0), sin(2 *
3.14*i / 10.0), 0);      }
glEnd();glFlush();
}
int main(int argc, char** argv)
{
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE |
GLUT_RGB);
glutInitWindowSize(400, 400);
glutInitWindowPosition(400, 100);
glutCreateWindow("aplicatii");
glutDisplayFunc(display);
glutMainLoop();
return 0; }
```

2. Sa se scrie setul de comenzi (utilizand *OpenGL*) care, in urma executiei, vor conduce la fereastra din figura alaturata (3 pct).



3. Prezentați cel puțin 6 idei principale din referatul întocmit (1 pct).

Partea II. Completați răspunsul corect:

1	2	3	4

- Alegeti afirmatia incorecta:** a) Imprimantele 3D sunt capabile sa realizeze obiecte in trei dimensiuni - lungime, latime si inaltime. b) Imprimantele 3D nu pot crearea obiecte fizice compuse din materiale precum plastic, metal, sticla, ceramica, rasina. c). Pentru a printa un obiect 3D este nevoie, de un fisier care sa defineasca forma si caracteristicile interne ale obiectului 3D. (0,5 pct)
- La elaborarea specificatiei OpenGL au stat la baza urmatoarele principii:** a) independenta fata de platforma hardware si fata de sistemul de operare; b) dependenta fata de platforma hardware si fata de sistemul de operare; c) doar dependenta fata de platforma hardware. (0,5 pct)
- Propriile formate de fisiere in Photoshop sunt:** a) PSD și PSB; b) PSD si JPG; c) JPG si PNG. (0,5 pct)
- Alegeti raspunsul corect, corespunzator functiei glutInit (int* argc, char** argv):** a) funcția inițializează GLUT-ul folosind argumentele din linia de comandă și întoarce o variabila de tip int care reprezintă numărul ferestrei care se initializeaza; b) nu este corect definită; c) funcția inițializează GLUT-ul folosind argumentele din linia de comandă si trebuie apelată înaintea oricăror alte funcții GLUT sau OpenGL. (0,5 pct)

Model bilet 4

Partea I Completati raspunsul corect:

1	2	3	4

5. Care este rolul primitive **GL_LINE_LOOP**? a) Desenează o linie poligonală formată din segmentele (v_0, v_1) , (v_1, v_2) , ... (v_n, v_0) ; b) Desenează linia poligonală formată din segmentele (v_0, v_1) , (v_1, v_2) , ... (v_{n-2}, v_{n-1}) ; c) Desenează segmentele de dreaptă izolate (v_0, v_1) , (v_2, v_3) , ... dacă n este impar ultimul vârf este ignorat. (0,5 pct)
6. In **OpenGL** stergerea unui submeniu se realizeaza utilizand functia? a) `glutDetachMenu`; b) `glutDestroyMenu`; c) `glutRemoveMenuItem`. (0,5 pct)
7. Pentru functia `glutMouseFunc(void(*MouseFunc)(unsigned int button, int state, int x, int y))` parametrul `button` poate avea valorile? a) `GLUT_LEFT_BUTTON`; b) `GLUT_BUTTON_MIDDLE`; c) `GLUT_BUTTON_RIGHT`. (0,5 pct)
8. Cu ajutorul carei functii se poate desena in **OpenGL** un cub cu latura 1? a) `glutWireCube(1)`; b) `glutCubeWire(1)`; c) `glutCubeSolid(1)`. (0,5 pct)

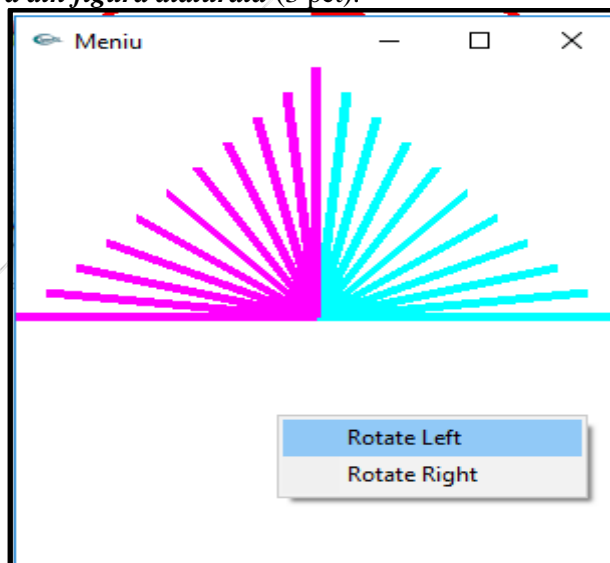
Partea II.

1. Sa se explice functiile *OpenGL* utilizate in programul urmator, cat si ce rezulta dupa testarea aplicatiei (3 pct).

```
#include "stdafx.h"
#include <gl/freeglut.h>
void Display()
{
    static float alpha = 20;
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0.1, 0.9, 1.0);
    glLoadIdentity();
    glPopMatrix();
    glRotatef(alpha, 1.9, 0.6, 0);
    glutWireTeapot(0.3);
    glPushMatrix();
    glFlush();
}
```

```
alpha = alpha + 0.1;
glutPostRedisplay();
}
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitWindowPosition(10, 10);
    glutCreateWindow("Test");
    glutDisplayFunc(Display);
    glutMainLoop();
    return 0;
}
```

2. Sa se scrie setul de comenzi (utilizand *OpenGL*) care, in urma executiei, vor conduce la fereastra din figura alaturata (3 pct).



3. Prezentați cel puțin 6 idei principale din referatul întocmit (1 pct).