

**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 procedurală și alte discipline studiate în anul I și anul II sem I.*

*O neînțelegere a noțiunilor 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 procentele menționate:**

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

**Fiecare student va realiza un proiect care contine doua parti: un referat de maxim 6 pagini, despre unul din subiectele tratate in curs si minim 8 programe asemanatoare cu cele tratate in suportul de curs. Toate functiile utilizate vor fi explicate ca rol si parametrii.**

**Proiectul se va sustine in ultimele doua laboratoare.**

Bibliografie	<ol style="list-style-type: none"><li>1. Suportul propriu de curs pentru studentii de la invatamant ID (accesibil de pe platforma e-learning/CD)</li><li>2. Baciu, R., Programarea aplicațiilor grafice 3D cu OpenGL, Editura Albastră, Cluj-Napoca, 2005.</li><li>3. Moldoveanu, F., Racoviță, Z., Hera, G., Petrescu, Ș., Zaharia, M., Grafica pe calculator, Editura Teora, București, 1996.</li><li>4. Ionescu, F., Grafica în realitatea virtuală, Editura Tehnică, București 2000.</li><li>5. Hearn, Donald, Backer, M. Pauline, Computer Graphics, Prentice-Hall, Inc, Englewood Cliffs, New Jersey, 1986</li><li>6. Foley, J., A. van Dame, Feiner, S.K., Hughes, J.F., Computer Graphics: principles and practice, Addison Wesley Publishing Company, second edition, 1993.</li><li>7. Neider, J., Davis, T., Woo, M., OpenGL Programming Guide, Addison-Wesley, Menlo Park, 1993.</li><li>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, 2013</li><li>9. <a href="https://docs.unity3d.com/Manual/index.html">https://docs.unity3d.com/Manual/index.html</a></li><li>10. <a href="https://developer.nvidia.com/opengl">https://developer.nvidia.com/opengl</a></li><li>11. <a href="https://www.opengl.org/">https://www.opengl.org/</a></li></ol>
--------------	---

## 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>Softurile 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>	33
4.1	<i>Functii pentru lucru cu ferestre .....</i>	34
4.2	Gestiunea meniurilor .....	36
4.3	Afișarea obiectelor 3D predefinite.....	49
4.4	Recapitulare functii principale in OpenGl.....	57
4.5	Probleme rezolvate .....	66
4.6.	Probleme propuse.....	76
5.	Unity Game Engine.....	78
5.1	Unity Game Engine .....	78
5.2	Functii din clasa Monobehaviour .....	80
5.3	Parcurserea unui tutorial .....	87
	Model bilet 1.....	96
	Model bilet 2.....	97
	Model bilet 3 /Partea I.....	98
	Model bilet 4.....	99

# 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, compozиii etc.

În baza tehnologiilor graficii computerizate s-au dezvoltat:

- Interfață 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**

OBSERVATIE: **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 grafica (**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 3 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. Imaginele procesate cu ajutorul scannerului sau aparatelor foto sănt formulate ca raster.

- *Neajunsurile* – e complicata 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 necesitați 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 subtractiv 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 necesitați. 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 și utilizarea vectorizării

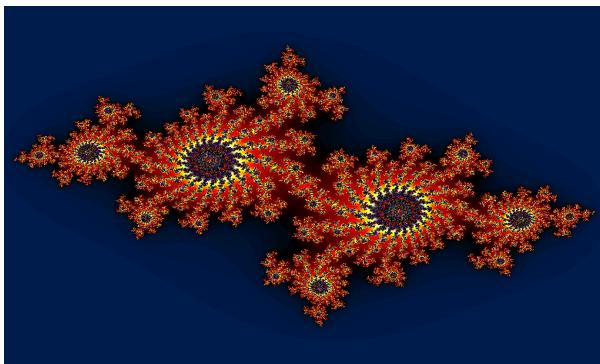
Fractal Explorer este un generator de fractali care poate produce imagini frumoase, misterioase matematic generate. Fractal Explorer poate face seturi de fractali clasici, polinomial, (cum ar fi Mandelbrot, Julia, Newton și variante ale lor). - Fractali complecsi 4D (numiți «quaternions»), «atractori 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 iterăției).

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 minusculе 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 diferențele 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 fară 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 intr-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 acesteia 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.

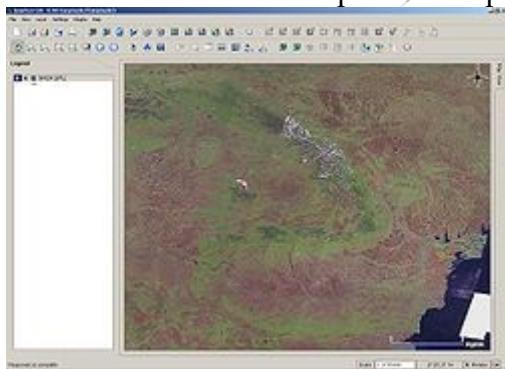
Datorită flexibilității în ceea ce privește rezoluția imaginilor vectoriale, acestea sunt folosite intensiv pentru crearea materialelor ce trebuie imprimate la mărimi foarte diverse: același fișier poate fi folosit pentru un card de vizită cât și pentru un panou publicitar, în ambele cazuri rezultatele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

O altă aplicație semnificativă a graficii vectoriale este în modelarea suprafetelor 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 e 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 repartiția 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 geografic 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 Informaticice 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 - aşa 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 funcționare a Interfețelor API.*
- *Intelegerea modului de funcționare pentru DirectX.*

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

#### Cuprins

- |  |
|--|
| 2.1 Softurilor dedicate pentru realizarea fisierelor grafice |
| 2.2 Utilizarea Interfețelor API                              |
| 2.3 DirectX rol și funcționare                               |

### 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 luminositate 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 bîti, 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 stabileste 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 combina cadre fotografice de expunerii diferite încă din același timp și 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 forma 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 forma, 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, Freelanec 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 aplicației 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 interfața API este un cod sursă oferit de către sistemul de operare sau o librerie pentru a permite apeluri la serviciile care pot fi generate din API-urile respective de către un program. Termenul API este folosit în 2 sensuri:

- O interfață 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 incorporate în Software Development Kit (SDK)

Modelul de design a Interfetelor API

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

Unele Interfete API, cum sunt cele standard pentru un sistem de operare, sunt implementate ca librării de cod separate care sunt distribuite împreună cu sistemul de operare. Altele au integrată funcționalitatea interfetei API direct în aplicatie. 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 aplicatie.

O interfață API care nu necesita 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)
- svgalib 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 colectie **Application Programming Interface (APIs)** care se ocupa cu sarcinile legate de multimedia, in special programarea jocurilor si video pe platformele Microsoft. La inceput numele acestor API-uri incepeau 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 inlocuia numele diferitelui API. Mai tarziu cand Microsoft a inceput 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 aplicatii software pentru vizualizare si taskuri grafice precum CAD/CAM.

La sfârșitul anului 1994 era gata sa lanseze urmatorul sau sisteme de operare, Windows 95. Cei de la Microsoft aveau teama ca sistemul de operare precedent al său MS-DOS sa fie vazută 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 initiala a lui DirectX a fost foarte lenta. Programatorii se temea ca DirectX sa fie inlocuit 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 dificila de a testa fiecare DirectX lansat cu o multitudine de hardware si software. O varietate de placi grafice, placi audio, placi de baza, CPU-uri, device-uri de input, jocuri si alte aplicatii multimedia erau testate cu fiecare lansare beta sau lansare finala. De asemenea echipa DirectX a construit si distribuit teste care permitea industriei hardware sa isi testeze hardware-ul si driverele pentru compatibilitate. Înainte sa apară DirectX, Microsoft includea OpenGL. La acel moment pentru OpenGL trebuia hardware foarte performant si era focusat pe ingineri si utilizatori de CAD. Direct3D a fost intenționat precum un partener minor a 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 interactiva precum jocuri , dând dezvoltatorilor posibilitatea de a alege intre Direct3D sau OpenGL precum un API 3D. O “luptă” intre dezvoltatorii care susțineau standardul deschis OpenGL si cei care susțineau Direct3D a lui Windows.

In versiunea specifica consolelor DirectX era API-ul utilizat in Microsoft Xbox, Xbox 360, si Xbox One. API-ul a fost dezvoltat precum o colaborare intre Microsoft si nVidia – partenerul care a dezvoltat componentele grafice hardware. In 2002, Microsoft a lansat DirectX 9, care suporta un program mult mai lung de shadere decât înainte cu un pixel si vertex shader versiune 2.0. Microsoft a continuat sa actualizeze DirectX introducând shader Model 3.0 in DirectX 9.0c. care a fost lansat in August 2004.

DirectX este compus din mai multe API-uri:

- Direct3D (D3D) – pentru grafica 3D
- DXGI – pentru a enumera adaptoarele si monitoarele pentru Direct3D 10 si 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 si pentru media streaming. DirectX Transform pentru interacțiuni Web si Direct3D Retained Mode pentru grafica 3D de nivel înalt.
- DirectX Diagnostics (DxDiag) – o unealta pentru a diagnostica si genera rapoarte pentru componente relevante pentru DirectX cum ar fi audio, video si driverele de input
- DirectX Media objects – suporta obiectele de streaming cum ar fi: encodere, decodere si efecte.
- DirectSetup – pentru instalarea componentelor DirectX si detectia 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 aceasta componenta in favoarea lui Direct2D
- DirectInput – pentru interfața cu device-urile de input precum tastaturi, mouse, joystick-uri precum si alte controller de jocuri. A fost înlocuit după versiune 8 cu XInput pentru Xbox 360 sau standardul WM\_INPUT pentru tastatura si mouse.
- DirectPlay – pentru comunicarea pe rețea. A fost înlocuit după versiune 8 in favoarea Games for Windows Live si Xbox Live
- DirectSound – pentru redarea si înregistrarea sunetelor. Înlocuit cu XAudio2 si XACT3
- DirectMusic – pentru redarea colonei sonore autorizate prin DirectMusic Producer. Înlocuit cu XAudio2 si XACT3

## DirectX 10

Un update important a lui DirectX a fost DirectX 10. Aceasta a fost disponibil doar in Windows Vista sau o versiune mai noua, versiunile mai vechi de Windows fiind limitate la versiunea 9.0c. Modificările aduse lui DirectX 10 au fost majore. Multe componente au fost depreciate si înlocuite

cu unele mai noi fiind păstrate doar pentru compatibilitate: DirectX 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 redă 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 aşa cum ar rula în Windows XP
- Direct 3D 9Ex – permite acces total la noile capabilități 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 capabilitățile de randare și flexibilitate, inclusiv Shader Model 4.

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

## DirectX 11

Microsoft a dezvoltat DirectX 11 în 2008 cu următoarele caracteristici:

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

Direct 3D 11 rulează pe orice versiune de Windows începând cu Windows 7. Partea din nou API, cum ar fi multi-threaded poate fi suportată și de hardware compatibil Direct3D 9/10/10.1 dar șezălare 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 a 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 adăugă noi caracteristici lui Direct2D precum realizări geometrice. **DirectX 11.X** este un superset a 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 încărării suplimentare aduse de drivere. Acum dezvoltatorii își pot implementa propria listă de comenzi și buffere direct în GPU, permitând utilizarea eficientă a resurselor prin calcul paralel. Suport pentru utilizarea mai multor placi grafice pe același sistem simultan. Înainte de DirectX 12 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-ri legate – va permite DirectX să percepă 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 placi 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 oferă DirectX, dar prin combinare de*

*librarii - 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 folosite 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 Aplicatii 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.

Desi 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](http://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. Funțiile OpenGL sunt apelabile din limbajele Ada, C, C++ și Java. Din consorțul 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.

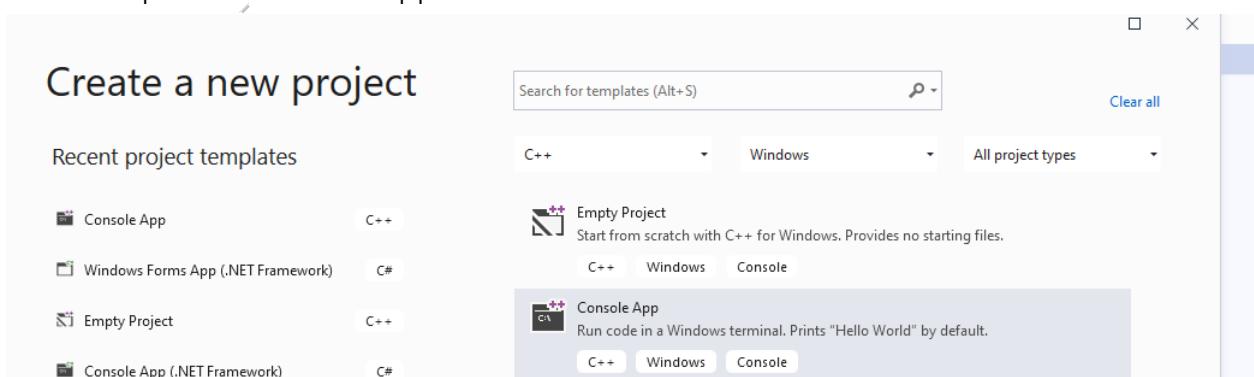
În redarea obiectelor tridimensionale, biblioteca OpenGL folosește un număr redus de primitive geometrice (puncte, linii, poligoane), iar modele complexe ale obiectelor și scenelor tridimensionale se pot dezvolta particularizat pentru fiecare aplicație, pe baza acestor primitive. Dat fiind că OpenGL 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:

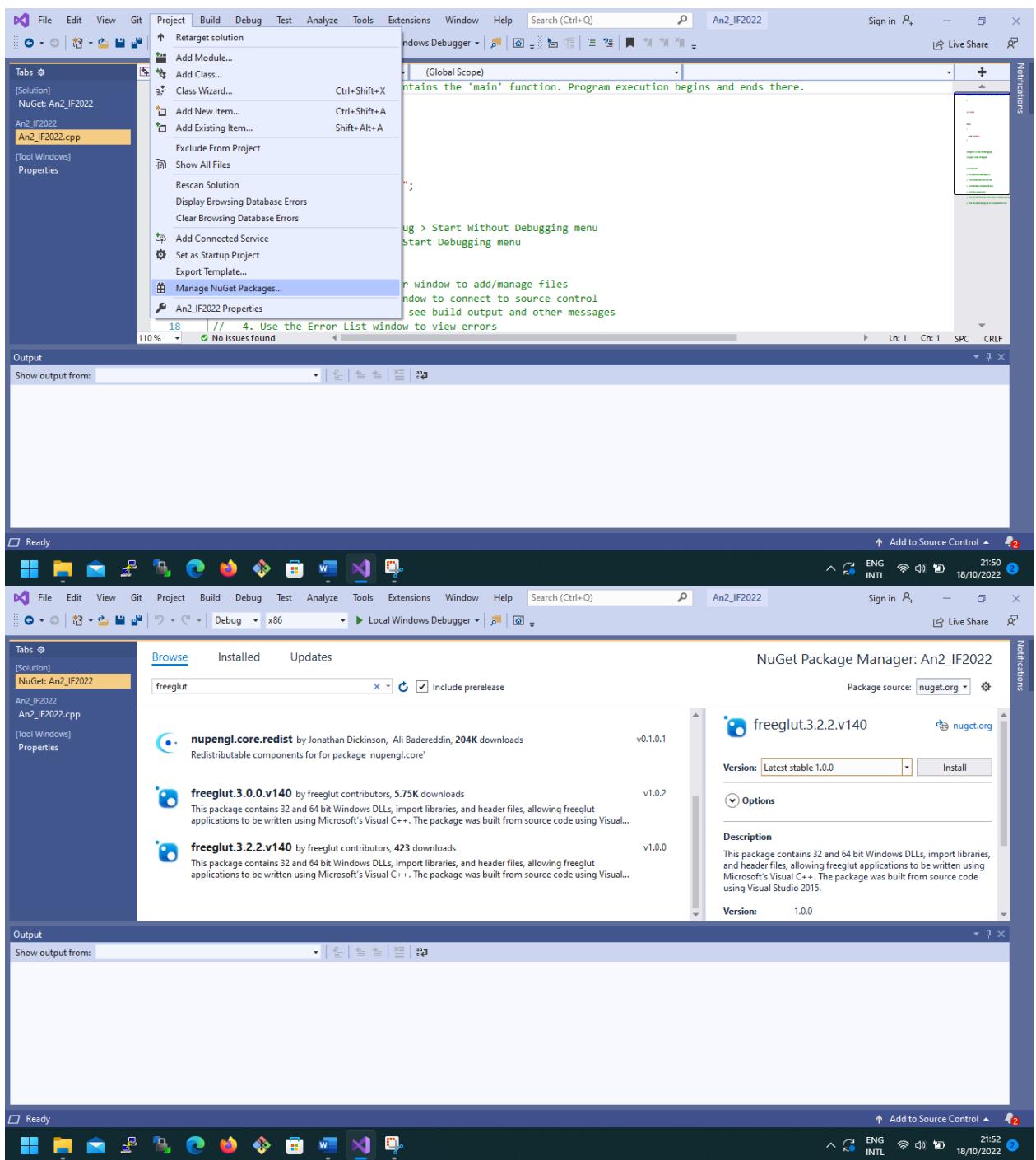
- *Biblioteca de funcții utilitare GLU (OpenGL Utility Library) permite definirea sistemelor de vizualizare, redarea suprafețelor curbe și alte funcții grafice.*
- *Pentru fiecare sistem Windows există o extensie a bibliotecii OpenGL care asigură interfața cu sistemul respectiv: pentru Microsoft Windows, extensia WGL, pentru sisteme care folosesc sistemul X Window, extensia GLX, pentru sisteme IMB 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.*

Utilizare freeglut in Microsoft Visual Studio

1. Creare proiect-console application



2. Din Project->Manage NuGet Packages ->Browse ->search freeglut->freeglut-> install si ->nupengl.core->install



## 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 administarea 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 functiilor 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 oricărora 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 şablon) 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.

Atenție:

Funcția Display (a aplicației) este apelată oridecă 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 button, int stare, int x, int y)
{
    switch(button)
    {
        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 opriță 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 supafețe 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 afişare 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 afişare Windows pentru contextul de redare OpenGL și accesul la contextul de afişare 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 afişare (window) și au valorile implicate 0, 0. Parametrii width și height specifică lățimea, respectiv înălțimea, porții de afişare, dată ca număr de pixeli.

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

ATENTIE: 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 bufferi 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 afişare. 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 generația elementelor componente, pentru noua imagine. Chiar dacă ar fi posibilă generația și înscrarea 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 pălpărire a imaginii. În grafica interactivă timpul necesar pentru înscrarea 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 pălpări.

În OpenGL comutarea bufferelor este efectuată de funcția `SwapBuffers()`. Această funcție trebuie să fie apelată la terminarea generației 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 ocupată, atunci grupul de vârfuri care descriau inițial primitiva respectivă este înlocuit cu un alt grup, în care pot apărea 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, înscriind 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 neschimbătă 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. Cordonatele 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 dă 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);
```

Există 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 ( $v_0, v_1$ ), ( $v_2, v_3$ ), ... și.a.m.d. Dacă n este impar ultimul vârf este ignorat
GL_LINE_STRIP	Desenează linia poligonală formată din segmentele ( $v_0, v_1$ ), ( $v_1, v_2$ ), ... ( $v_{n-2}, v_{n-1}$ )
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 ( $v_0, v_1, v_2$ ), ( $v_3, v_4, v_5$ ), și.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 $(v_0, v_1, v_2), (v_2, v_1, v_3), \dots$ s.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 $(v_0, v_1, v_2), (v_0, v_2, v_3)$ , s.a.m.d.
GL_QUADS	Desenează o serie de patrulatere $(v_0, v_1, v_2, v_3), (v_4, v_5, v_6, v_7)$ , s.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 $(v_0, v_1, v_3, v_2), (v_3, v_2, v_5, v_4)$ , s.a.m.d. Dacă $n < 4$ , nu se desenază 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 unpredictibil.

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 î 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 parcuse 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 (*countrerclockwise*) 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 componente de culoare R, G, B și transparență 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 tabelei culorilor și, în plus, nu se pot efectua unele dintre prelucrările grafice importante (cum sunt umbrarea, 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 umbrarea, ceața, etc.

În aplicațiile de realitate virtuală nu se poate folosi modelul de culori indexate, de aceea în continuare nu vor mai fi prezentate comenzi 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 transparentă (*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 translacție** 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  $\mathbf{T}(x,y,z)$ , dată de relația 3.2, înmulțește la dreapta matricea curentă  $\mathbf{C}$ , iar rezultatul înlocuiește conținutul matricei curente  $\mathbf{C}$ , deci:  $\mathbf{C} = \mathbf{C} \mathbf{T}(x,y,z)$ .

**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  $\mathbf{S}(x,y,z)$ , 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 Aplicatii rezolvate

#### Aplicatie Puncte

```
#include <iostream>
#include <gl/freeglut.h>
int dist, i;
//puncte.cpp
void init()
{
    glClearColor(0.0, 0.0, 0.0, 0.0);
    //glPointSize(40.0);
}
void display()
{
    glColor3f(1.0, 1.0, 0.0);
    glPointSize(40.0);
    glBegin(GL_POINTS);

    for (dist = 0, i = 1; i <= 3; i++)
    {

        glVertex2i(20 * i + dist, 20);
        dist += 40;//y
    }
    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);//specificație 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(100, 100);
    glutCreateWindow("puncte");
    init();
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutMainLoop();
}

```

Aplicatia 2

```

#include <iostream>
#include <gl/freeglut.h>
//puncte.cpp
void init()
{
    glClearColor(0.0, 0.0, 0.0, 0.0);
    //glPointSize(40.0);
}
void display()
{
    glColor3f(1.0, 1.0, 0.0);
    glBegin(GL_POLYGON); //initializare desen poligon
    glVertex2f(0.0, 0.0); //stabilire coordonate triunghi
    glVertex2f(200.0, 200.0);
    glVertex2f(0.0, 200.0);
    glEnd();
    glFlush();
    glPointSize(40.0);
    glColor3f(1.0, 0.0, 0.0);
    glBegin(GL_POINTS);
    glVertex2i(300, 300);
    glVertex2i(20, 20);
    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); //specificaare 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();
}

```

Aplicatia 3

```

#include <iostream>
#include <gl/freeglut.h>
#include<math.h>
int width = 400;
int height = 400;
int psize = 40;
int distx = 0;
int disty = 0;
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glPointSize(psize);

    glBegin(GL_POINTS);
    disty = 20;
    for (int k = 0; k < 3; k++)
    {
        distx = 20;
        for (int j = 0; j < 3; j++)
        {
            double r = ((double)rand() / (RAND_MAX));
            double g = ((double)rand() / (RAND_MAX));
            double b = ((double)rand() / (RAND_MAX));
            glColor3d(r, g, b);
            glVertex2i(40 * j + distx, 40 * k + disty);

        }
        disty += 0;
    }
    glEnd();
    glFlush();
}

void reshape(int w, int h)
{
    glViewport(0, 0, (GLsizei)w, (GLsizei)h); //stabilirea viewportului la
dimensiunea ferestrei
    glMatrixMode(GL_PROJECTION); //specificaare 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
}//end reshape()

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(400, 300);
    glutCreateWindow("Puncte");
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutMainLoop();
    return 0;
}

```

```

#include <iostream>
#include <gl/freeglut.h>

using namespace std;

void Display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0, 1, 0);
    glBegin(GL_LINES);
    // Cadran 1
    for (int i = 0; i < 20; i++)
    {
        glVertex3f(0, 0, 0);
        glVertex3f(1 - i / 20.0, i / 20.0, 0);
    } // Cadran 2
    glColor3f(1, 0.4, 0);
    for (int i = 0; i < 20; i++)
    {
        glVertex3f(0, 0, 0);
        glVertex3f(-1 + i / 20.0, i / 20.0, 0);
    } // Cadran 3
    glColor3f(1, 0.4, 1);
    for (int i = 0; i < 20; i++)
    {
        glVertex3f(0, 0, 0);
        glVertex3f(-1 + i / 20.0, -i / 20.0, 0);
    }
    // Cadran 4
    glColor3f(0.8, 0.4, 0.2);
    for (int i = 0; i < 20; i++)
    {
        glVertex3f(0, 0, 0);
        glVertex3f(1 - i / 20.0, -i / 20.0, 0);
    }
    glEnd();    glFlush();    glColor3f(0, 0, 1);
}
int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(600, 600);
    // se specifică modelul de culoare al ferestrei: un singur buffer și culoare RGB
    glutCreateWindow("laborator 2");

    glutDisplayFunc(Display);
    glutMainLoop();
    return 0;
}

```

```

Aplicatia 5
#include <iostream>
#include <gl/freeglut.h>
#include<math.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(10.0);
    glColor3f(1, 1, 1);
    glBegin(GL_POINTS);
    for (int i = 0; i < 10; i++) {
        glVertex3f(cos(2 * 3.141 * 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;
}

Aplicatie 4
#include <iostream>
#include <gl/freeglut.h>

void init()
{
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glPointSize(40.0);
    glShadeModel(GL_FLAT);
}
void desen()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0.0, 1.0, 0.0);
    glBegin(GL_POLYGON); //initializare desen poligon
    glVertex2f(0.0, 0.0); //stabilire coordonate triunghi
    glVertex2f(200., 0.);
    glVertex2f(200.0, 200.0); //stabilire coordonate triunghi
    glVertex2f(00.0, 200.0); //stabilire coordonate triunghi
    glEnd(); //sfisit desenare
                //executare functie
    glFlush();

    glColor3f(1.0, 1.0, 0.0);
    glBegin(GL_POINTS);
    glVertex2i(100, 300);
    glVertex2i(200, 300);
    glVertex2i(200, 400);
    glColor3f(1.0, 0.0, 1.0);
    glVertex2i(20, 20);
}

```

```

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, 500);
    glutInitWindowPosition(150, 150);
    glutCreateWindow("Aplicatie Poligon si Puncte");
    init();    glutDisplayFunc(desen);    glutReshapeFunc(reshape);
    glutMainLoop();
    //return 0;
}

```

**Aplicatie 6**

```

// An2_2022.cpp : This file contains the 'main' function. Program execution begins
and ends there.
//

#include <gl/freeglut.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(1.0, 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
    glLineWidth(5);
    //glPointSize(40.0);

    glBegin(GL_LINE_LOOP); //initializare desen poligon
    glColor3f(1.0, 1.0, 0.0); //culoarea de desenare
    glVertex2f(200.0, 200.0); //stabilire coordonate triunghi
    glVertex2f(400.0, 0.0); //stabilire coordonate triunghi
    glVertex2f(400.0, 400.0); //stabilire coordonate triunghi
    glEnd(); //sfarsit 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 fereastra
{
    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 fereastra principala fereastra de
    afisare
    glutCreateWindow("TRIUNGHIEURI");
    init();
    glutDisplayFunc(display); // se reimprospatare continua continutul ferestrei
    glutReshapeFunc(reshape); // functia redesenare
    glutMainLoop(); // bucla de procesare a evenimentelor
    return 0;
}

```

#### Aplicatie folosind cu functie de rotatie

```

#include <GL/glut.h>
#include <stdlib.h>

#include <iostream>
#include <gl/freeglut.h>
void roteste_Y(int p_grade)
{
    glRotatef(p_grade, 0.0, 1.0, .0);
    glutPostRedisplay();
}
void roteste_X(int p_grade)
{
    glRotatef(p_grade, 0., 1., .0);
    glutPostRedisplay();
}
void OnKeyPress(unsigned char key, int x, int y)
{
    if (key == 27)
        exit(0);
    switch (key)
    {
    case 'q':
    case 'Q':
        roteste_Y(3);
        break;
    case 'w':
    case 'W':
        roteste_Y(-3);
        break;
    case 'a':
    case 'A':
        roteste_X(3);
        break;
    }
}

```

```

        case 's':
        case 'S':
            roteste_X(-3);
            break;
    }
}
void OnMouseClick(int button, int state, int x, int y)
{
    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
    {
        rotete_Y(20);
    }
    if (button == GLUT_RIGHT_BUTTON && state == GLUT_DOWN)
    {
        rotete_Y(-20);
    }
}

void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    int l = 10;

    for (double i = 0; i <= l; i++) {
        glBegin(GL_LINE_LOOP);
        glColor3f(1 - i / 10, i / 20, 1);
        glVertex3f(1 - i / l, 0, 0);
        glVertex3f(0, 1 - i / l, 0);
        glVertex3f(-(1 - i / l), 0, 0);
        glVertex3f(0, -(1 - i / l), 0);
        glEnd();
    }

    glFlush();
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB); // se specifica modelul de culoare
    al ferestre: un singur buffer si culoare RGB
    glutCreateWindow("Curs 2022");
    glutKeyboardFunc(OnKeyPress);
    glutMouseFunc(OnMouseClick);

    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}

```

#### **4. Realizarea desenelor 2D / 3D folosind OpenGL**

*Obiective generale:*

- Utilizarea functiilor de lucru cu ferestre in OpenGL
- Gestionarea utilizarii 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 ferestra 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 distrugere 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.

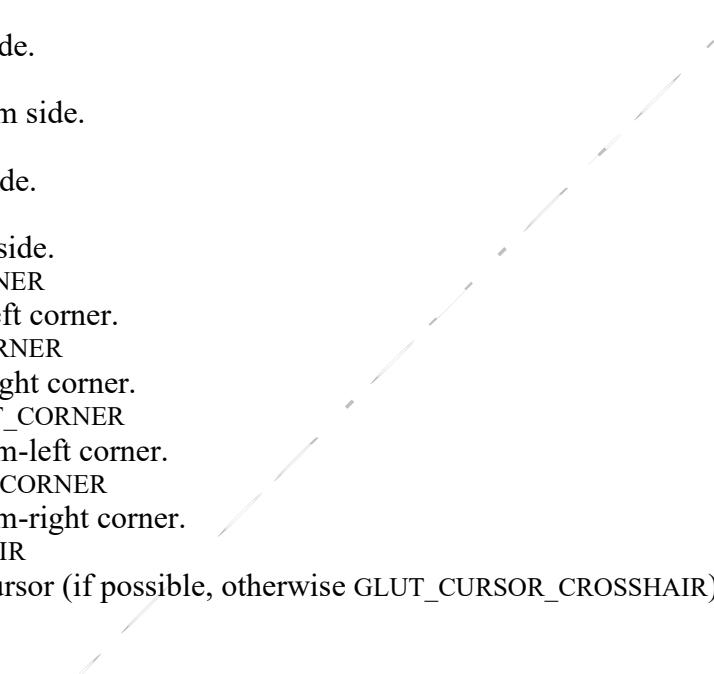
**GLUT\_CURSOR\_CYCLE**

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-left 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 glutChangeToMenuItem](#)
- [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 identificatorul corespunzător, de tip întreg. Identificatorii meniurilor încep de la valoarea 1. Valorile lor sunt separate de identificatorii 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 transmisa 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.

## Stergerea 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 ștersă sunt renumerotate.

## Distrugerea unui meniu

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

Funcția distrugere meniul specificat prin parametru.

Observatie: dacă meniul distrus este cel curent, atunci meniul curent fi devine 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 meniu curent sau meniul curent a fost distrus.

## Atașare / detașare meniu 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.**

Aplicații rezolvate:

```
#include <iostream>  
#include <gl/freeglut.h>
```

```
void init()  
{  
    glClearColor(0.0, 0.0, 0.0, 0.0);  
    glPointSize(40.0);  
    glShadeModel(GL_FLAT);
```

```

}

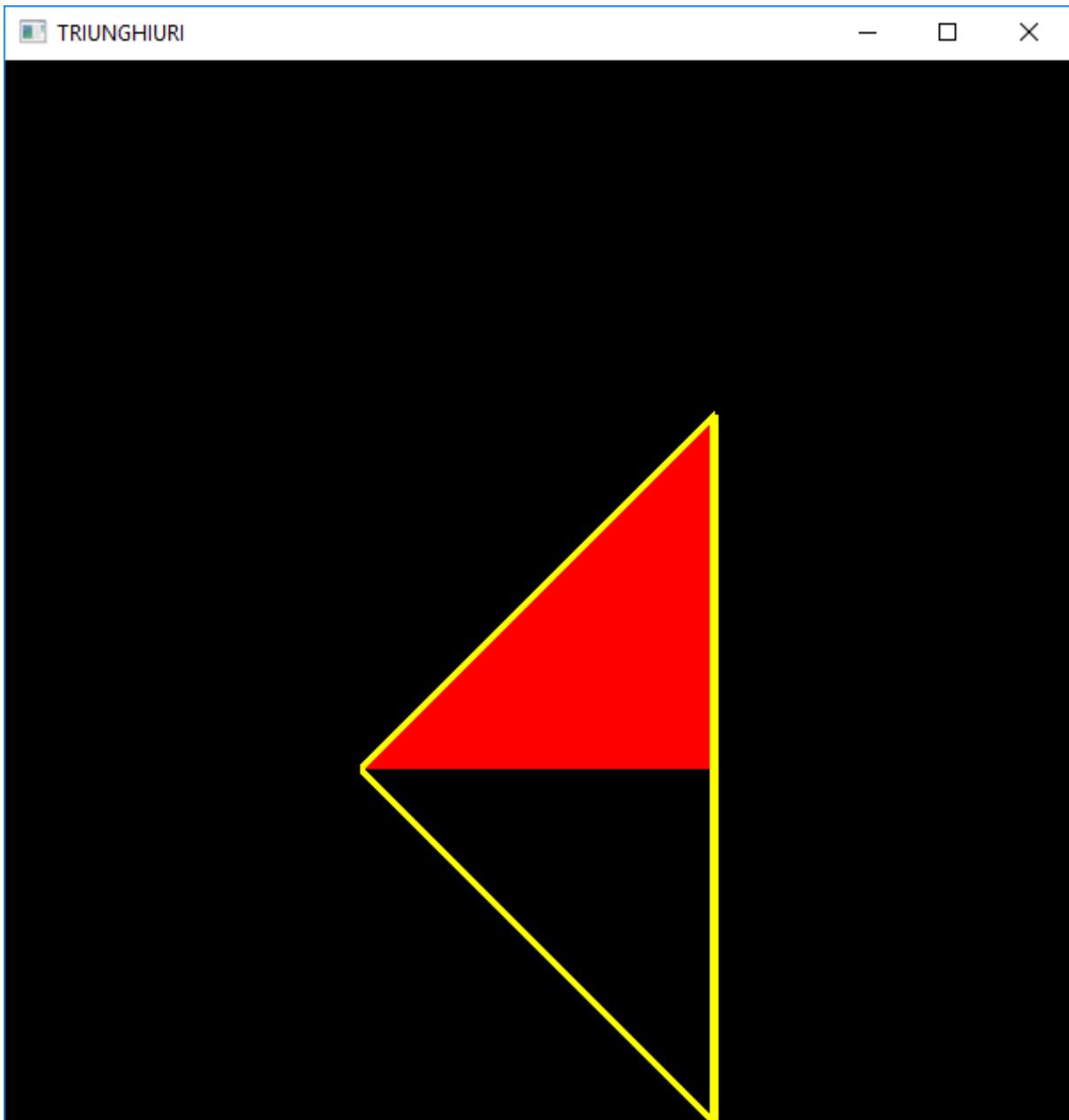
void desen()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0.0, 1.0, 0.0);
    glBegin(GL_POLYGON); //initializare desen poligon
    glVertex2f(0.0, 0.0); //stabilire coordonate triunghi
    glVertex2f(200., 0.);
    glVertex2f(200.0, 200.0); //stabilire coordonate triunghi
    glVertex2f(0.0, 200.0); //stabilire coordonate triunghi
    glEnd(); //sfisit desenare
        //executare functie
    glFlush();

    glColor3f(1.0, 1.0, 0.0);
    glBegin(GL_POINTS);
    glVertex2i(100, 300);
    glVertex2i(200, 300);
    glVertex2i(200, 400);
    glColor3f(1.0, 0.0, 1.0);
    glVertex2i(20, 20);
    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); //specificaare 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, 500);
    glutInitWindowPosition(150, 150);
    glutCreateWindow("Aplicatia 1");
    init();    glutDisplayFunc(desen);    glutReshapeFunc(reshape);
    glutMainLoop();
    //return 0;
}

```



```
#include <gl/freeglut.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(1.0, 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(); //sfarsit desenare
    glFlush(); //executare functie
    glLineWidth(5);
//glPointSize(40.0);
```

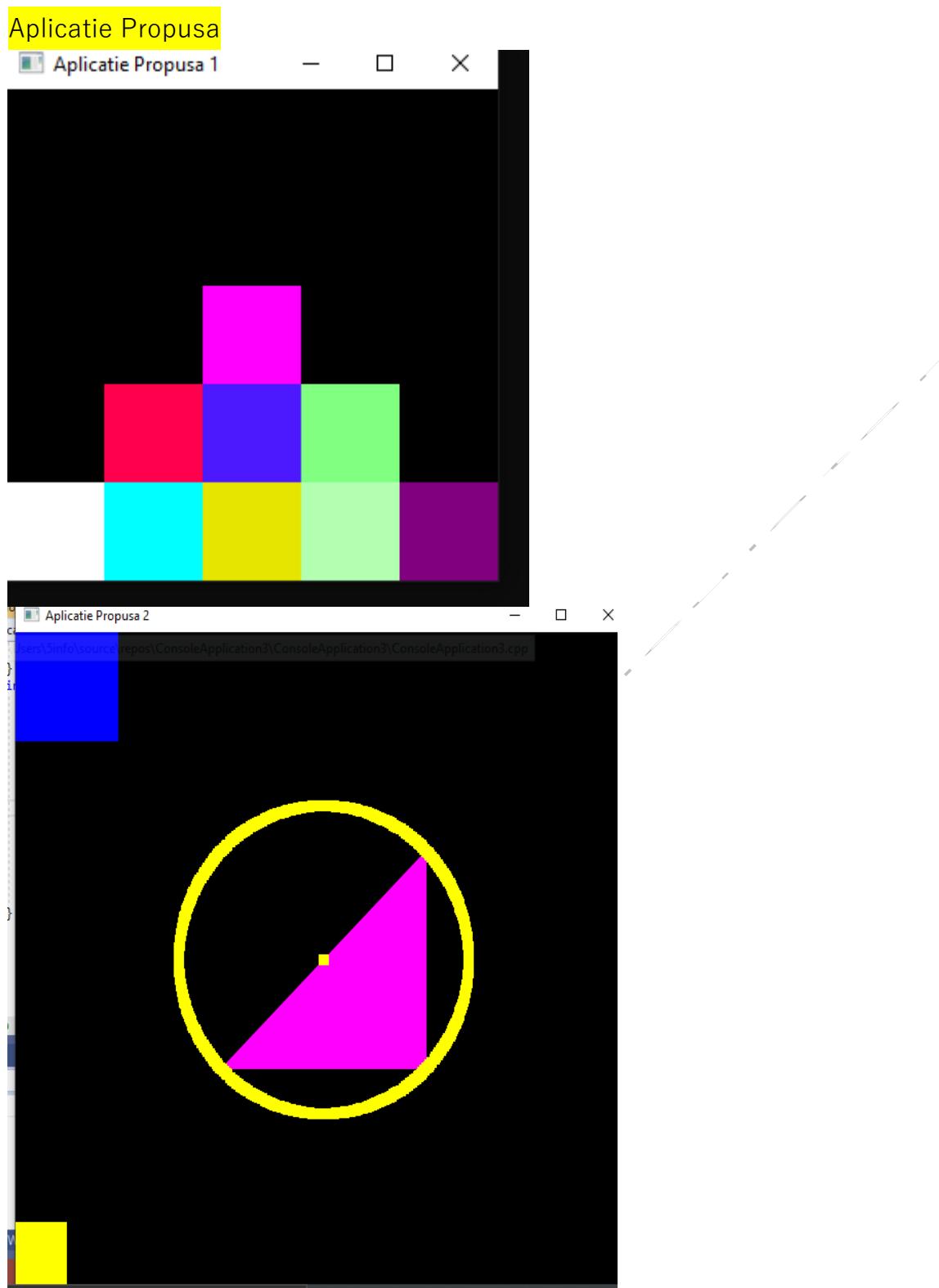
```

glBegin(GL_LINE_LOOP); //initializare desen poligon
glColor3f(1.0, 1.0, 0.0); //culoarea de desenare
glVertex2f(200.0, 200.0); //stabilire coordonate triunghi
glVertex2f(400.0, 0.0); //stabilire coordonate triunghi
glVertex2f(400.0, 400.0); //stabilire coordonate triunghi
glEnd(); //sfarsit 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 fereastra
{
    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 fereastra principala fereastra de
afisare
    glutCreateWindow("TRIUNGHIURI");
    init();
    glutDisplayFunc(display); //se reimprospateaza continutul ferestrei
    glutReshapeFunc(reshape); //functia redesenare
    glutMainLoop(); //bucla de procesare a evenimentelor
    return 0;
}

```



```

#include <GL/glut.h>

#include <stdlib.h>

#include <iostream>
#include <gl/freeglut.h>
void roteste_Y(int p_grade)
{
    glRotatef(p_grade, 0.0, 1.0, .0);
    glutPostRedisplay();
}
void roteste_X(int p_grade)
{
    glRotatef(p_grade, 0., 1., .0);
    glutPostRedisplay();
}
void OnKeyPress(unsigned char key, int x, int y)
{
    if (key == 27)
        exit(0);
    switch (key)
    {
    case 'q':
    case 'Q':
        roteste_Y(3);
        break;
    case 'w':
    case 'W':
        roteste_Y(-3);
        break;
    case 'a':
    case 'A':
        roteste_X(3);
        break;
    case 's':
    case 'S':
        roteste_X(-3);
        break;
    }
}
void OnMouseClicked(int button, int state, int x, int y)
{
    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
    {
        roteste_Y(20);
    }
    if (button == GLUT_RIGHT_BUTTON && state == GLUT_DOWN)
    {
        roteste_Y(-20);
    }
}
void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);

    int l = 10;

```

```

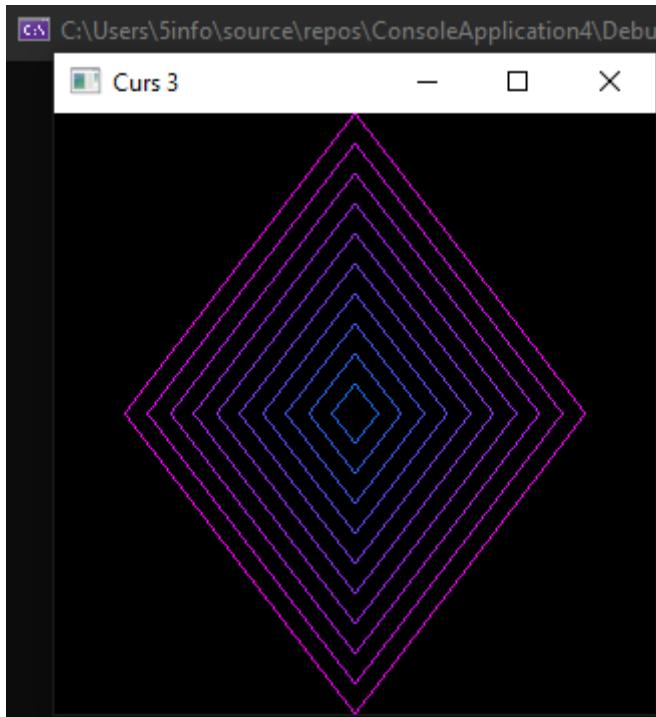
        for (double i = 0; i <= 1; i++) {
            glBegin(GL_LINE_LOOP);
            glColor3f(1 - i / 10, i / 20, 1);
            glVertex3f(1 - i / 1, 0, 0);
            glVertex3f(0, 1 - i / 1, 0);
            glVertex3f(-(1 - i / 1), 0, 0);
            glVertex3f(0, -(1 - i / 1), 0);
            glEnd();
        }

        glFlush();
    }

    int main(int argc, char** argv) {
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB); // se specifica modelul de culoare
        al ferestrei: un singur buffer si culoare RGB
        glutCreateWindow("Curs 07.11.2020");
        glutKeyboardFunc(OnKeyPress);
        glutMouseFunc(OnMouseClicked);

        glutDisplayFunc(display);
        glutMainLoop();
        return 0;
    }
}

```



### *Aplicatie propusa*

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

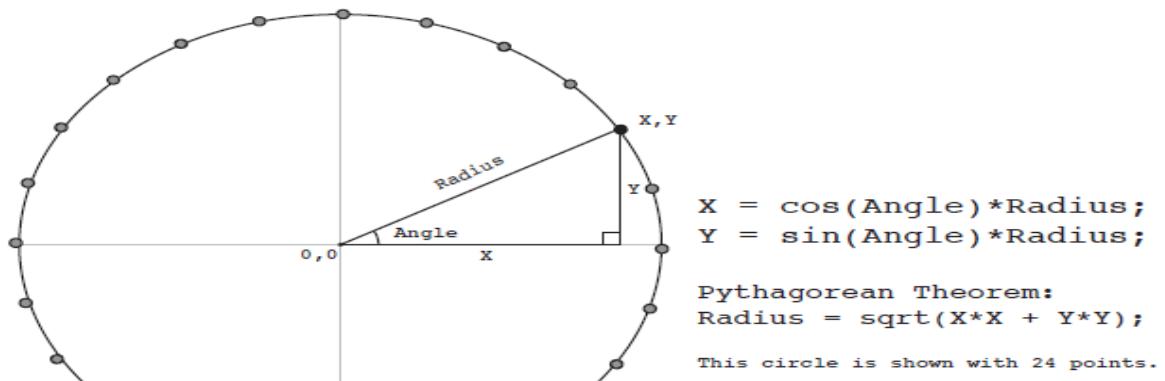
```

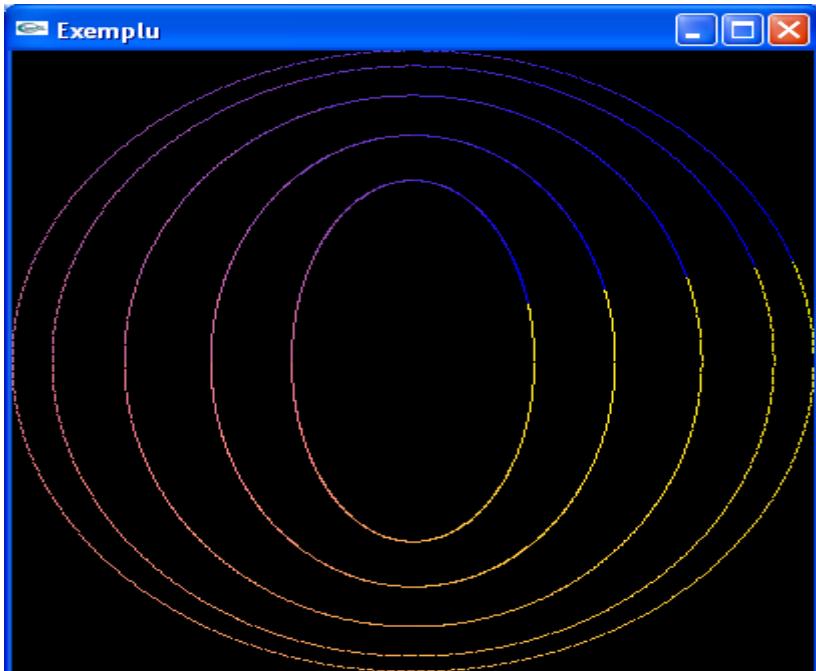
```

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.





## Meniuri

```
#include <iostream>
#include <gl/freeglut.h>
#include<math.h>
#include<stdio.h>
#include <stdlib.h>

void init()
{
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glPointSize(50.0);
    glShadeModel(GL_FLAT);
}
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 < 1000; ++i)
    {
        glVertex3f(cos(2 * 3.14159 * i / 1000.0), sin(2 * 3.14159 * i / 1000.0),
0);
    }
    glEnd();

    glFlush();
}
```

```

int meniu_1, meniu_2, meniu_3, meniu_main;

void meniu_principal(int key)
{
    if (key == 0)
    {
        exit(0);
    }
}

void callback_1(int key)
{
    switch (key)
    {
    case 0:
        printf("Cerc 1\n");
        break;
    case 1:
        printf("Cerc 2\n");
        break;
    }
}

void callback_2(int key)
{
    switch (key)
    {
    case 0:
        printf("Ati selectat dreptunghi 1\n");
        break;
    case 1:
        printf("Ati selectat dreptunghi 2\n");
        break;
    }
}

void callback_3(int key)
{
    switch (key)
    {
    case 0:
        printf("Ati selectat triunghi 1\n");
        break;
    case 1:
        printf("Ati selectat triunghi 2\n");
        break;
    }
}

GLint x = 10;
GLint y = 20;
GLint WindowWidth = 400;
GLint WindowHeight = 400;

void mouseHandler(int button, int state, int mouse_x, int mouse_y)
{
    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
    {
        x = mouse_x;
        y = WindowHeight - mouse_y;
        glColor3f(1, 0, 0);
        glBegin(GL_POINTS);
        glVertex2i(x, y);
        printf("x=%d , y=%d \n", x, y);
        glEnd();
    }
}

```

```

        glFlush();
        glClear(GL_COLOR_BUFFER_BIT);
    }
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(400, 400);
    glutInitWindowPosition(400, 100);
    glutCreateWindow("aplicatii 2022");
    init();
    glutMouseFunc(mouseHandler);

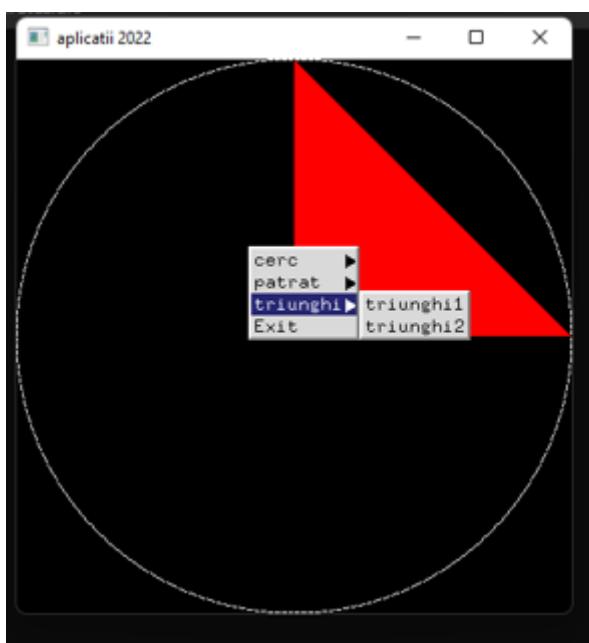
    glutDisplayFunc(display);
    meniu_1 = glutCreateMenu(callback_1);
    glutAddMenuEntry("cerc1", 0);
    glutAddMenuEntry("cerc2", 1);
    meniu_2 = glutCreateMenu(callback_2);
    glutAddMenuEntry("dreptunghi1", 0);
    glutAddMenuEntry("dreptunghi2", 1);
    meniu_3 = glutCreateMenu(callback_3);
    glutAddMenuEntry("triunghi1", 0);
    glutAddMenuEntry("triunghi2", 1);

    meniu_main = glutCreateMenu(meniu_principal);
    glutAddSubMenu("cerc", meniu_1);
    glutAddSubMenu("patrat", meniu_2);
    glutAddSubMenu("triunghi", meniu_3);
    glutAddMenuEntry("Exit", 0);
    glutAttachMenu(GLUT_RIGHT_BUTTON);

    glutMainLoop();

    return 0;
}

```



## 4.3 Afişarea obiectelor 3D predefinite

GLUT conține funcții pentru afișarea următoarelor obiecte 3D:

con	icosaedru	teapot
cub	octaedru	tetraedru
dodecaedru	sferă	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 msecs, 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)

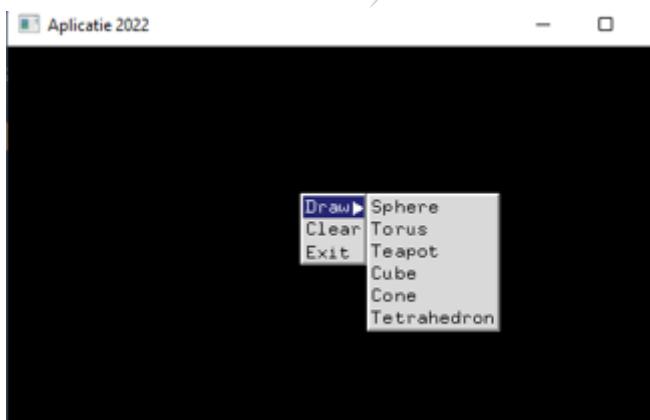
#### Font Rendering

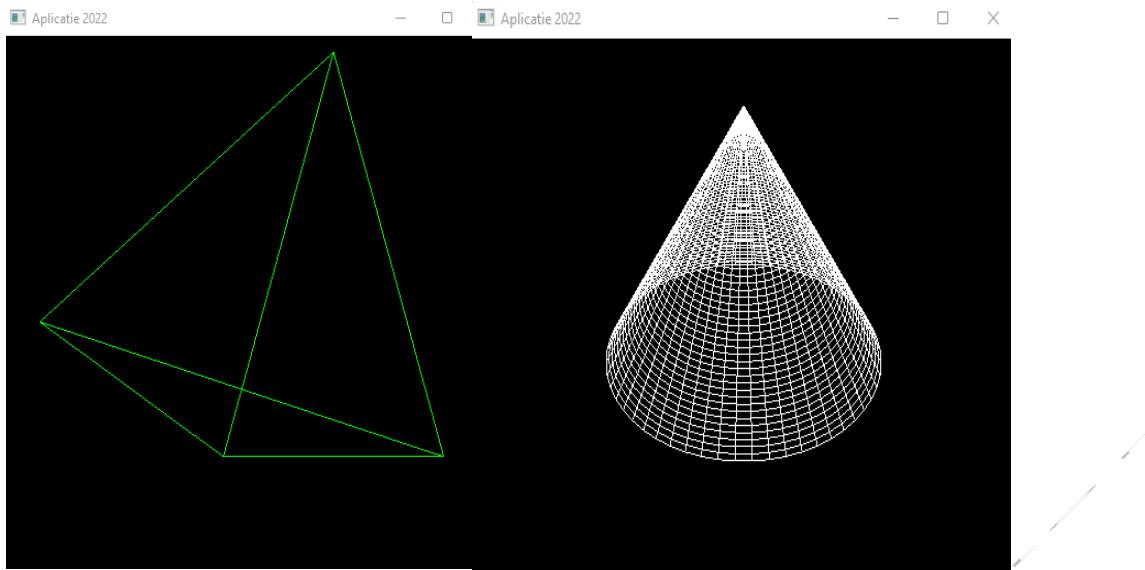
- void **glutBitmapCharacter** (void \*font, int character)
- int **glutBitmapWidth** (GLUTbitmapFont font, int character)
- void **glutStrokeCharacter** (void \*font, int character)
- int **glutStrokeWidth** (GLUTstrokeFont font, int character)

#### 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)

Aplicatie cu obiecte 3D si meniuri





```
#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;
    }
}
```

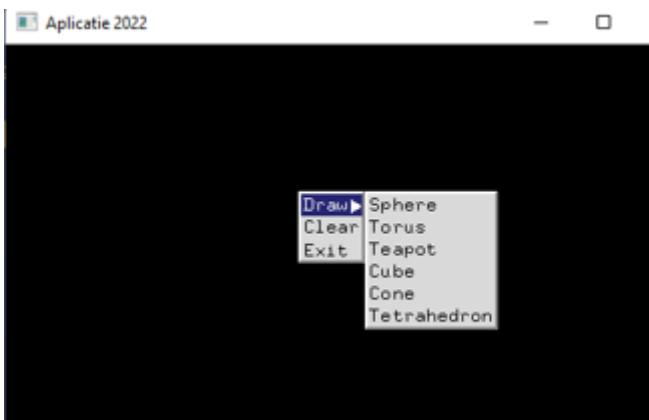
```

    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;
}

```



```
#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_RGB | 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**(*x, y, 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:

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

**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); // Stergerea 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 liniiilor ce vor fi afișate; *width* trebuie să fie mai mare ca 0.0, iar valoarea implicită este 1.0.

puncte - Microsoft Visual C++ 10.223.2.15

File Edit View Insert Project Build Tools Window Help

(Globals) (All global members) main

**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);

    glColor3f (1.0, 0.0, 0.0);
    glBegin(GL_POINTS);
    glVertex2i(340,60);
    glColor3f (0.5, 0.0, 0.0);
    glVertex2i(380,100);
    glColor3f (0.0, 0.0, 0.5);
    glVertex2i(380,60);
    glColor3f (1.0, 0.5, 0.0);
    glVertex2i(300,100);
    glColor3f (1.0, 0.0, 0.5);
    glVertex2i(300,60);
    glColor3f (0.0, 1.0, 1.0);
    glVertex2i(340,100);
    glColor3f (1.0, 0.0, 0.7);
    glVertex2i(340,20);
    glColor3f (0.5, 0.5, 0.5);
    glVertex2i(300,20);
    glColor3f (1.0, 1.0, 0.7);
    glVertex2i(380,20);
    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);
}

void main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize (400, 400);
    glutInitWindowPosition (880,600);
    glutCreateWindow ("Puncte");
    init ();
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutMainLoop();
}

```

ClassView FileView

----- Configuration: puncte - Win32 Debug -----

Linking ...

puncte.exe - 0 error(s), 0 warning(s)

Build Debug Find in Files 1 Find in Files 2 Results

Ln 42, Col 1 REC COL OVR READ

Start Fw: Grafica - Outlook W... OpenGL 1.1 Reference: ... Downloads New Folder puncte - Microsoft Vis... de cbit.docx - Microsoft ... 3:03 PM



The screenshot shows the Microsoft Visual Studio interface. The main window displays the source code for 'triunghi.cpp'. The code includes OpenGL initialization, vertex definitions, and window creation logic. Below the code editor is the 'Output' window, which shows the compilation process: 'Compiling triunghi.cpp', 'Linking...', and 'triunghi.exe - 0 error(s), 0 warning(s)'. The status bar at the bottom indicates 'Ln 48, Col 3'.

```

trinighi - Microsoft Visual C++
File Edit View Insert Project Build Tools Window Help
(Globals) (All global members) main
triunghi.cpp *
#include <GL/glut.h>
void init()
{
    glClearColor (0.0, 0.0, 0.0, 0.0);
    glShadeModel (GL_FLAT);
}
void display()
{
    glClear (GL_COLOR_BUFFER_BIT);

    glBegin(GL_POLYGON);
    glColor3f (1.0, 1.0, 1.0);
    glVertex2f(200.0,200.0);
    glVertex2f(400.0,200.0);
    glVertex2f(400.0,400.0);
    glEnd();
    glFlush ();
    glColor3f (1.0, 1.0, 0.0);
    glBegin(GL_POLYGON);
    glVertex2f(200.0,400.0);
    glVertex2f(400.0,400.0);
    glVertex2f(200.0,200.0);
    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);
}

void main(int argc, char** argv)
{
    glutInit(&argc, &argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize (600, 600);
    glutInitWindowPosition (100,100);
    glutCreateWindow ("Triunghi");
    init ();
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutMainLoop();
}

```

## 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 să fie apelată înaintea oricărei alte comenzi GLUT.

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

Folosirea modelului RGBA (culoarea se specifică prin componente sale roșu, verde, albastru și transparență 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 apăsat 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 explice a aplicației, prin apelul funcției **glutPostRedisplay()**.

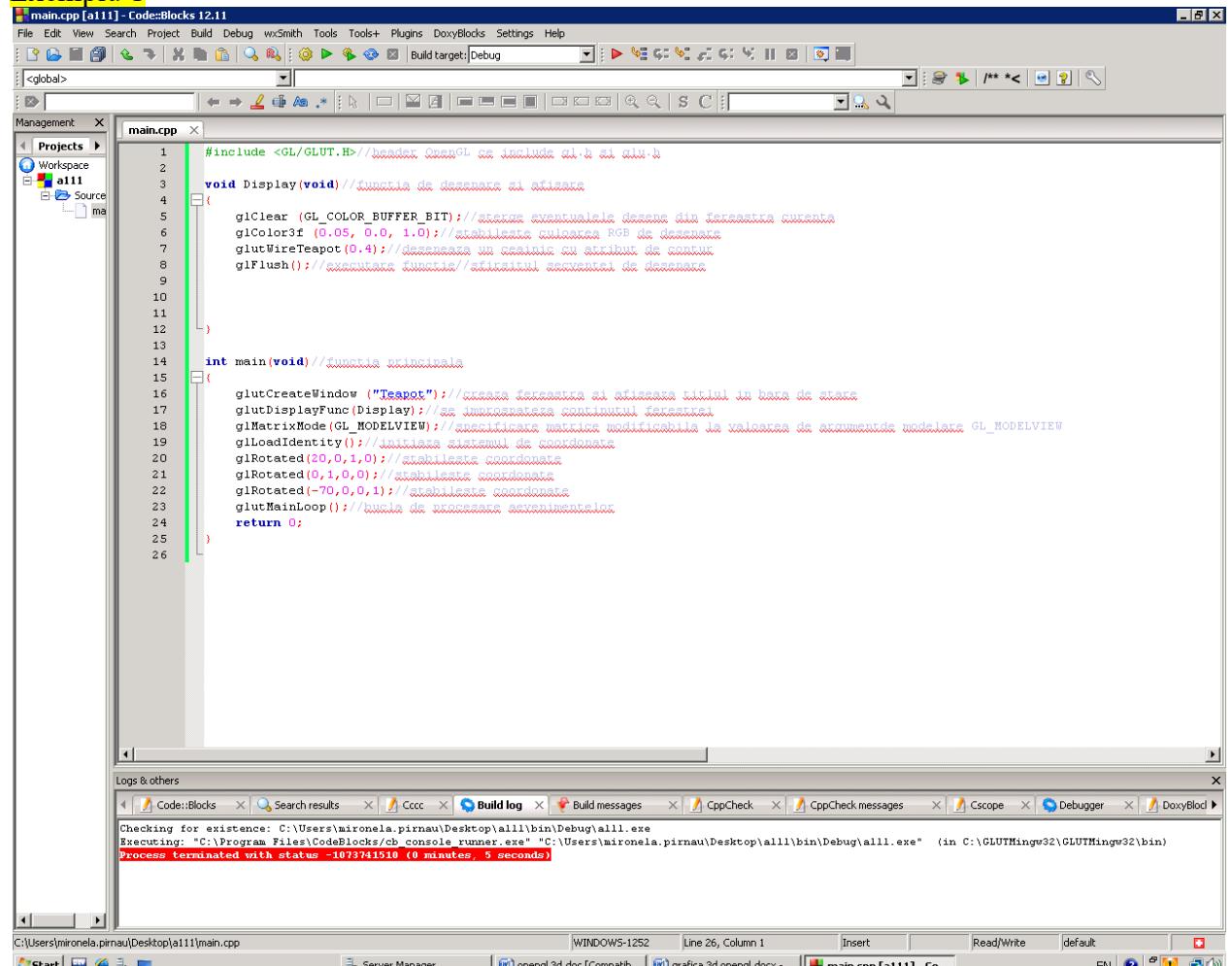
#### Bucla de execuție a aplicației

```
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 propose

### Exemplu 1



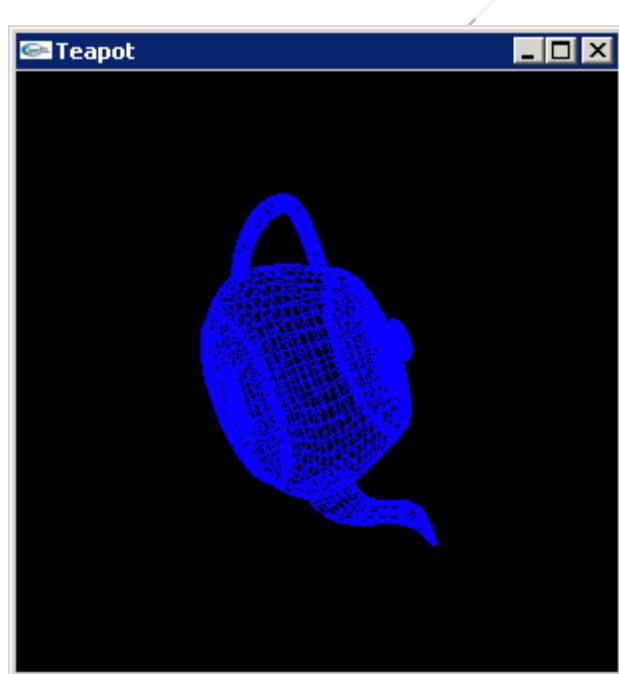
The screenshot shows the Code::Blocks IDE interface. The main window displays the source code for `main.cpp` in the `a111` project. The code uses OpenGL functions to render a wireframe teapot. The `Display` function sets up the drawing environment, while the `main` function initializes the window and starts the rendering loop. Below the code editor is the `Logs & others` panel, which shows the build log and output from the console runner. The log indicates the program was executed successfully. At the bottom, the taskbar shows the application is running under the name `Teapot`.  
  
Code Snippet:

```
#include <GL/glut.h> // header OpenGL ce include gl.h si glu.h

void Display(void) // functie de desenare si afisare
{
    glClear(GL_COLOR_BUFFER_BIT); // sterge ecranul cu valoarea din desenarea curenta
    glColor3f(0.05, 0.0, 1.0); // stabileste culorarea RGB de desenata
    glutWireTeapot(0.4); // desenaza un casnic cu atribut de contur
    glFlush(); // executare functie // afisarea actualului de desenat

}

int main(void) // functia principala
{
    glutCreateWindow("Teapot"); // creaza ferestra si afisarea tuturor in baza de date
    glutDisplayFunc(Display); // se impregneaza continutul ferestrei
    glMatrixMode(GL_MODELVIEW); // specificare matricea modificabila la valoarea de argumente modelare GL_MODELVIEW
    glLoadIdentity(); // initializeaza sistemul de coordonate
    glRotated(20, 0, 1, 0); // rotirea primelor
    glRotated(0, 1, 0, 0); // rotirea a doua
    glRotated(-70, 0, 0, 1); // rotirea a treia
    glutMainLoop(); // bucla de procesare evenimentelor
    return 0;
}
```



## Exemplu 2.

Screenshot of the Code::Blocks IDE showing the main.cpp file and its output window, along with a screenshot of the OpenGL window displaying a wireframe teapot.

The main.cpp file contains the following code:

```
#include <GL/glut.h> // header OpenGL care includea si.h si glu.h
void Display(void) // functie de desenare si afisare
{
    glClear (GL_COLOR_BUFFER_BIT); // sterge ecranul actualizand desene din ferestra curenta
    glMatrixMode(GL_MODELVIEW); // seteaza matricea modelului la valoarea actualizarii de modelare GL_MODELVIEW
    glLoadIdentity(); // initializa sistemul de coordonate // transfera matricea identitate
    glPushMatrix();
    glRotated(45,1,1,1); // rotire sub un anumit unghi si stabilire coordonate de afisare in ferestra
    glutWireTeapot(0.3); // desenarea unei esamale din un obiect de comun
    glPopMatrix();
    glPushMatrix();
    glTranslated(0.5,0.5,0); // stabilire coordonate de afisare a esamului in ferestra
    glutWireTeapot(0.3); // permite desenarea unui esanic doar cu atribut de contur
    glPopMatrix();
    glPushMatrix();
    glScaled(0.5,1,0); // stabilire coordonate de afisare in ferestra (culoarea esamului)
    glTranslated(-1,0.5,0); // stabilire coordonate de afisare a esamului in ferestra
    glutWireTeapot(0.4); // permite desenarea unui esanic doar cu atribut de contur
    glPopMatrix();
    glFlush(); // executare functie // aceea functie nu se executa pana acum va fi executata
}
int main(void) // creare functie de afisare
{
    glutCreateWindow ("Teapot"); // creare ferestra si afisarea initial "Teapot" in baza de date
    glutDisplayFunc(Display); // ac calibruarea continutului ferestrei
    glutMainLoop(); // bucla de procesare a evenimentelor
    return 0;
}
```

The Logs & others window shows the build process:

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

The operating system taskbar at the bottom shows the application is running.

The OpenGL window titled "Teapot" displays a wireframe rendering of a teapot and a cup.

### 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

<global> main(void) : int

Management X

Projects X

Workspace a111 Source ma

\*main.cpp X

```

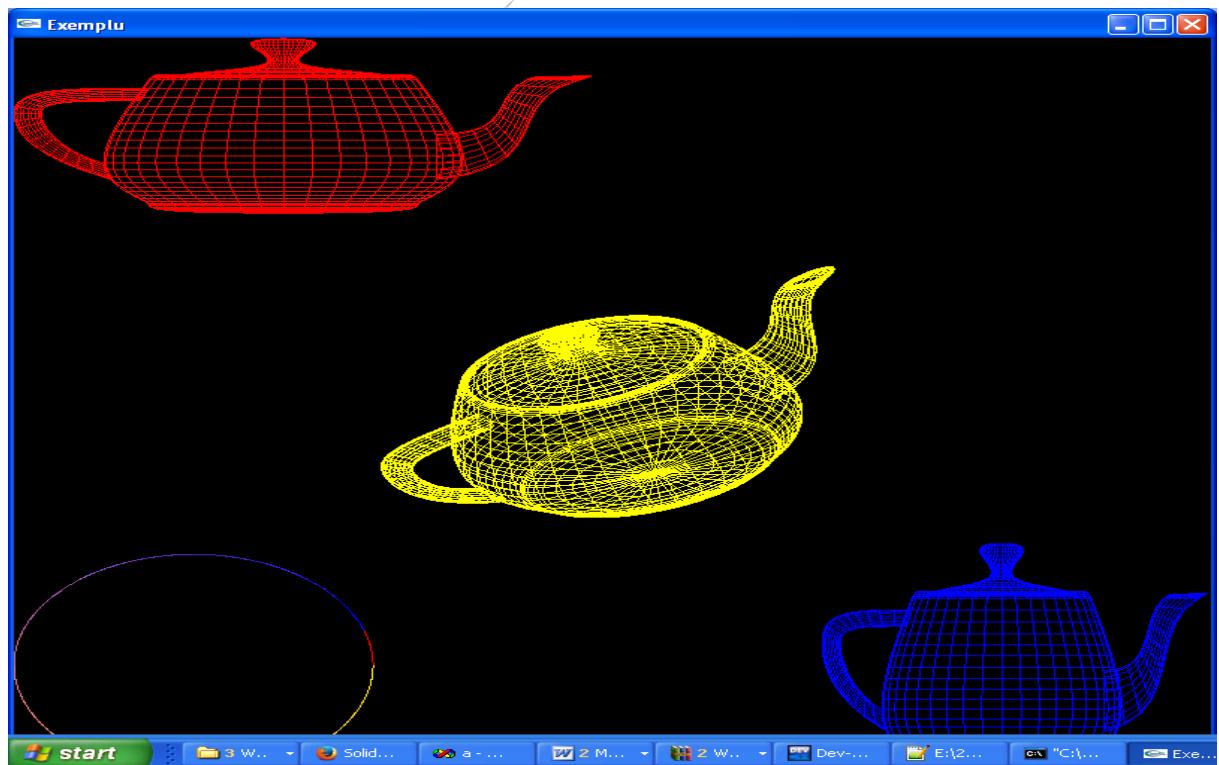
1 #include <stdio.h>
2 #include <math.h>
3 #include <gl/glut.h>
4 void Display(void)
5 {
6     glClear(GL_COLOR_BUFFER_BIT);
7     glMatrixMode(GL_MODELVIEW);
8     glLoadIdentity();
9     glPushMatrix(); glColor3f(1,1,0);
10    glRotated(45,1,1,1); //rotate the teapot
11    glutWireTeapot(0.3); //create 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 }
35 int main(void)
36 {
37     glutInitWindowSize(800,800);
38     glutCreateWindow("Exemplu");
39     glutDisplayFunc(Display);
40     glutMainLoop(); return 0;
}

```

Logs & others

C:\Users\mironela.pinaul\Desktop\a111\main.cpp

Start Server Manager \*main.cpp [a11...]



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);

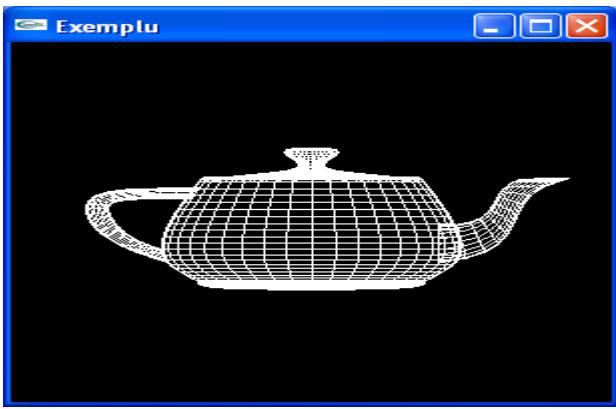
    glFlush();
}

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

    glMatrixMode(GL_MODELVIEW);

    glutMainLoop();
    return 0;
}

```



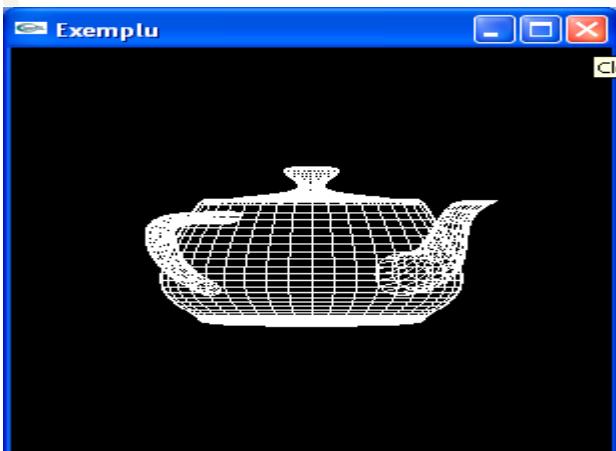
### Aplicatia 2

```
#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);
    glRotated(45,1,1,1);
    glutMainLoop();
}
```

## 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();

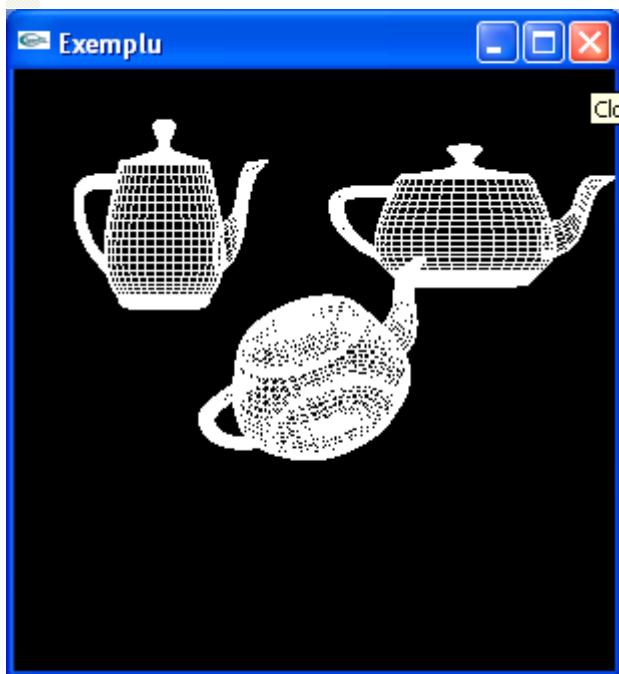
    glPushMatrix();
    glTranslated(0.5, 0.5, 0);
    glutWireTeapot(0.3);
    glPopMatrix();

    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

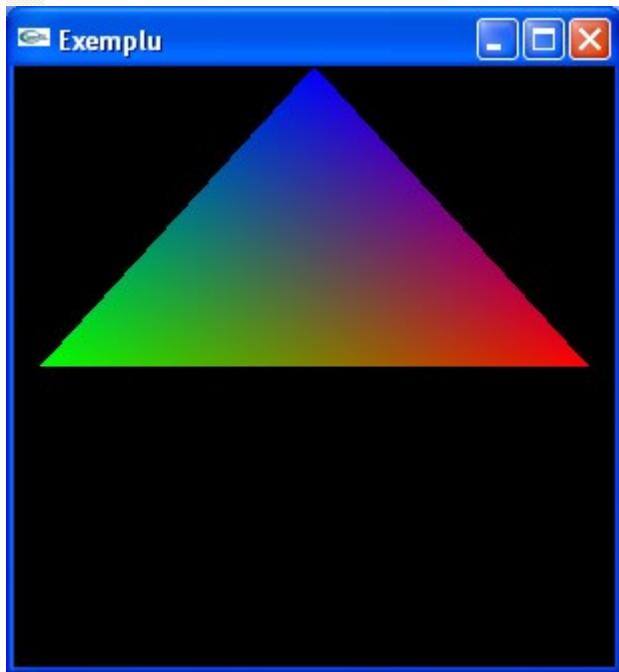
```
#include <stdio.h>
#include <GL/GLUT.H>
int GAngle=0;
void DrawPlanetMoon(float radius,int angle)
{
    glPushMatrix();
    glRotated(angle,0,1,0);
    glPushMatrix();
    glTranslated(radius,0,0);
    glutWireSphere(radius,10,10);
    glPopMatrix();
    glPushMatrix();
    glTranslated(-2*radius,0,0);
    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);
    glEnable(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;
}
```

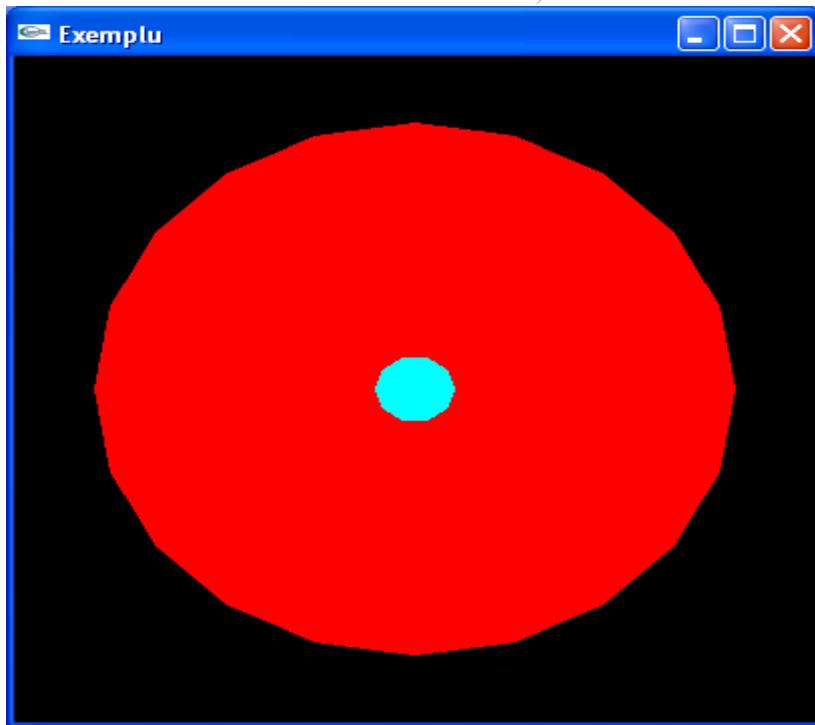
## Aplicatia 14

```

#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 razal = 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(razal * cos(i * doiPi / sectiuni),
                        razal* 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 15

```
#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();
}
```

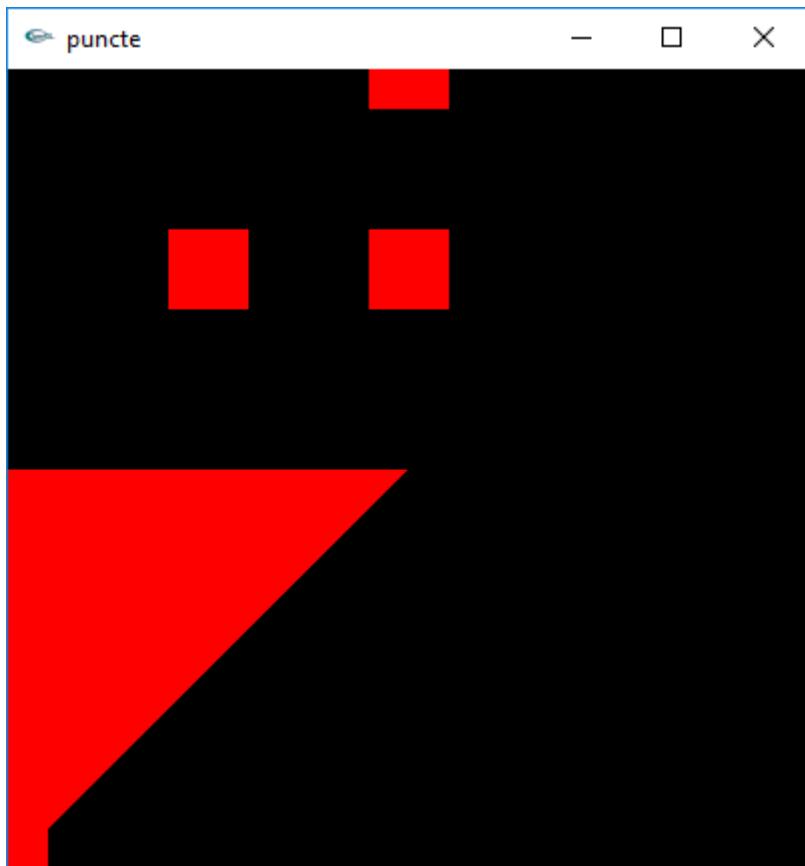


## 4.6. Probleme propuse

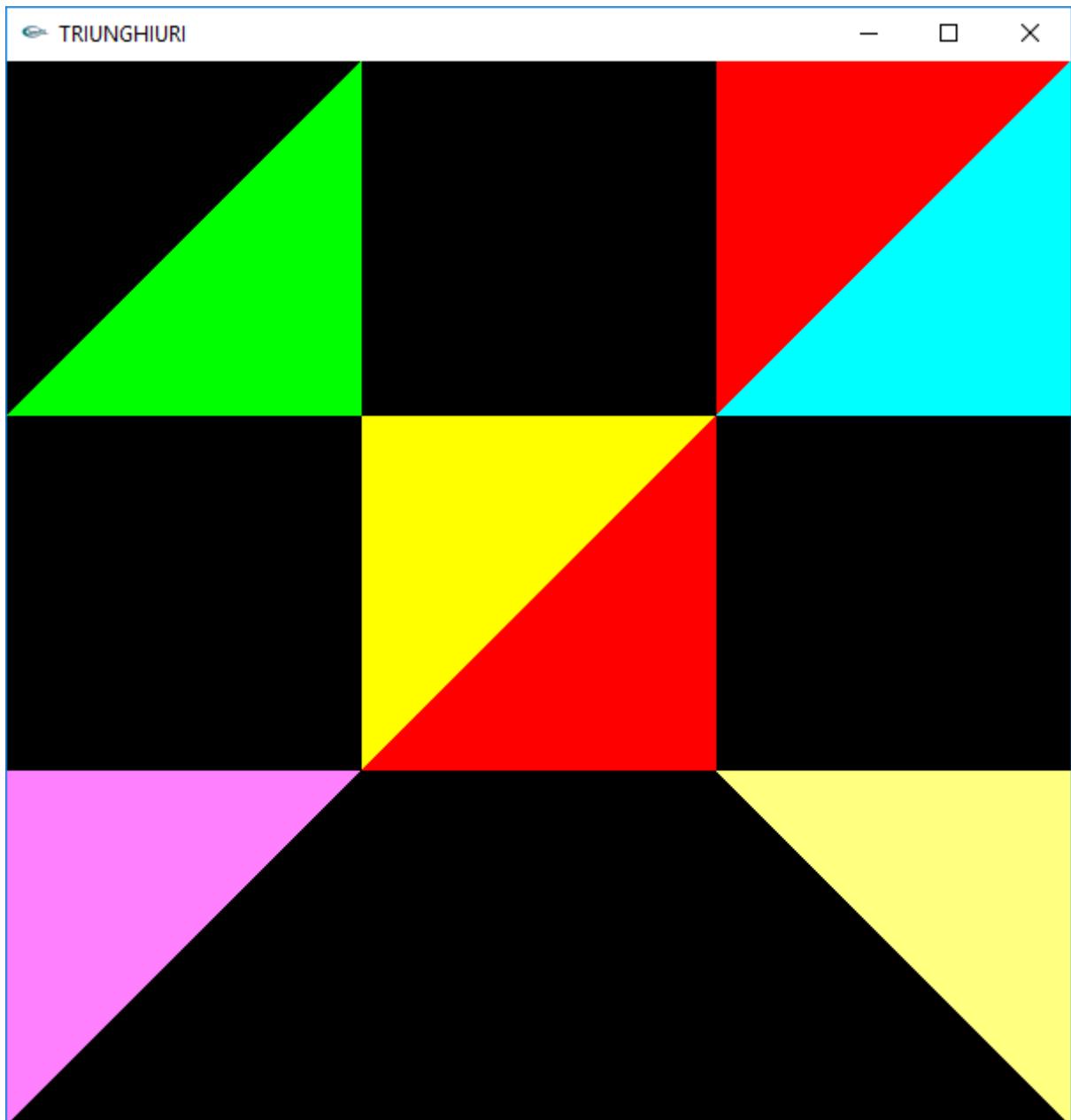
Probleme Propuse

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

Aplicatia 1



Aplicatia 2



# 5. Unity Game Engine

## OBIECTIVE GENERALE

- Cunoasterea interfetei grafice pentru Unity Game Engine;
- Intelegerea principalelor tutoriale de la <https://unity3d.com/learn>
- Intelegerea celor mai utilize functii din clasa Monobehaviour

*Keywords:* Unity 3D, clasa Monobehaviour

*Cuprins:*

- 5.1 Unity Game Engine;
- 5.2 Functii 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

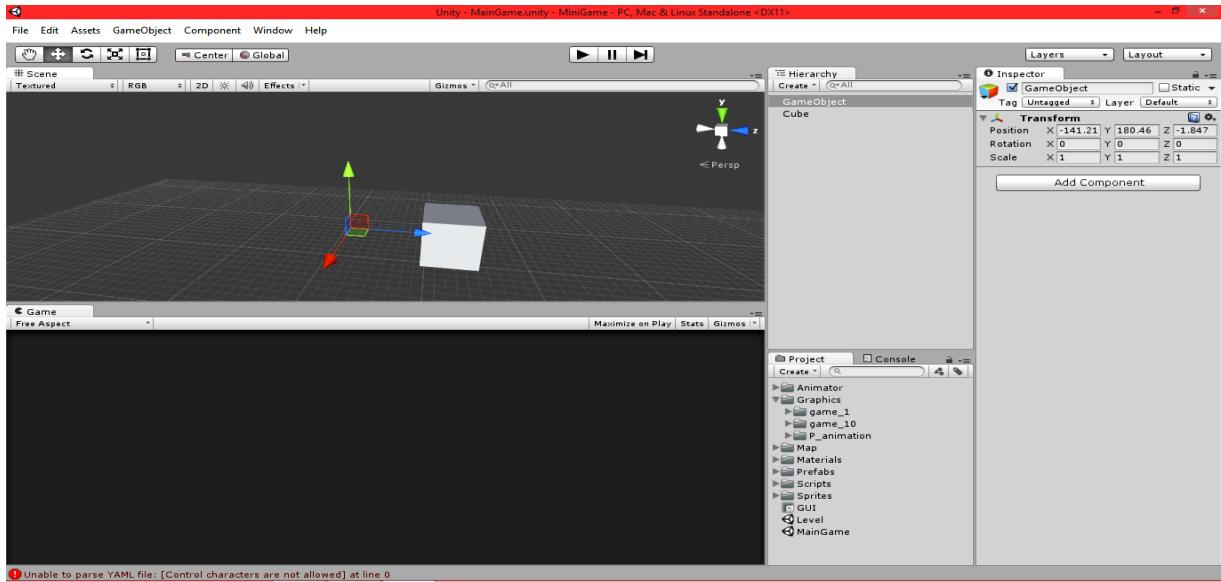
Unity este un game engine cu ajutorul caruia putem realiza jocuri 2D cat si 3D pentru PC, console, telefoane mobile si website. Acesta a fost lansat in 2005 si este scris in C,C++ si C#. Initial acest game engine a fost lansat doar pentru Sistemele de operare Mac, iar in versiunea lansata initial continea shadere orientate OpenGL. In urmatoarele versiuni de Unity 1.x s-a inclus suport pentru rularea jocurilor pe sistemul de operare Windows.

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

Unity este un game engine care a evoluat mult de-a lungul anilor, reusind sa isi extinda aria ca dispozitive target. A devenit mult mai usor de utilizat, oferind programatorului cat si amatorului programator posibilitatea de a creea jocuri 2D cat si 3D cu un bagaj mic de cunostinte. Cu timpul a devenit tot mai popular, tot mai cunoscut prin jocurile create si s-a format o intreaga comunitate de developeri amatori si profesionisti care pot pune la dispozitie raspunsuri la intrebarile utilizatorilor. De asemenea pe site-ul oficial ([www.unity3d.com](http://www.unity3d.com)) se gasesc o multime de tutoriale inclusive live, cu scopul de a demonstra cat de practic si puternic poate fi acest game engine, iar ca un plus, noua versiune lansata in acest an Unity 5, este gratuita. Unity suporta 3 limbaje diferite ( C#, JavaScript si 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:

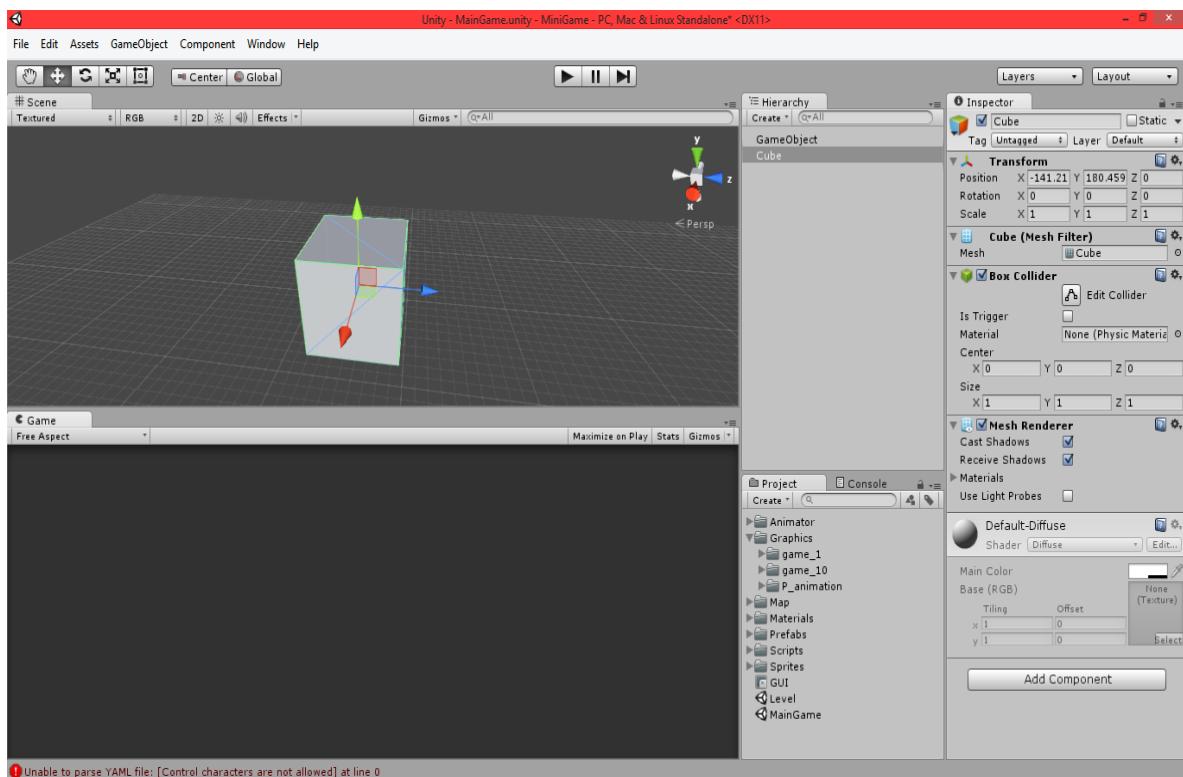
1. **Scena** - reprezinta toate componentelete pe care le ai in joc (fiecare scena contine o camera);
2. **Game** - reprezinta fereastra cu jocul si este ceea ce camera ta vede in scena;
3. **Ierarhie** – contine structurate obiectele din scena;
4. **Proiect** - reprezinta proiectul utilizatorului si 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 doarit.

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 dea a iti porta jocul pe o platforma dorita de tine, iti pune la dispozitie o multime de pachete ce acopera totul incepand de la texturi 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 comenzi de control.

**}**

**FixedUpdate() {**

// Functie care este apelata la un numar de frameuri constante, pot fi folosite pentru a adauga o forta unui obiect, la fiecare miscare a obiectului functia FixedUpdate este apelata.

**}**

**LateUpdate() {**

Functie apelata la finalul rularii instructiunilor din Update()

**}**

**Coroutine() {**

\* Aceste functii sunt folosite atunci cand ai nevoie de realizarea unei actiuni la un anumit moment din joc, coroutines sunt apelate cu StartCoroutine(functie) si este de forma :

IEnumerator functie() {

Instructiuni

Yield return new WaitForSeconds(1.4f);

Instructiuni

```

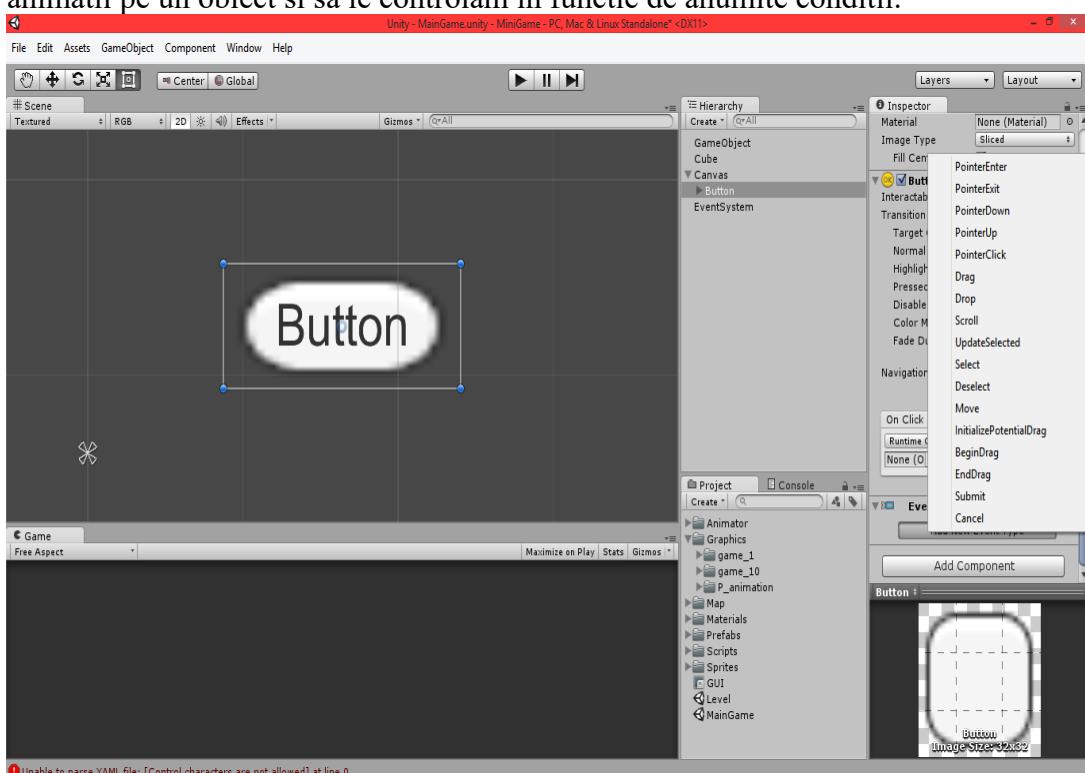
...
}
astfel ai controlul asupra derularii unor actiuni.
*/
}

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.

Incepand cu Unity 4.6 dezvoltatorii de jocuri au accesul la un UI Tool care nu necesita scrierea de foarte mult cod. Creearea unei interfete cu care utilizatorul sa interactioneze este foarte usoara, deoarece avem la dispozitie un Animator Controller care ne ajuta sa mentionam anumite animatii pe un obiect si sa le controlam in functie de anumite conditii.



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

## Assets/Unity Assets Store

Unity asset store găzduiește mii de active gratuite sau la prețuri accesibile, care economisesc timp și efort critic pentru creatorii Unity.

## Ce este Unity Assets Store

Unity Asset Store este o bibliotecă în creștere de active. Atât Unity Technologies, cât și membrii comunității creează aceste active și le publică în magazin. Există diferite tipuri de

active în magazin, de la texturi, animații și modele la exemple complete de proiecte, tutoriale și extensii de editor. Există o combinație de active comerciale gratuite și accesibile pe care le puteți descărca direct în proiectul dumneavoastră Unity. Puteți deveni editor în magazinul de active și puteți vinde creațiile dvs. Unity.

Puteți vizita Unity Asset Store în câteva moduri: fie vizitați site - ul web, fie prin intermediul Unity Game Engine.

## Audio

Designul sunetului este o componentă importantă pentru a crea o experiență de joc captivantă și încărcată emoțional. Găsirea muzicii perfecte pentru a merge cu jocul nu este ușoară, dar nu trebuie să o facem de la zero. Unity are o bibliotecă de fișiere de sunet pe care le puteți utiliza pentru a îmbogăți experiența de utilizare a proiectului. Unity Asset Store are o varietate de materiale audio gratuite și accesibile, inclusiv ambientale, muzicale și efecte sonore, astfel încât este ușor să găsiți exact ceea ce căutăm.

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

In 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 librariilor JavaScript nu vor merge dacă sunt doar copiate în Unity.

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

Deasemenea este destul de problematic să specifici mereu când cauti rezolvarea unei probleme dacă te referi la limbajul real "JavaScript" sau "UnityScript".

In comparație cu JavaScript, UnityScript folosește clase. Deasemenea în UnityScript o clasa odată definită este mai mult sau mai puțin fixă pe durata rularii 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>

Ca motor de joc, Unity este capabil să ofere multe dintre cele mai importante funcții încorporate care fac ca un joc să funcționeze. Astă înseamnă lucruri precum fizica, redarea 3D și detectarea coliziunilor. Unity oferă utilizatorilor posibilitatea de a crea jocuri și experiențe atât în 2D, cât și în 3D, iar motorul oferă un API de scriptare principal în C #, atât pentru editorul Unity sub formă de plugin-uri, cât și pentru jocuri în sine, precum și funcționalitatea de drag and drop . Înainte ca C # să fie principalul limbaj de programare utilizat pentru motor, acesta folosea anterior Boo, care a fost eliminat odată cu lansarea Unity 5, și o versiune de JavaScript numită UnityScript, care a fost depreciată în august 2017, după lansare of Unity 2017.1, în favoarea C #. În cadrul jocurilor 2D, Unity permite importul de sprite și un renderer avansat 2D. În jocurile 3D, Unity permite specificarea compresiei texturilor, a mipmap-urilor și a setărilor de rezoluție pentru fiecare platformă pe care motorul de joc o acceptă și oferă suport pentru cartografiere cu bumpuri, mapare cu reflexie, mapare cu paralaxă, ocluzie ambientală a spațiului pe ecran (SSAO), dinamică umbre folosind hărți de umbre, redare textură și efecte de post-procesare pe ecran complet.

De asemenea Unity include și un „magazin de active” care este locul în care dezvoltatorii își pot încărca creațiile și le pot pune la dispoziția comunității.

Doriți un efect de foc frumos, dar nu aveți timp să construiți unul de la zero? Verificați magazinul de active și probabil veți găsi ceva. Doriți să adăugați controale de înclinare la joc fără a trece prin laboriosul proces de reglare fină a sensibilității? Probabil că deja există unul creat și pentru asta!. **Dezvoltatorul jocului este liber să se concentreze asupra a ceea ce contează: proiectarea unei experiențe unice și distractive.**

Pe lângă un motor de joc, Unity este un IDE - un mediu de dezvoltare integrat, care descrie o interfață ce vă oferă acces la toate instrumentele de care aveți nevoie pentru dezvoltare într-un singur loc. Software-ul Unity are un editor vizual care permite creatorilor să tragă și să elibereze elemente în scene și apoi să le manipuleze proprietățile.

Unity Software oferă, de asemenea, o serie de alte funcții și instrumente utile: cum ar fi posibilitatea de a naviga prin foldere în proiect sau de a crea animații printr-un instrument de cronologie.

Cu privire la codificare, Unity contine un editor alternativ. Cea mai comună opțiune

este Visual Studio de la Microsoft, Unity folosește C# pentru a gestiona codul și logica, cu o grămadă de clase și API-uri. Deoarece Unity este multiplataforma, aceasta înseamnă că este la fel de ușor să creați jocuri pentru iOS, PC, Android sau chiar console de jocuri. Unity oferă, de asemenea, suport VR excelent pentru acei dezvoltatori interesați să dezvolte pentru Oculus Rift sau HTC Vive.

In 2008, când Apple și-a lansat App Store Unity a rapid suport pentru iPhone. De câțiva ani, motorul a fost necontestat pe iPhone și a devenit bine cunoscut de către dezvoltatorii de jocuri iOS. Unity 3.0 a fost lansat în septembrie 2010, cu funcții care extind caracteristicile grafice ale motorului pentru computere desktop și console de jocuri video. În plus față de suportul pentru Android, Unity 3 a inclus integrarea instrumentului Beast Lightmap de la Illuminate Labs, redarea amânată, un editor de copac încorporat, redarea fonturilor native, cartografierea automată UV și filtrele audio, printre altele. În noiembrie 2012, Unity Technologies a livrat Unity 4.0. Această versiune a adăugat DirectX 11 și suportul Adobe Flash, noi instrumente de animație numite Mecanim și acces la previzualizarea Linux.

Facebook a integrat un kit de dezvoltare software pentru jocuri folosind motorul de joc Unity în 2013. Aceasta conținea instrumente care permiteau urmărirea campaniilor publicitare și a legăturilor profunde, în care utilizatorii erau conectați direct de la postările de pe rețelele sociale la anumite porțiuni din jocuri și partajarea ușoară a imaginilor în joc. În 2016, Facebook a dezvoltat o nouă platformă de jocuri pentru PC cu Unity. Unity a oferit suport pentru platformele de jocuri Facebook, iar dezvoltatorii Unity ar putea exporta și publica mai rapid jocuri pe Facebook.

Verge a spus despre lansarea Unity 5 din 2015: „Unity a început cu scopul de a face dezvoltarea jocului universal accesibilă. [...] Unity 5 este un pas mult așteptat către acel viitor.” Cu Unity 5, motorul a îmbunătățit iluminarea și sunetul. Prin WebGL, dezvoltatorii Unity și-au putut adăuga jocurile la browserele web compatibile, fără a fi necesare plug-in-uri pentru jucători. Unity 5.0 a oferit iluminare globală în timp real, previzualizări de mapare a luminii, Unity Cloud, un nou sistem audio și motorul fizic Nvidia PhysX 3.3. Cea de-a cincea generație a motorului Unity a introdus, de asemenea, Efecte de imagine cinematice pentru a ajuta jocurile Unity să arate mai puțin generice. Unity 5.6 a adăugat noi efecte de iluminare și particule, a actualizat performanțele generale ale motorului și a adăugat suport nativ pentru Nintendo Switch, Facebook Gameroom, Google Daydream și API-ul grafic Vulkan. A introdus un player video 4K capabil să ruleze videoclipuri la 360 de grade pentru realitate virtuală. Cu toate acestea, unii jucători au criticat accesibilitatea Unity datorită volumului mare de jocuri produse rapid publicate pe platforma de distribuție Steam de către dezvoltatorii neexperimentați.

În decembrie 2016, Unity Technologies a anunțat că va schimba sistemul de numerotare

a versiunilor pentru Unity de la identificatori pe bază de secvență la anul de lansare pentru a alinia versiunea la cadența lor de lansare mai frecventă; Unitatea 5.6 a fost, aşadar, urmată de Unitatea 2017. Instrumentele Unity 2017 au prezentat un motor de redare grafică în timp real, gradarea culorilor și construirea lumii, analiza operațiunilor live și raportarea performanței. Unity 2017.2 a subliniat planurile Unity Technologies dincolo de jocurile video. Aceasta a inclus noi instrumente, cum ar fi Timeline, care le-a permis dezvoltatorilor să tragă și să fixeze animații în jocuri și Cinemachine, un sistem de camere inteligente în cadrul jocurilor. Unity 2017.2 a integrat, de asemenea, instrumentele 3DS Max și Maya ale Autodesk în motorul Unity pentru un proces de iterație simplificat de partajare a activelor în joc.

Unity 2018 a prezentat conducta de redare scriptabilă pentru dezvoltatori pentru a crea grafică de ultimă generație. Aceasta a inclus Pipeline de redare de înaltă definiție pentru experiențe de consolă și PC și Pipeline de redare ușoară pentru mobil, realitate virtuală, realitate augmentată și realitate mixtă. Unity 2018 a inclus, de asemenea, instrumente de învățare automată, cum ar fi Imitația de învățare, prin care jocurile învață din obiceiurile reale ale jucătorilor, suport pentru Magic Leap și şablonane pentru noii dezvoltatori.

Codul sursă C # al Unity a fost publicat sub o licență „numai referință” în martie 2018, interzicând reutilizarea și modificarea.

În iunie 2020, Unity a introdus Mixed and Augmented Reality Studio (MARS), care oferă dezvoltatorilor funcționalități suplimentare pentru generarea de reguli de aplicații de realitate augmentată (AR). Unity a lansat Unity Forma, un instrument de vânzare automata, pe 9 decembrie 2020.

**Inteligenta artificiala în jocuri** a fost relevanta chiar de la aparitia jocurilor video în anul 1950. Aceasta controlează acțiunile caracterelor care nu sunt controlate de jucători (NPC) și este responsabil de deciziile făcute de acesta. Primul exemplu de utilizare a inteligenței artificiale este în cadrul jocurilor de tip Arcade. În prezent, inteligenta artificială din jocuri are un potențial imens de a crește folosindu-se de tehnici de „Machine Learning”, după cum au dovedit cercetătorii de la DeepMind, autorii lui „AlphaStar” - un agent ce învață care a ajuns la nivelul celor mai buni jucători din lume în succesorul jocului „Starcraft”, „Starcraft2”(3). Oferind capacitatea de a învață, **inteligenta artificială** capătă nivele cu mult superioare fata de ce poate un om să realizeze datorita lipsei de limitări naturale ale oamenilor. Spre exemplu controlarea simultană și perfectă a unui număr mare de unități, sau în Șah calcularea tuturor mișcărilor posibile în următoarele 50 de mișcări, depășind cu mult capacitatea unui om de a face același lucru, calculatorul fiind limitat doar de viteza de procesare și de mărimea memoriei.

## 5.3 Parcurea unui tutorial

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

### Noutăți și facilități Unity

Unity a adus îmbunătățiri și noi instrumente pentru dezvoltatorii de jocuri 2D. A adăugat suport pentru manevrarea animațiilor direct din „timeline” pentru previzualizarea și modificarea acestora. **Interfața programului** foarte bine organizată și foarte ușor de înțeles (Figura 1).

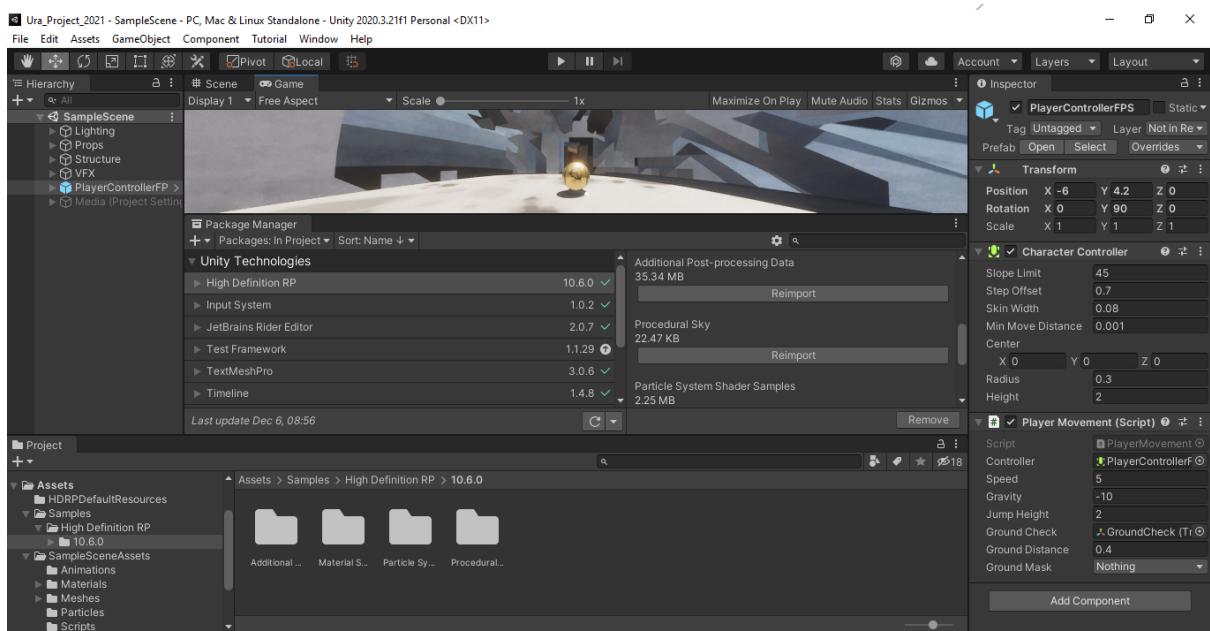


Figura 1. Interfața principală în Unity

Aceasta conține multe opțiuni, principalele sunt următoarele: Project (care ne arată assets-urile), Hierarchy (vizualizăm toate elementele din interiorul jocului), Console (afișează erori, mesaje de tip warning sau putem să vizualizăm debug-ul), Assets Store (pentru a achiziționa diverse GameObject-uri), Prefab-uri (pentru diferite efecte din camera sau teren etc), fereastra Scene (locul unde editam tot ce se află în interiorul proiectului), fereastra Game (defineste interacțiunea cu jocul) cat și butoanele Play, Pause și Forward. De asemenea Unity pune la dispoziția creatorului de jocuri, scripturi care adaugă funcționalități în "Tilemap Editor" precum noi pensule și „tiles” care se pot folosi din pachetul 2D Extras. Editorul 2D pentru tilemap-uri nu mai este disponibil la instalarea Unity, dar poate fi descărcat din Package Manager. O facilitate importantă este introducerea în Unity a componentei 2D Sprite Shape, destinată creării de teren curbat, însotit de un editor care permite definirea marginilor și umpluturii, precum și crearea automată de collider [2]. 2D PSD Importer permite importul Sprite-urilor direct din Photoshop păstrând stratificarea imaginii ceea ce permite proiectarea animațiilor strat cu strat fără să fie nevoie de importarea acestora individuală.

*MonoBehaviour* este clasa de bază a oricărui script care va fi atașat unui obiect în joc și se foloseste la a scrie cod mai rapid, crearea de module cu Inteligență Artificială (AI) fără a avea

cunoștințe prealabile de AI, lumini etc.

#### **Principalele funcții uzuale din clasa MonoBehaviour sunt:**

1. Start() – o funcție apelată o singura dată, în primul frame după activarea scriptului, înaintea oricărei funcții de tipul „Update”
2. Update() – o funcție apelată la fiecare frame
3. FixedUpdate() – o funcție apelată cu frecvența sistemului de fizică, implicit o dată la 0.02 secunde
4. Awake() – o funcție apelată o singura dată asupra tuturor obiectelor din scenă, înainte de toate funcțiile „Start” când instanța scriptului este încarcată

Proprietăți utile:

1. enabled – când un obiect este marcat activ, acesta poate fi actualizat printr-o funcție „Update”.
2. gameObject – face referință la obiectul pe care este atașat componentul.
3. tag – face referință la tag-ul asociat obiectului, poate fi folosit pentru a compara tag-ul obiectului.
4. transform – oferă acces la poziția, rotația și scara obiectului și manipularea acestora
5. name – numele obiectului, exemplu de cod pentru a compara numele obiectului.

Metode importante: GetComponent(Type tipulObiectului); SendMessage(string numeMetoda, object valoare); Destroy(Object obiect) – distrugă obiectul; DontDestroyOnLoad(Object obiect) – obiectul nu va fi distrus la încărcarea unei noi scene; Find(string nume) – găsește un obiect cu numele ales [2][8].

## **Unelte și tehnologii Unity**

**Limbajul C#** este un limbaj de programare cu utilizare generală, de tipul „strong typing”, orientată pe obiect și component. Acesta a fost dezvoltat de Microsoft în 2000 ca parte din proiectul .NET și permite programatorilor să dezvolte aplicații care rulează folosind acesta sistem. Cu ajutorul limbajului C# folosit în Unity, se pot defini componente noi, se pot extinde componente existente, se pot defini interacțiunile jucătorilor și comportamentele inamice, se pot modela obiectele în mișcare și întreg jocul realizat.

#### Caracteristici principale C# sunt:

- „Garbage collection” eliberează automat memoria ocupată de obiecte neutilizate.
- Gestionaerea excepțiilor oferă un mod organizat de a detecta erori
- Pune accent pe versionare pentru asigurarea compatibilității programelor în timpul dezvoltării
- Într-un proiect sponsorizat de Microsoft, numit Mono, C# devine versatil din punct de vedere al portabilității. Mono este o platformă de development open source bazată pe .NET, pe standardele ECMA pentru C# și pe limbajul de infrastructură comun (CLI). Aceasta face posibila portarea pe Linux, macOS, Sony PlayStation și consolele Xbox.

Încapsularea, cunoscută și sub conceptul de „data hiding”, este definită prin gruparea codului împreună cu datele pe care le manipulează, astfel restrictionând accesul la aceste date din exteriorul clasei. Avantaje ale încapsulării: ascunderea implementării clasei față de utilizatori externi; flexibilitate și posibilități de modularizare; Codul este reutilizabil și ușor de modificat. Moștenirea este un concept al programării orientate pe obiect care permite definirea unor clase derivate unei clase de bază care vor moșteni sau supraîncărca funcționalitățile acesteia. Polimorfismul este al treilea concept de bază al programării orientate pe obiect și este definit

de supraîncărcarea metodelor clasei de bază pentru a manipula specific obiectul din clasa derivată.

### Scripturi și rolul acestora

Scripturile sunt un ingredient esențial în implementarea oricărei aplicații în Unity. Un script este constituit dintr-o listă de comenzi care sunt executate de un anumit program. Acestea sunt folosite pentru automatizarea de procese. În Unity, scripturile C# sau JavaScript sunt atașate obiectelor din scenă pentru a determina comportamentul acestuia. Pentru a adăuga un script la componentele unui obiect se apasă butonul „Add Component” din inspector. Unity Engine folosește pentru orice script implementat un sistem de metode „eveniment” care sunt predefinite chiar dacă nu au fost declarate în clasa scriptului. Unity vă permite să vă creați propriile componente folosind scripturi pentru a declanșa evenimente de joc dar și să modificați proprietățile componentelor care formează jocul.

In Unity un script se creează ca în Figura 2.

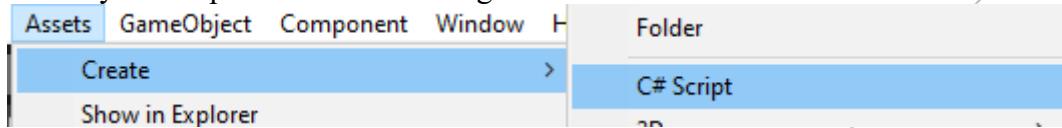


Figura 2. Creare Script C# in Unity

Scriptul se crează în folderul selectat din panoul Proiectului. Implicit Unity va folosi Visual Studio pentru editarea scriptului dar se poate selecta orice editor din Secțiunea Preferences → External Tools, vezi figura 3.

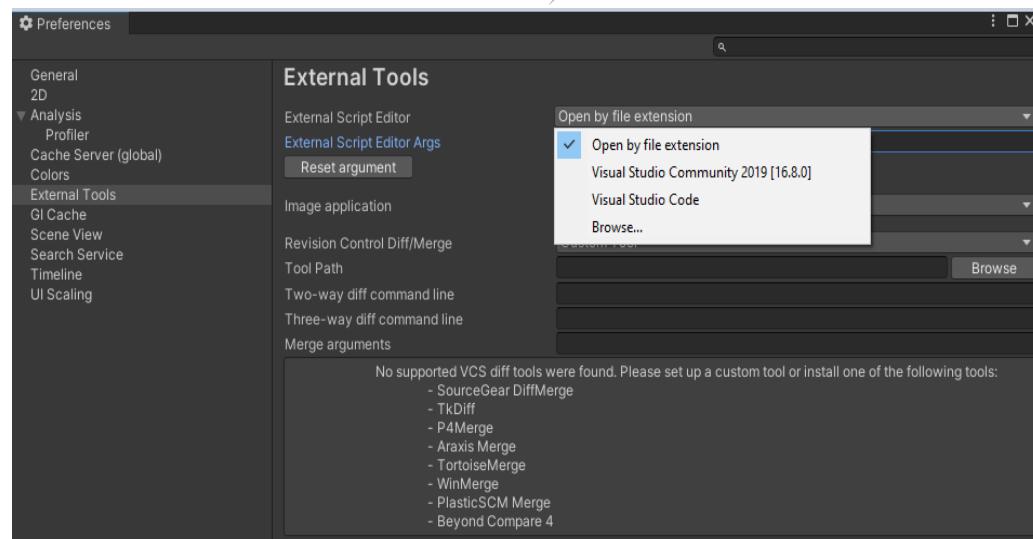
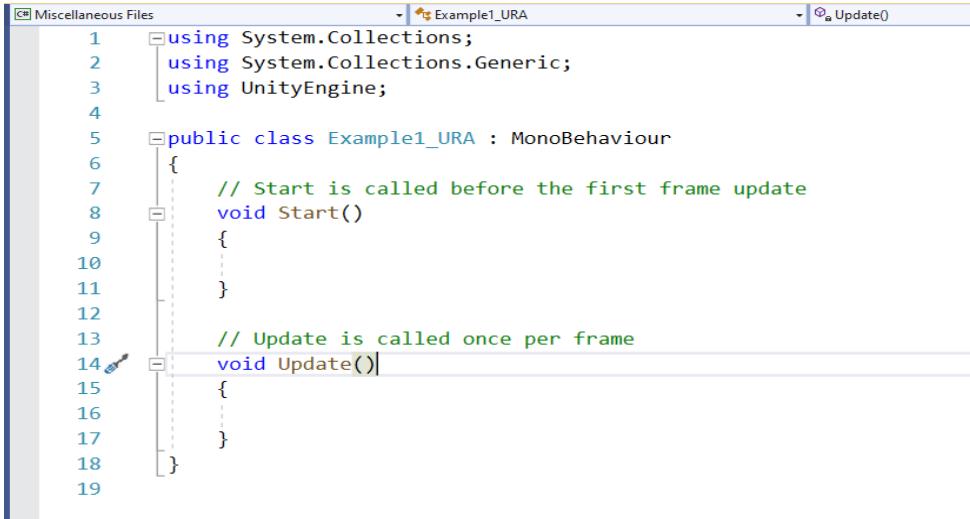


Figura 3. Setare editor pentru scrierea scripturilor

Conținutul inițial al fișierului arată ca în Figura 4. Scriptul realizează legătura cu funcționarea internă din Unity prin implementarea unei clase care derivă din clasa încorporată numită MonoBehaviour.

Dacă atașați o componentă de script la un GameObject, aceasta creează o nouă instanță a obiectului definit de proiect. Numele clasei este preluat din numele pe care l-ați furnizat când a fost creat fișierul. Numele clasei și numele fișierului trebuie să fie aceleași pentru a permite atașarea componentei de script la un GameObject în cazul nostru numele acesteia este "Example1\_URA", ca în figura 4. Funcția „Update()” este locul în care se introduce codul care se va ocupa de actualizarea cadrului pentru GameObject. Aceasta include mișcarea, declanșarea

acțiunilor și răspunsul la intrarea utilizatorului. Este util să puteți configura variabile, să citiți preferințele și să faceți conexiuni cu alte GameObjects înainte de a avea loc orice acțiune de joc. Funcția „**Start()**” se foloseste pentru a face orice inițializare și va fi apelată de Unity înainte ca funcția **Update()** să fie apelată pentru prima dată.



```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class Example1_URA : MonoBehaviour
6  {
7      // Start is called before the first frame update
8      void Start()
9      {
10
11
12
13      // Update is called once per frame
14      void Update()
15      {
16
17
18
19
}

```

Figura 4. Conținutul implicit al unui script in Unity

In Unity, inițializarea unui obiect nu se face folosind o funcție de constructor, deoarece construcția obiectelor este gestionată de editor și nu are loc la începutul jocului. Dacă intentionati să definiți un constructor pentru o componentă de script, acesta va interfera cu funcționarea normală a Unity și poate cauza probleme majore cu proiectul. Un script definește doar un plan pentru o Componentă și, prin urmare, niciunul din codul său nu va fi activat până când o instanță a scriptului este atașată la un GameObject. Puteți ataşa un script trăgând materialul de script într-un GameObject din panoul ierarhic sau în inspectorul din GameObject care este selectat. Elementul prefabricat acționează ca un şablon din care puteți crea noi instanțe prefabricate în scenă. O scenă este o fereastră interactivă din Unity către lumea care este create în aplicatia/jocul curent. Aceasta conține ierarhia obiectelor introduse, camera, luminile, canvas, prefabs. Imediat după ce o scenă este creată aceasta trebuie să fie adăugată în array-ul de scene care se accesează din tabul File → Build Settings, vezi figura 5.

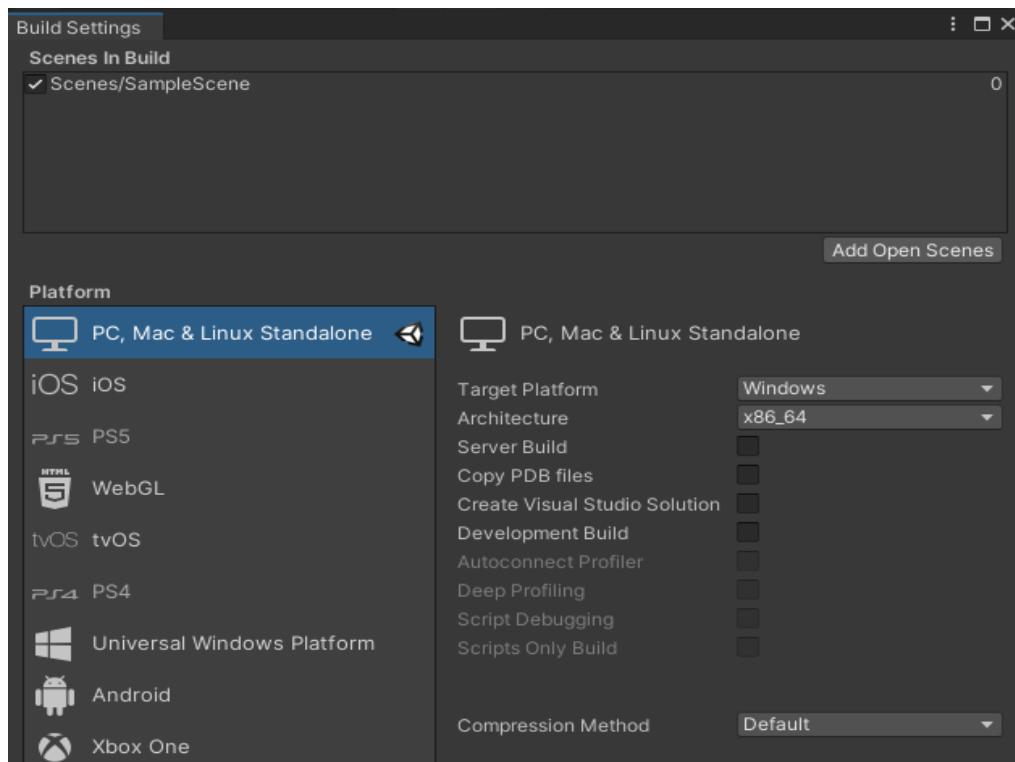


Figura 5. Fereastra Build Settings

Ordinea în care scenele sunt dispuse este foarte importantă deoarece trecerea de la una la cealaltă se poate face doar de sus în jos. Prefabul este un obiect de tip template care permite stocarea unui obiect cu toate elementele sale pe care îl putem folosi de câte ori dorim. Prefabul se poate folosi ca GameObject în scripturi, el se salvează cu toate setările, componentele și scripturile adăugate. Salvarea unui prefab necesită suprascriere atât timp cât nu este accesat direct din interfața prefab editor.

Sistemul Prefab din Unity vă permite crearea, configurarea, stocarea unui GameObject complet cu toate componentesale. Dacă intenționați să reutilizați un GameObject configurat ca de exemplu un personaj, un peisaj – în diferite locuri din scena sau în mai multe scene ale proiectului, trebuie să îl convertiți într-un prefabricat. Acest lucru recomandat, și nu simplă copiere și lipire a GameObject, deoarece sistemul Prefab vă permite să păstrați automat toate copiile sincronizate. Pentru a crea ierarhii complexe de obiecte este util să se realizeze o imbricare a prefabricatorilor în alte prefabricate, deoarece se pot suprascrie setările pentru instanțe prefabricate individuale. Este utilă utilizarea Prefabs cand trebuie să instantiem GameObjects. Pentru a instanția un prefabricat în timpul execuției, codul scriptului trebuie să contină o referință la acel prefabricat. Puteți face această referință creând o variabilă publică în codul dvs. pentru a păstra referința Prefab. Variabila publică din codul apare ca un câmp atribuit în Inspector. Puteți atribui apoi prefabricatul real pe care doriti să îl utilizați în Inspector.

### Lumini și iluminare în Unity game engine

Creatorul de jocuri în Unity poate realiza o iluminare realistă care se potriveste foarte bine pentru o gamă mare a stilurilor de artă cunoscuță. În Unity iluminarea este implementată prin aproximarea modului în care se comportă lumina în lumea reală deoarece Unity folosește modele detaliate despre cum funcționează lumina pentru un rezultat mai realist, sau modele simplificate pentru un rezultat mai stilizat. Iluminare creata în Unity poate fi directă și indirectă iar pentru a obține rezultate realiste de iluminare, trebuie să simulați atât lumina directă, cât și cea indirectă. Lumina directă este lumina care este emisă, loviște o suprafață o dată și apoi este reflectată direct într-un senzor (de exemplu o cameră). Lumina indirectă reprezintă lumina care loviște suprafețele de mai multe ori inclusiv lumina cerului. Unity poate calcula iluminarea

directă, iluminarea indirectă sau atât iluminarea directă, cât și cea indirectă iar tehniciile de iluminare pe care le folosește Unity depind de modul în care creatorul de joc configurează proiectul. Instrumentele pentru luminarea Scenei folosesc parametrii ușor de configurat, cum se observă în Figura 6.



Figura 6 Panoul pentru setarea luminii

Principalele tipuri de proprietăți pentru definirea luminii în Unity se pot observa în figura 7.

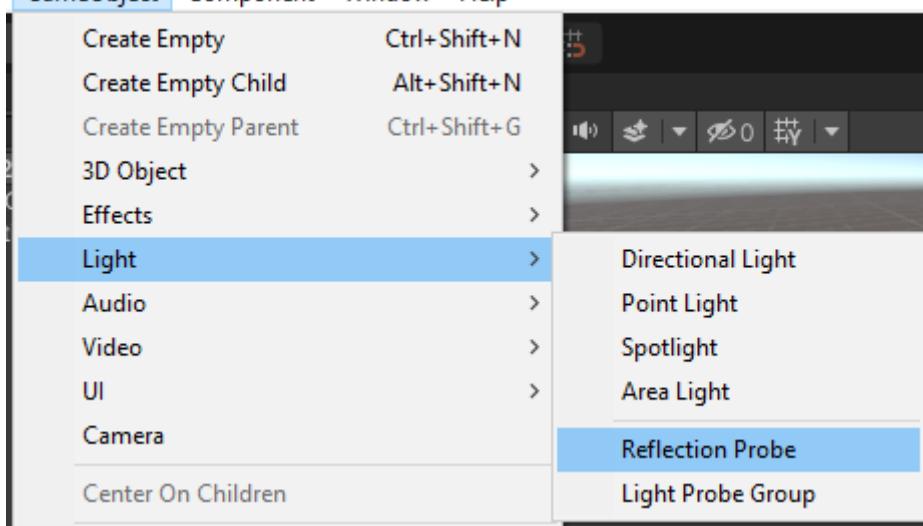


Figura 7. Tipurile de lumini în Unity

#### **Identificarea caracteristicilor principalele pentru diverse tipuri de lumini:**

- **Lumini punctiforme**

O lumină punctiformă este localizată într-un punct din spațiu și trimite lumina în toate direcțiile în mod egal. Direcția luminii care lovește o suprafață este linia de la punctul de contact înapoi spre centrul obiectului luminos. Intensitatea scade odată cu distanța față de lumină, ajungând la zero într-un interval de timp specificat. Intensitatea luminii este invers proporțională cu pătratul distanței de la sursă, vezi Figura 8. Aceasta este cunoscută sub numele de „lege inversului păratului” și este similară cu modul în care se comportă lumina în lumea reală. Luminile punctiforme sunt utile pentru simularea lămpilor și a altor surse locale de lumină într-o scenă și se pot putea folosi pentru a face o scânteie sau o explozie în scopul de a lumina împrejurimile

cat mai convingător.

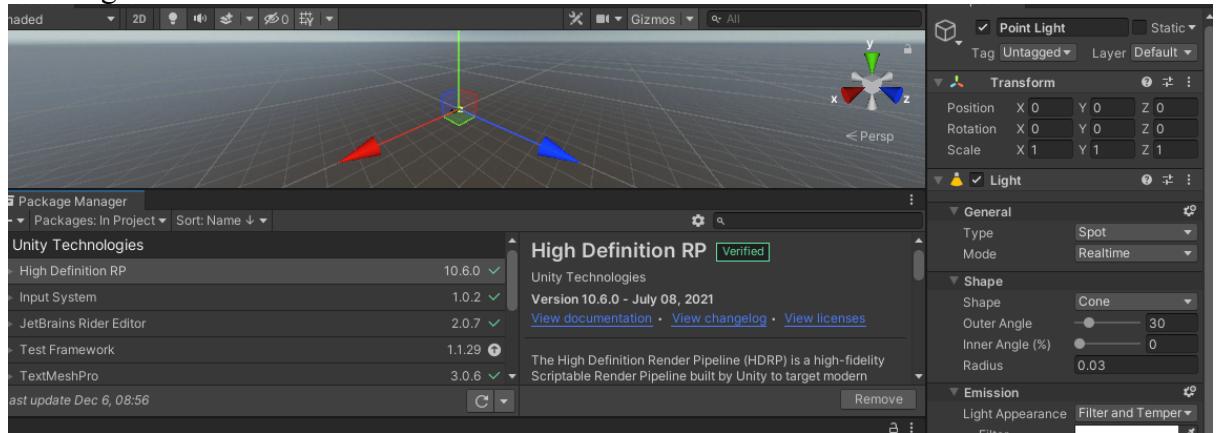


Figura 8. Point Light in Unity

- **Spoturi**

O lumină spot are o locație specificată și un interval peste care lumina se stinge. Cu toate acestea, lumina spot este limitată la un unghi, ceea ce duce la o regiune de iluminare în formă de con. Centrul conului indică direcția înainte (Z) a obiectului luminos. Lumina scade, de asemenea, la marginile conului de lumina. Luminile spot sunt utilizate pentru surse de lumină artificială, cum ar fi lanterne, faruri etc. O lumină spot în mișcare va lumina doar o mică zonă a scenei și va crea efecte de iluminare dramatice. Fereastra pentru definirea proprietăților luminii spot este în Figura 9

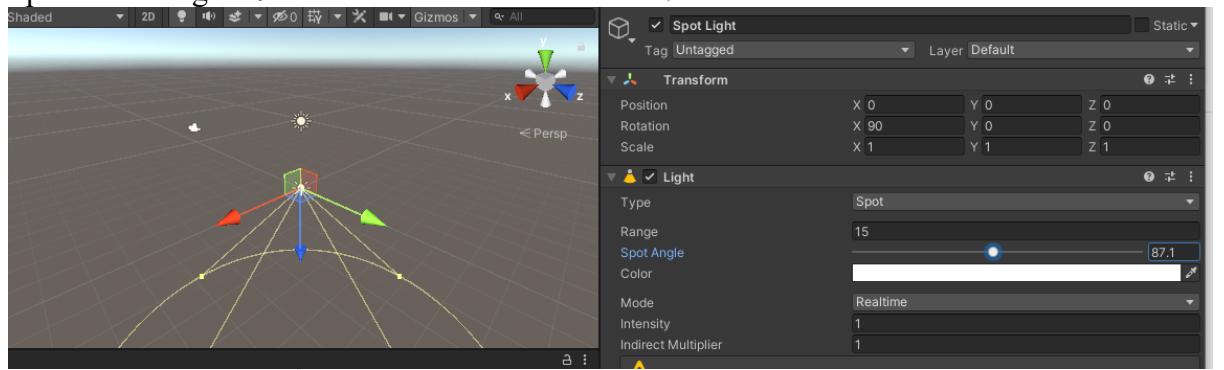


Figura 8. Lumina spot in Unity

- **Lumini direcționale**

Luminile direcționale se folosesc pentru a crea efecte ca și lumina soarelui și pot fi gândite ca surse de lumină îndepărtate, care există la o distanță infinită față de obiecte. Toate obiectele din scenă sunt iluminate ca și cum lumina ar fi întotdeauna din aceeași direcție. Distanța luminii față de obiectul său nu este definită și deci lumina nu se diminuează. Luminile direcționale reprezintă surse mari, îndepărtate, care provin dintr-o poziție în afara gamei lumii jocului. În scenele realiste, ele pot fi folosite pentru a simula soarele sau luna iar în jocurile abstractive permit adăugarea de umbre obiectelor fără a specifica exact de unde provine lumina.

În mod implicit, fiecare nouă scenă din Unity conține o lumină direcțională. Rotirea luminii direcționale implicate (sau „Soarele”) provoacă „Skybox”-ul sa se actualizeze. Cu lumina înclinată în lateral, paralel cu solul, se pot realiza efecte de apus. În plus, îndreptarea luminii în sus face ca cerul să se întunece, ca și cum ar fi fost noaptea. Cu lumina înclinată de sus în jos, cerul va semăna cu lumina zilei. Dacă Skybox-ul este selectat ca sursă ambientală, Ambient Lighting se va schimba în raport cu aceste culori, vezi figura 10.

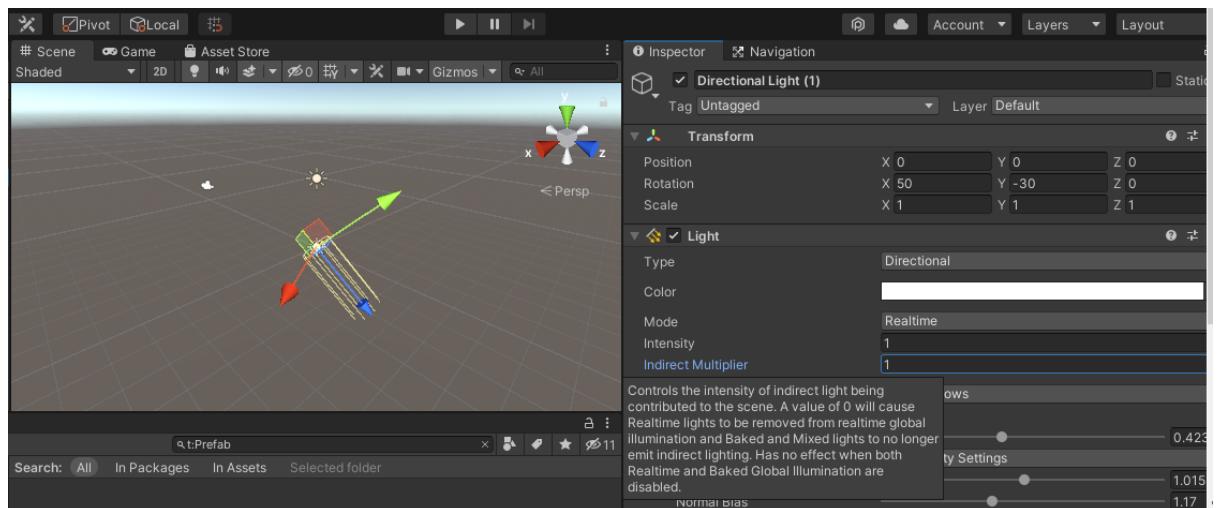


Figura 10. Lumini direcționale in Unity

### • Lumina de zonă

O lumină de zonă este definită de un dreptunghi în spațiu. Lumina este emisă în toate direcțiile uniform pe suprafața lor, dar numai dintr-o parte a dreptunghiului. Nu există un control manual pentru raza unei zone de lumină, cu toate acestea intensitatea va scădea în pătratul invers al distanței pe măsură ce se deplasează departe de sursă. Deoarece calculul iluminatului este destul de intensiv în procesoare, luminile din zonă nu sunt disponibile la timpul de funcționare și pot fi prezentate doar cu light maps. Deoarece o lumină de zonă luminează un obiect din mai multe direcții diferite simultan, umbrirea tinde să fie mai moale și mai subtilă decât celelalte tipuri de lumină. Setarea acestui de lumina se realizeaza ca in Figura 11.

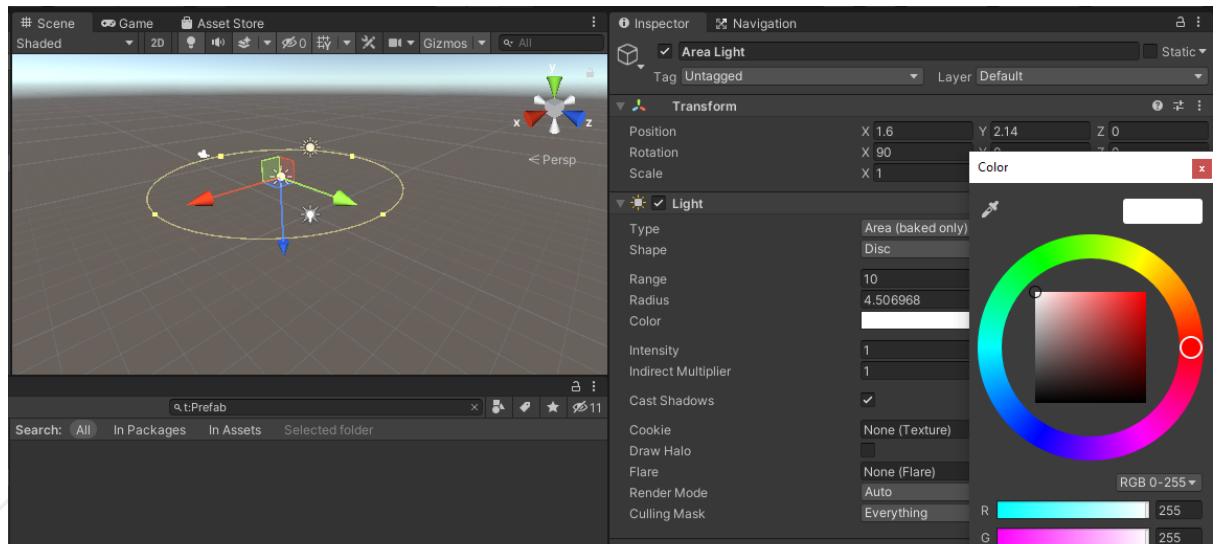


Figura 10. Area Light in Unity

Unity oferă utilizatorilor posibilitatea de a crea jocuri 2D și 3D astfel incat motorul de joc contine un API primar de scripturi realizat in limbajul C#. Pentru jocurile 2D Unity permite importul de sprite-uri cat si un render avansat 2D. Pentru realizarea jocurile 3D Unity permite specificarea compresiei texturii, a mipmap-urilor și a setărilor de rezoluție pentru fiecare platformă suportata de catre game engine-ul. S-a constatat ca, Unity oferă suport pentru mapare cu bump, mapare reflecție, mapare cu parallax, ocluzie spațială pe ecran, umbre dinamice folosind shadow maps dar si efecte de post-procesare full-screen. Creatorii de jocuri au

facilitatea de a deveni editori în magazinul de active iar apoi au posibilitatea de a vinde creațiile lor. Unity Asset Store poate fi vizitat din site-ul aplicatiei, sau prin intermediul Unity Game Engine. Astazi, in conditii de pandemie exista tendinta mare in realizarea si dezvoltarea platformelor game engine, fapt sustinut inclusiv de cautarile utilizatorilor in Internet.

### Bibliografie

- [1] Tsai, Y.T., Jhu, W.Y., (...), Chen, C.Y., Unity game engine: interactive software design using digital glove for virtual reality baseball pitch training, Microsystem Technologies-Micro-And Nanosystems-Information Storage And Processing Systems, 27 (4), pp.1401-1417, Apr 2021;
- [2] <https://unity.com/learn>
- [3] Goldstone, W., Unity Game Development Essentials, Packt Publishing Ltd., ISBN 978-1-847198-18-1, UK, 2009;
- [4] Menard, M. and Wagstaff, B., Game development with Unity. Nelson Education, 2015;
- [5] Valcasara, N., Unreal Engine Game Development Blueprints. Packt Publishing Ltd., 2015;
- [6] Hocking, J., Unity in Action: Multiplatform Game Development in C# with Unity 5, 1st Edition, Manning Publications, 2015;
- [7] Murray, J., C# Game Programming Cookbook for Unity 3D, CRC Press, 2014;
- [8] Thorn, A., Pro Unity Game Development with C#. Berkeley, CA: Apress, ISBN: 978-1-4302-6745-4, New York, 2014;
- [9] Dickson, P.E. et al., An experience-based comparison of unity and unreal for a stand-alone 3D game development course. In: Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education, pp. 70–75, 2017;
- [10] Blackman, S., Beginning 3D game development with Unity: World's most widely used multi-platform game engine, Apress, 1st Edition, ISBN: 978-1-4302-3423-4, 2011;

## Model bilet 1

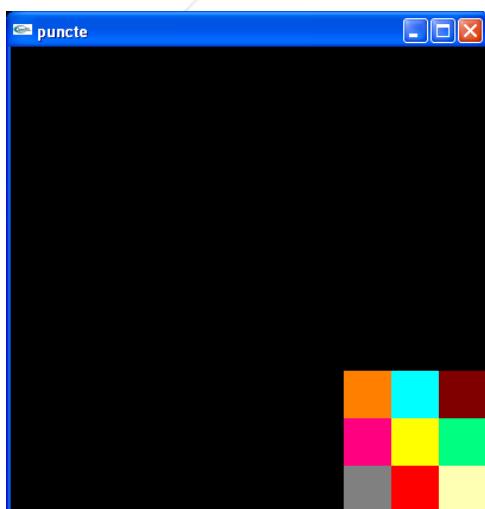
1. Caracterizati grafica digitală (2 pct).
2. Caracterizati grafica vectorială (2 pct).
3. Explicati modelele de culori RGB si CMYK (2 pct).
4. Sa se explice fiecare functie *OpenGL* utilizata in programul urmator (1.5 pct).  

```
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. Prezentati cel putin 4 idei principale din referatul intocmit (2 pct).

2. Caracterizati grafica vectorială (2 pct).

3. Explicati modelele de culori RGB si CMYK (1 pct).

4. Sa se explice fiecare functie *OpenGL* utilizata in programul urmator (2 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. Explicati ce rezulta in urma testarii setului de comenzi (2 pct).

```

1. #include <GL/glut.h>
2. #include <stdlib.h>
3. #include <math.h>
4. void Display(void)
a. {
b.     glClear(GL_COLOR_BU
FFER_BIT);
c.     glBegin(GL_LINES);
d.     for(int i=0;i<100;++)
i.     {
ii.         glVertex3f(0,0,0);
iii.        glVertex3f(1-
i/100.0,i/100.0,0);
```

```

iv.    }
e.     glEnd();
f.     glFlush();
g.    }

5. int main(void)
a. {
b.     glutCreateWindow("mode
l");
c.     glutDisplayFunc(Display);
d.     glColor3f(1,1,0);
e.     glutMainLoop();
f.     return 0;
g.    }
```

### 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);
```

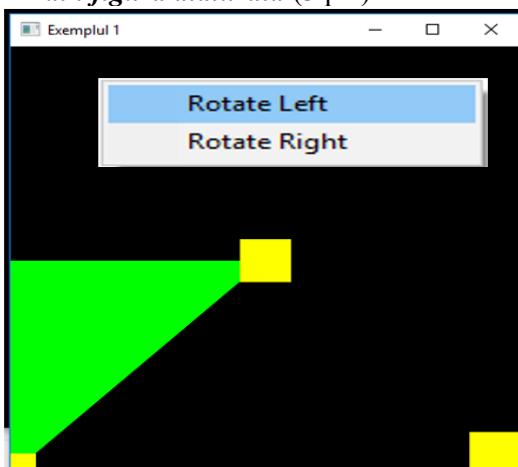
```
	glutCreateWindow("aplicatii");
```

```
	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. Prezentati cel putin 6 idei principale din referatul intocmit (1 pct).

### Partea II. Completati raspunsul corect:

1	2	3	4

1. **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)
2. **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)
3. **Propriile formate de fisiere in Photoshop sunt:** a) PSD si PSB; b) PSD si JPG; c) JPG si PNG. (0,5 pct)
4. **Alegeti raspunsul corect, corespunzator functiei glutInit (int\* argc, char\*\* argv):** a) functia initializeaza GLUT-ul folosind argumentele din linia de comanda si intoarce o variabila de tip int care reprezinta numarul ferestrei care se initializeaza; b) nu este corect definita; c) functia initializeaza GLUT-ul folosind argumentele din linia de comanda si trebuie apelata inaintea oricror altor functii GLUT sau OpenGL. (0,5 pct)

## Model bilet 4

### Partea I Completati raspunsul corect:

1	2	3	4

5. **Care este rolul primitivei GL\_LINE\_LOOP?** a) Desenează o linie poligonală formată din segmentele  $(v_0, v_1), (v_1, v_2), \dots (v_n, v_0)$ ; b) Desenează linia poligonală formată din segmentele  $(v_0, v_1), (v_1, v_2), \dots (v_{n-2}, v_{n-1})$ ; c) Desenează segmentele de dreapta izolate  $(v_0, v_1), (v_2, v_3), \dots$  dacă  $n$  este impar ultimul vârf este ignorat. (0,5 pct)
6. In OpenGL stergerea unui submenu 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\_MIDDLE\_BUTTON; c) GLUT\_RIGHT\_BUTTON. (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.

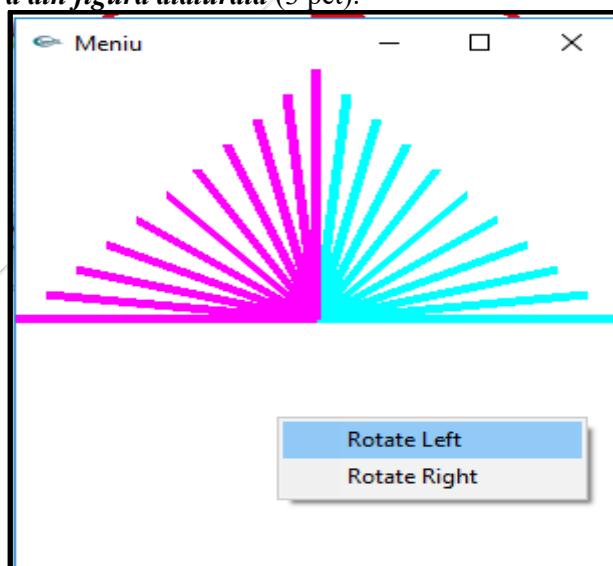
1. Sa se explice functiile *OpenGL* utilize in programul urmator, cat si ce rezulta dupa testarea aplicatiei (3 pct).

```
#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. Prezentati cel putin 6 idei principale din referatul intocmit (1 pct).