

Grafică pe calculator (MLR5060)

Elemente de grafică 3_D

1. OpenGL - Noțiuni introductice

Realizarea aplicațiilor 3D utilizând OpenGL în C# cu TAO

TAO realizează o legatură cu bibliotecile utilizate la aplicații de grafică 2D/3D:

- *OpenAl - API* multiplatformă folosit pentru generarea de sunete poziționate 3D în jocuri și alte aplicații audio.
- *OpenGL - Open Graphics Library API* principal folosit la generarea de imagini 2D,3D
- *Sdl - Simple DirectMedia Layer* este folosită la creare de aplicații multiplatformă multimedia care oferă acces ușor la realizarea de grafică 2D și 3D, sunet, de asemenea și la dispozitivele de IO (mouse, tastatura).
- *Platform.Windows* - oferă access la API-ul Windows și la funcțiile de desenare de tip *GDI (Graphics Device Interface)*
- *PhysFs* - oferă suport pentru realizarea și accesarea de arhive abstracte folosite în jocuri pentru a memora hărțile de nivel (ex.: Quake 1,2)
- *FreeGlut* - este o alternativă opensource la *GLUT (OpenGL Utility Toolkit)*. Oferă suport pentru operații de tip IO cu tastatura sau mouse-ul, definirea și crearea de ferestre, acces-ul la diferite primitive grafice

... Realizarea aplicațiilor 3D utilizând OpenGL în C# cu TAO

- *ODE- Open Dynamics Engine* reprezintă un motor de fizică în timp real fiind compus dintr-o parte care se ocupă cu fizica corpurilor rigide și una care asigură detectarea coliziunilor.
- *Glfw* - Facilitează dezvoltarea *OpenGL* oferind acces la creare de ferestre, monitorizare IO, încarcarea texturilor din fișiere, crearea și sincronizarea thread-urilor.
- *DevIL - Developer's Image Library* cu ajutorul căreia pot fi încărcate imagini salvate în diferite formate (.bmp, .cut, .dds, .doom, .exr, .hdr, .gif, .ico, .jp2, .jpg, .lbt, .mdl, .mng, .pal, .pbm, .pcd, .pcx, .pgm, .pic, .png, .ppm, .psd, .psp, .raw, .sgi, .tga și .tif file)
- *Cg - C for Graphics* limbaj de nivel înalt dezvoltat de *Nvidia* folosit la crearea de shadere pentru plăcile video care suportă aceasta tehnologie.
- *Lua* - permite executarea de script-uri scrise în limbajul *Lua*.

Etape în realizarea aplicațiilor 3D utilizând OpenGL în C# cu TAO

Instalarea :

Framework-ul **Tao** poate fi descărcat de la adresa [Tao framework](http://www.taoframework.com/downloads):
<http://www.taoframework.com/downloads>

În aplicatia prezentată în continuare vom folosi pentru afişare controlul **SimpleOpenGLControl** care se găseşte în **Tao.Platform.Windows**.

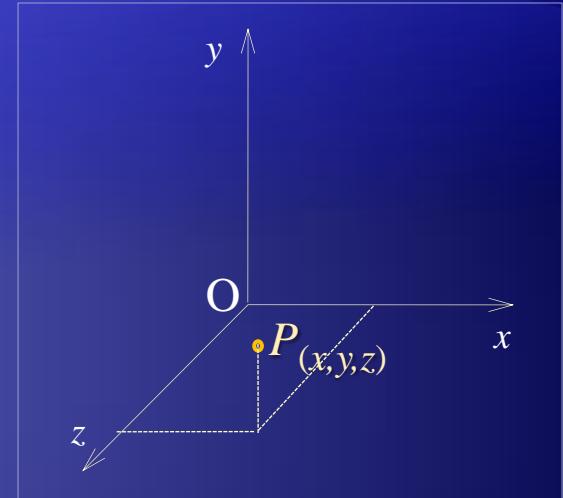
În *Visual Studio* vom creea un proiect de tip *Windows Application* la care vom adăuga ca referințe **Tao.OpenGl** și **Tao.Platform.Windows**.

Notiuni de bază ~ OpenGL

Coordonate

În *OpenGL* utilizam un sistem cartezian în care axele X, Y sunt în planul monitorului, iar axa Z (adâncimea) vine spre observator.

Un punct P este precizat prin trei coordonate (x,y,z) .



Primitive

În *OpenGL* denumirile functiilor contin sufixul

- 3f pentru coordonate 3D,
- 2f pentru *coordonate* 2D [$f \rightarrow$ argumente de tip *float* (in virgula mobila)].

Valorile (in virgula mobila) pentru componente de culoare $\in [0,1]$.

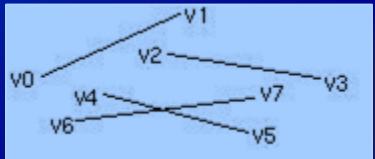
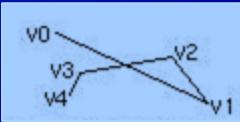
... Primitive

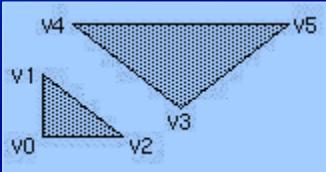
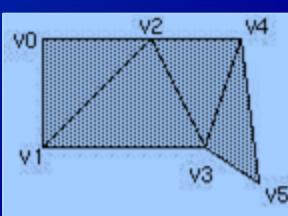
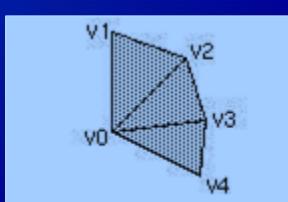
Desenarea se realizeaza prin scrierea unei secvențe de comenzi între (apelul functiilor) *glBegin()* si *glEnd()*.

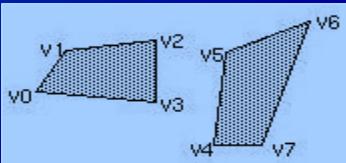
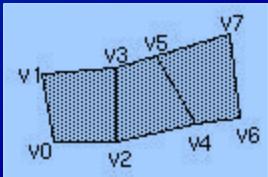
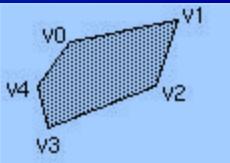
Functia *glBegin()* primește ca parametru tipul primitiveelor grafice care urmează să fie desenate. În funcție de acest parametru și de modul în care sunt organizate vârfurile, putem desena primitivele din tabelul de mai jos:

<i>Primitiva</i>	<i>Argument glBegin</i>	<i>Explicatii</i>
	<i>GL_POINTS</i>	Deseneaza câte un punct pe fiecare axă <code>glBegin(GL_POINTS);</code> <code>glVertex3f(0.5,0.0,0.0);</code> <code>glVertex3f(0.0,0.5,0.0);</code> <code>glVertex3f(0.0,0.0,0.5);</code> <code>glEnd();</code>

... Primitive :

<i>Primitiva</i>	<i>Argument glBegin</i>	<i>Explicatii (intre glBegin() si glEnd())</i>
	<i>GL_LINES</i>	Se declara câte două pentru fiecare linie.
	<i>GL_LINE_STRIP</i>	Specificam doar punctele pentru prima linie și după ele urmatoarele puncte care compun conturul
	<i>GL_LINE_LOOP</i>	Se mai trasează și o linie de la primul la ultimul punct specificat, pentru a desena poligoane.

Primitiva	Argument <code>glBegin</code>	Explicatii (intre <code>glBegin()</code> si <code>glEnd()</code>)
	<code>GL_TRIANGLES</code>	<p>Cu ajutorul functiei <code>glVertex3f</code> specificam câte 3 puncte pentru fiecare triunghi:</p> <ol style="list-style-type: none"> I. vârful din stânga jos al triunghiului, II. vârful din dreapta jos, III. vârful de sus al triunghiului.
	<code>GL_TRIANGLE_STRIP</code>	<p>Daca avem mai multe triunghiuri pe o latura putem sa declarăm triunghiul initial de la care se pleaca si lista de puncte aditionale incluse în celelalte triunghiuri.</p>
	<code>GL_TRIANGLE_FAN</code>	<p>Utila daca triunghiurile au un vârf comun [putem desena cele 3 triunghiuri cu numai 5 apeluri <code>glVertex3f</code> (in loc de 6)].</p>

<i>Primitiva</i>	<i>Argument glBegin</i>	<i>Explicatii (intre glBegin() si glEnd())</i>
	<i>GL_QUADS</i>	Permite desenarea patrulaterelor specificand cele 4 puncte varfuri.
	<i>GL_QUAD_STRIP</i>	Se Specifica primul patrulater, apoi punctele aditionale, reducand apelurile functiei glVertex3f.
	<i>GL_POLYGON</i>	Se vor specifica varfurile poligonului.

Transformari geometrice

Transformarile in *OpenGL* sunt: *Rotatia*, *Scalarea*, *Translatia* si *Proiectia*, coordonatele unui vârf fiind date ca vector coloana cu 4 elemente, (*coordonate omogene*), ultima valoare fiind 1.

OpenGL asigura *stive de matrice* pentru memorarea matricelor pentru fiecare stare reprezentata printr-o matrice curenta (stive si matrice curente pentru modelare-vizualizare, proiectie, texturare etc). Se va comuta pe fiecare tip de stiva si se va stabili matricea curenta înainte de reprezentarea scenei.

Matricea de *modelare-vizualizare* contine toate transformarile geometrice care s-au aplicat scenei curente, (poate contine 32 de elemente).

Stiva de *proiectie* se refera la pozitionarea camerei si la tipul acesteia. Prin schimbarea acestei stive putem obtine efecte interesante de vizualizare de tipul "*fish-eye*".

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \alpha & -\sin \alpha & 0 \\ 0 & \sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_y(\alpha) = \begin{bmatrix} \cos \alpha & 0 & \sin \alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \alpha & 0 & \cos \alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_z(\alpha) = \begin{bmatrix} \cos \alpha & -\sin \alpha & 0 & 0 \\ \sin \alpha & \cos \alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$S(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & t_y \\ 0 & 0 & s_z & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T(t_x, t_y, t_z) = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Stiva de *texturare* contine transformarii aplicate texturiilor curente. Pentru a stabili care stiva de matrici vrem sa o modificam *OpenGL* pune la dispozitie functia [*glMatrixMode*] care primeste ca parametru tipul stivei:

- ❑ *GL_MODELVIEW* (implicit),
- ❑ *GL_PROJECTION,*
- ❑ *GL_TEXTURE.*

Pentru operatii cu stiva curenta avem urmatoarele functii :

- a) *glLoadIdentity()* - încarca matricea identitate pe stiva,
- b) *glLoadMatrixf(const GLfloat *m)* - încarca în matricea curenta matricea trimisa ca parametru (pointer la tablou de 16 elemente)
- c) *glMulMatrix(const GLfloat *m)* - înmulteste matricea curenta cu matricea primita ca parametru
- d) *glPushMatrix()* - pune în vârful stivei matricea curenta
- e) *glPopMatrix()* - extrage matricea din vârful stivei fara a o salva undeva.

Stivele de matrice sunt utilizate deoarece este mai eficient sa salvezi sau sa restaurezi o matrice decât sa o calculezi sau generzi din nou. Asupra obiectelor scenei se va aplica întotdeauna matricea curentă. Pentru a aplica mai mult de 2 transformari putem folosi functia de înmulțire `glMulMatrix`. Modul descris mai sus de a executa transformari este destul de greoi deoarece trebuie specificate matricile de transformare. O modalitate mai usoara este folosirea functiilor de transformare. Acestea sunt:

- a) `glRotatef(GLfloat angle, GLfloat x, GLfloat y, GLfloat z)` - parametrul `angle` reprezinta unghiul de rotatie în grade iar `x,y` si `z` reprezinta coordonatele vectorului în jurul caruia se face rotatia.
- b) `glScalef(GLfloat x, GLfloat y, GLfloat z)` - parametri reprezinta factorii de scalare pentru fiecare axa
- c) `glTranslate(GLfloat tx, GLfloat ty, GLfloat tz)` - functia va produce o translatie de vector de coordonate (`tx ty tz`)

Referințe

1. A gentle introduction to Tao.OpenGl using SimpleOpenGLControl,
 - http://members.hellug.gr/nkour/Tao.OpenGL_Builder/SimpleIntro_Borland.html
2. Dezvoltarea aplicațiilor OpenGL pe platforma .NET folosind suita TAO,
 - http://profs.info.uaic.ro/~alaiba/mw/index.php?title=Dezvoltarea_aplica%C5%A3iilor_OpenGL_pe_platforma_.NET_folosind_suita_TAO
3. Cursuri L_T,
 - http://www.cs.ubbcluj.ro/~per/Grafica/L_T/L_T.htm
4. Cursuri, Laboratoare G_A,
 - <http://www.cs.ubbcluj.ro/~anca/grafica/> , <http://www.cs.ubbcluj.ro/~anca/graphics/>

Temă

Realizarea unei aplicatii simple care sa permita:

- ✓ *vizualizarea unui obiect 3D,*
- ✓ *rotirea si*
- ✓ *miscarea sa.*

Success!