20/11/2011

FACULTATEA DE AUTOMATICA SI CALCULATOARE

ELEMENTE DE GRAFICA PE CALCULATOR

Laborator 6

Lantul de transformari si camera

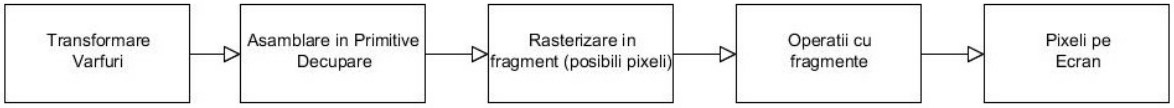
Acest laborator are 2 parti: partea A, in care sunt prezentate transformarile prin care se ajunge de la un vertex intr-un spatiu abstract, definit prin unitati de masura abstracte la o reprezentare exacta pe ecran si partea B in care sunt prezentate cunostiintele necesare pentru implementarea unei camera simple.

PARTEA A - Transformari

1. Banda grafica pe scurt

Transformarea descrierii scenei 3D intr-o imagine afisata pe ecran este realizata automat de API-urile 3D, printr-un lant de prelucrari **fix**. Acest lucru este valabil pentru versiunea de OpenGL 1.0, care este predata la laborator.

Lantul de prelucrari consta din etapele redate in figura urmatoare:



Deoarece fiecare etapa primeste intrari de la etapele anterioare si transmite rezultate pentru etapele urmatoare, secventa de operatii este asemanata cu o “banda de asamblare” intr-un proces de fabricatie. Deoarece obiectul fabricat in acest caz este imaginea unei scene 3D, banda este denumita „banda grafica” („graphics pipeline”). Operatiile din banda grafica fixa, redata in figura de mai sus, sunt aceleasi pentru toate primitivele unei scene 3D.

Semnificatia fiecarei etape, foarte pe scurt:
Transformare varfuri – se ia fiecare varf(vertex) din scena si se transforma in parelel, pe placa grafica, pana cand ajunge in pozitia sa finala (dupa proiectie)
Asamblare in Primitive si Decupare – cu vertecsii de la etapa precedenta si cu informatia ce defineste legarea lor se creeaza triunghiuri. Triunghiurile ce sunt in afara volumului vizual sunt decupate.
Rasterizare in fragmente – se iau primitivele de la etapa precedenta si se transforma in patrate de dimensiunea pixelilor ce se numesc fragmente.
Operatii cu fragmente – se iau fragmentele de la etapa precedenta si se fac operatii pe ele: colorare, iluminare, etc.
Pixeli pe ecran – fragmentele ce au trecut de etapa precedenta sunt procesate alaturi de pixelii deja existenti.

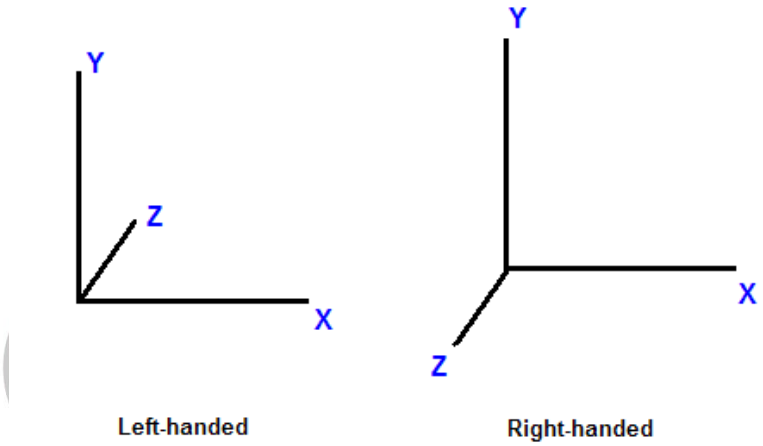
2. Sisteme de axe si ordine operatii

Inainte de a putea trece la operatiile efective trebuie inteles modul in care OpenGL lucreaza cu datele. Toate datele sunt prelucrate prin operatii vectoriale/matriceale, de aceea este nevoie de o intelegere clara a acestor operatii cat si a spatiului in care sunt facute.

OpenGL foloseste un sistem de coordonate dreapta. Acest lucru inseamna ca orice produs vectorial a doua axe in directia pozitiva va avea ca rezultat cea de-a treia axa in directia pozitiva sau mai simplu spus se respecta regula mainii drepte si sunt valabile urmatoarele relatii:

$$i \times j = k \quad j \times i = -k$$
$$j \times k = i \quad k \times j = -i$$
$$k \times i = j \quad i \times k = -j$$

In figura urmatoare se poate vedea diferenta dintre un sistem de coordonate dreapta si un sistem de coordonate stanga.



Trebuie mentionat faptul ca se folosesc matrici si vectori in forma **coloana**. Ca o scurta recapitulare:
Forma Linie: $[x' , y' , z'] = [x, y, z] * M1 * M2 * M3$
Forma Coloana $[x', y', z']^T = M3 * M2 * M1 * [x, y, z]^T$ unde T reprezinta operatorul de transpunere.

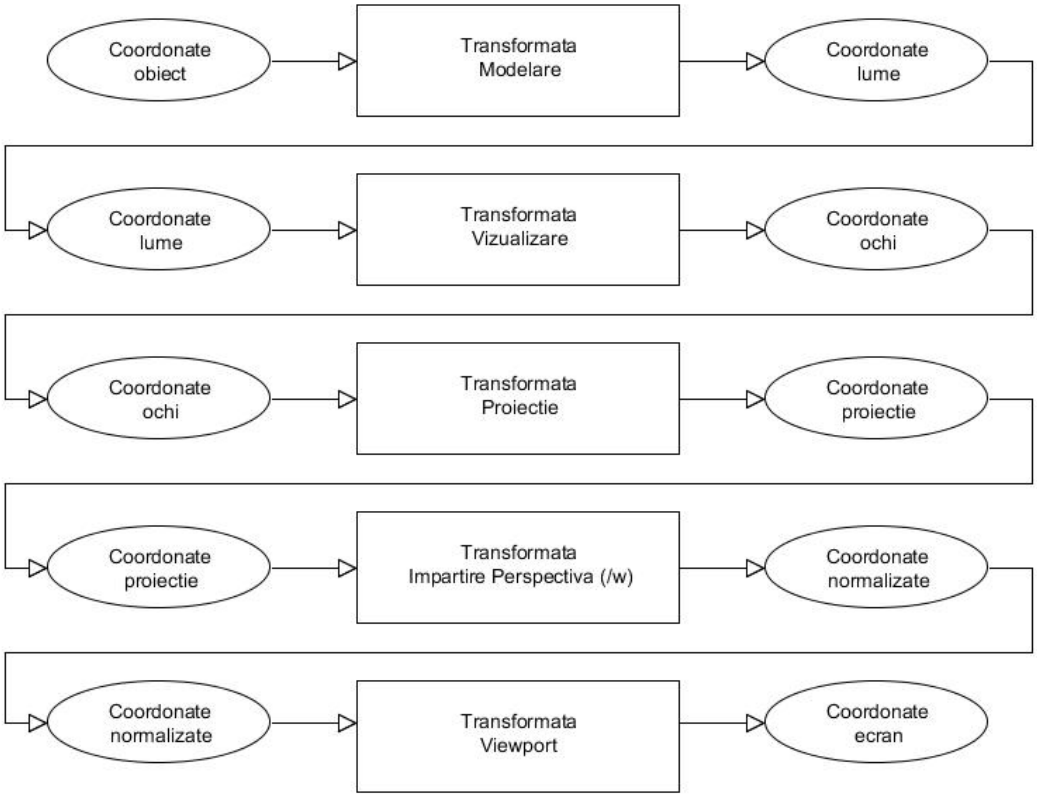
Deci in OpenGL daca se doreste mai intai rotatie , apoi translatia si la sfarsit scalarea unui set de coordonate rezultatul final este: $[x', y', z']^T = S * T * R * [x, y, z]^T$

OpenGL lucreaza cu matrici de dimensiunea 4x4, iar aceastea pot fi obtinute sub forma liniara, deci un vector de 16 float-uri. Coeficientii unei matrici sunt numerotati astfel (ordinea lor poate creea usor confuzii!):
m0 m4 m8 m12
m1 m5 m9 m13
m2 m6 m10 m14
m3 m7 m11 m15

Deci primele 4 valori din matrice in forma liniara reprezinta prima coloana nu prima linie!

3. Lantul de transformari

Lantul de transformari pe care il urmeaza un varf este impartit in mai multe etape, dar inainte ca aceastea sa fie explicate este indicat sa fie facuta o prezentare de ansamblu a intregului proces.



Dupa cum se observa in imagine un varf trece printr-o serie de ***spatii***, fiecare avand o semnificatie specifica. In spatiul obiect sunt modelate coordonatele varfurilor unui obiect, in spatiul lume este pozitionat acest obiect intr-o scena, in spatiul ochi sunt reprezentate obiectele asa cum sunt ele vazute de observator, in spatiul proiectie sunt pozitionate obiectele intr-un volum de vizualizare, in spatiul NDC(normalized device coordinates) sunt pozitionate obiectele intr-un volum canonic de vizualizare, iar in spatiul ecran obiectele sunt pozitionate relativ la fereastra de afisare.

In cazul OpenGL, matricile de modelare si de vizualizare sunt grupate intr-o singura matrice numita GL_MODELVIEW. Matricea de proiectie este numita GL_PROJECTION. Matricea cu care lucram poate fi specificata prin comanda glMatrixMode(matrice).

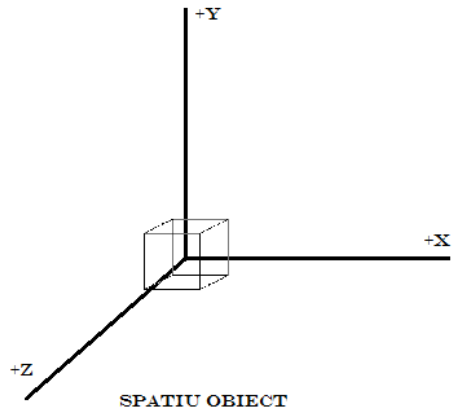
Astfel, pentru a putea desena o scena in OpenGL este necesar sa specificam 3 transformari: modelarevizualizare, proiectie si viewport.

4. Specificarea transformatelor

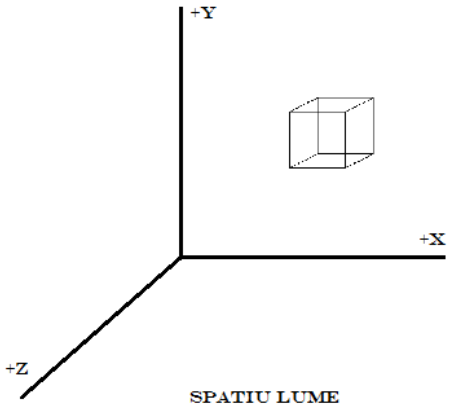
Transformarea de MODELARE pozitioneaza un obiect undeva in lume.Puteti folosi rotatii, translatii, scalari. Obiectele le puteti aseza/transforma in scena folosind:

- **glScalef(GLfloat x, GLfloat y, GLfloat z)**
- **glTranslatef(GLfloat x, GLfloat y, GLfloat z)**
- **glRotatef (GLfloat unghi, GLfloat x, GLfloat y, GLfloat z)**

Matricea de transformare din coordonate obiect in coordonate lume va fi deci dependenta doar de dimensiunea, orientarea si pozitia obiectului, dupa cum vedem in imagine:



SPATIU OBIECT



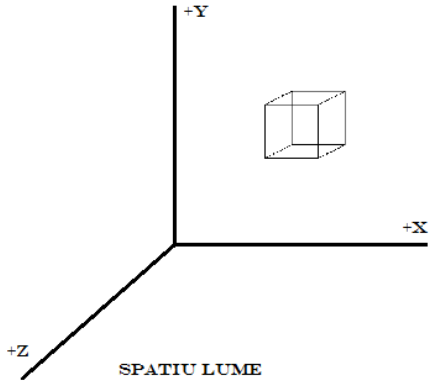
SPATIU LUME

Transformarea de VIZUALIZARE pozitioneaza observatorul si astfel perceptia utilizatorului asupra spatiului observat. Pentru a stabili transformata de vizualizare se foloseste functia:

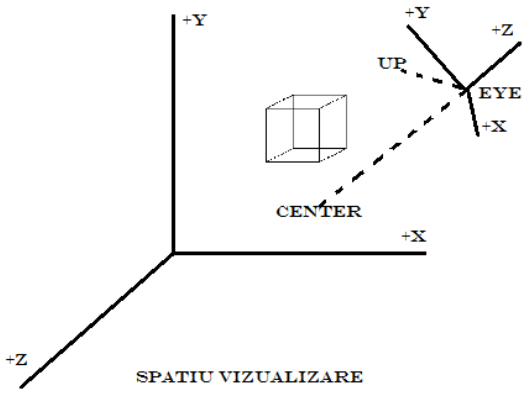
gluLookAt (GLdouble eye_x, GLdouble eye_y, GLdouble eye_z,
GLdouble center_x, GLdouble center_y, GLdouble center_z,
GLdouble up_x, GLdouble up_y, Gldouble up_z) unde:

- (eye_x, eye_y, eye_z) reprezinta pozitia observatorului
- (center_x, center_y, center_z) arata directia in care priveste acesta
- (up_x, up_y, up_z) reprezinta directia sus din planul de vizualizare

O imagine in care se prezinta transformarea de vizualizare:



SPATIU LUME



SPATIU VIZUALIZARE

Transformata de PROIECTIE face scena sa treaca dintr-un spatiu bidimensional intr-un spatiu tridimensional. Pentru a face acest lucru trebuie sa delimitam spatiul vizibil, numit **volumul de vizualizare**. OpenGL permite mai multe moduri de a face acest lucru.

Proiectia ortografica se realizeaza cu ajutorul functiei:

- **glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far)**

Volumul de vizualizare este un paralelipiped dreptunghic. Nu uitati insa ca pozitia observatorul este implicit in origine si priveste in directia negativa a axei oz. Deci doar obiectele cu z intre -near si -far vor fi vizibile.

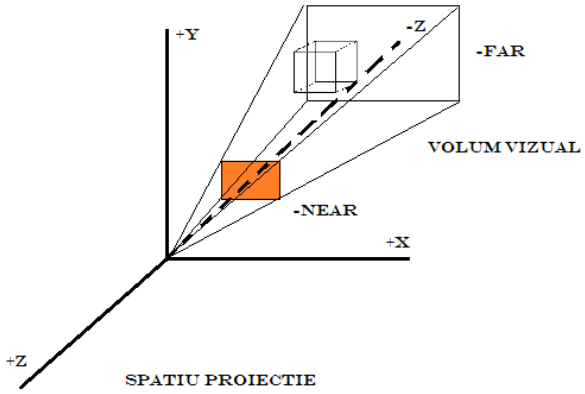
Proiectia perspectiva se poate realiza in doua moduri:

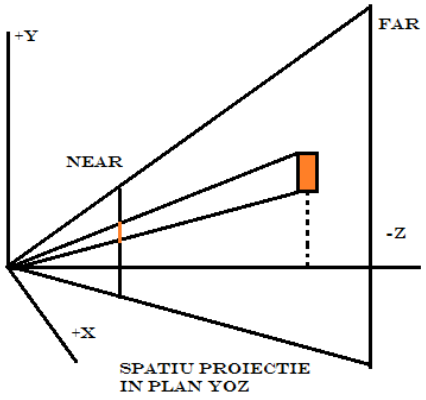
- **glFrustum(GLdouble left, GLdouble right, GLdouble bottom,GLdouble top, GLdouble near, GLdouble far)**

left, right, bottom, top se refera in acest caz la fereastra din planul apropiat. Cum stim pozitia observatorului (e cea implicita sau cea setata de noi cu gluLookAt) putem calcula volumul de vizualizare. Distantele near si far trebuie sa fie pozitive. Un alt mod de a specifica acest volum de vizualizare este cu ajutorul functiei:

- **gluPerspective(GLdouble fovy, GLdouble aspect, GLdoublenear, GLdouble far)**

Ati recunoscut deja parametrii near si far. Parametrul aspect reprezinta raportul dintre lungimea si inaltimea ferestrei in planul de aproape. Pentru a obtine o imagine nedistorsionata acest raport trebuie sa corespunda cu cel al ferestrei de afisare. Parametrul fovy este unghiul de vizualizare in planul xOz si trebuie sa fie in intervalul [0.0, 180.0].





Transformata VIEWPORT se defineste folosind functia:

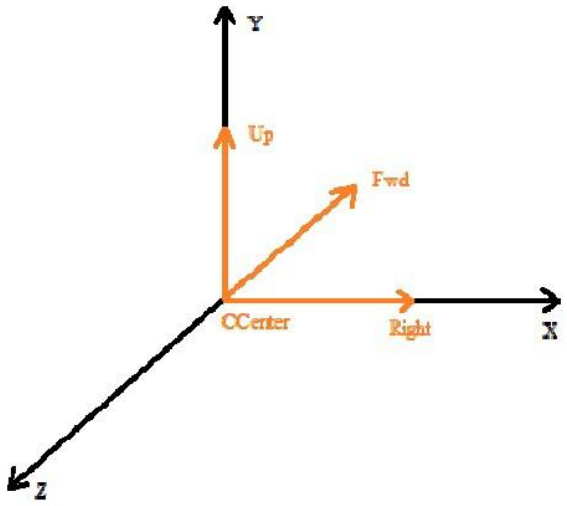
- **glViewport (GLint px, GLint py, GLsizei width, Glsizei height)** unde :

px, py reprezinta coordonatele in fereastra ale coltului stanga jos al portii. Implicit sunt 0,0. width, height sunt latimea si inaltimea portii. Valorile implicite sunt date de latimea si inaltimea ferestrei in care se afiseaza. Este o transformare fereastra poarta cu mentiunea ca z este in intervalul (0,1).

PARTEA B - Camera

1. Principii

O camera reprezinta o metoda prin care se poate specifica matricea de vizualizare a scenei intr-un mod ce pastreaza continuitate (nu ca o serie de vederi clar distincte). O camera este intotdeauna definita de 3 vectori: fata(directia de privire), dreapta si sus. Este necesar ca vectorii sa fie perpendiculari intre ei, pentru a pastra un spatiu euclidian. Acesti vectori, numiti axe, sunt usor de observat in figura de mai jos.



Astfel cand se apeleaza functia **gluLookAt()** primele trei coordonate vor fi pozitia curenta a observatorului (CCenter), urmatorul set de coordonate vor reprezenta un punct de pe semidreapta formata de CCenter si Forward (de exemplu Ccenter + Fwd), iar ultimii trei parametri ai functiei **gluLookAt()** sunt chiar componentele vectorului Up.

De exemplu daca vrem sa ne situam pe axa Z si sa ne uitam pe –Z, putem face urmatorul apel de functi:

gluLookAt(0,0,3, 0,0,0, 0,1,0), adica, din pozitia (0,0,3) ma uit catre (0,0,0), si vectorul sus coincide cu axa Oy.

Acum ca stim cum sa ne definim o camera static se pune problema mutarii camerei. Ce este vital de mentionat este ca **toate operatiile sunt vectoriale**.

2. Tipuri de camere si operatii

Problema principala in aceasta discutie o reprezinta tipul miscarii camerei. Intr-o camera de tip First Person rotatiile se fac pastrand observatorul pe loc si modificand pozitia in care se priveste.

Astfel, daca dorim sa mutam observatorul in fata va trebui sa calculam pozitia sa viitoare:

Mutare in fata: **New_Observator = Observator + Forward*factor**
Mutare la dreapta: **New_Observator = Observator + Right*factor**

Iar daca dorim sa facem o **rotatie** a camerei la dreapta(deci fata de axa OY):

New_Forward = Forward *cos(factor) + Right*sin(factor)

New_Right = Forward x Up

In camera de tip Third Person observatorul se muta in jurul unui obiect de interes, ce reprezinta intotdeauna centrul atentiei. Deci rotatiile se fac intr-un mod diferit:

Daca dorim sa facem o **rotatie** a camerei la dreapta relativ la obiectul de interes(deci fata de axa OY) vom proceda astfel: vom muta observatorul in locul obiectului de interes, vom face o rotatie simpla in **noul** spatiu iar apoi vom muta camera inapoi, pe directia **noului** vector de privire.

Distanta = distanta(Observator, Punct_de_Interes)

New_Forward = Forward *cos(factor) + Right*sin(factor)

New_Right = Forward x Up

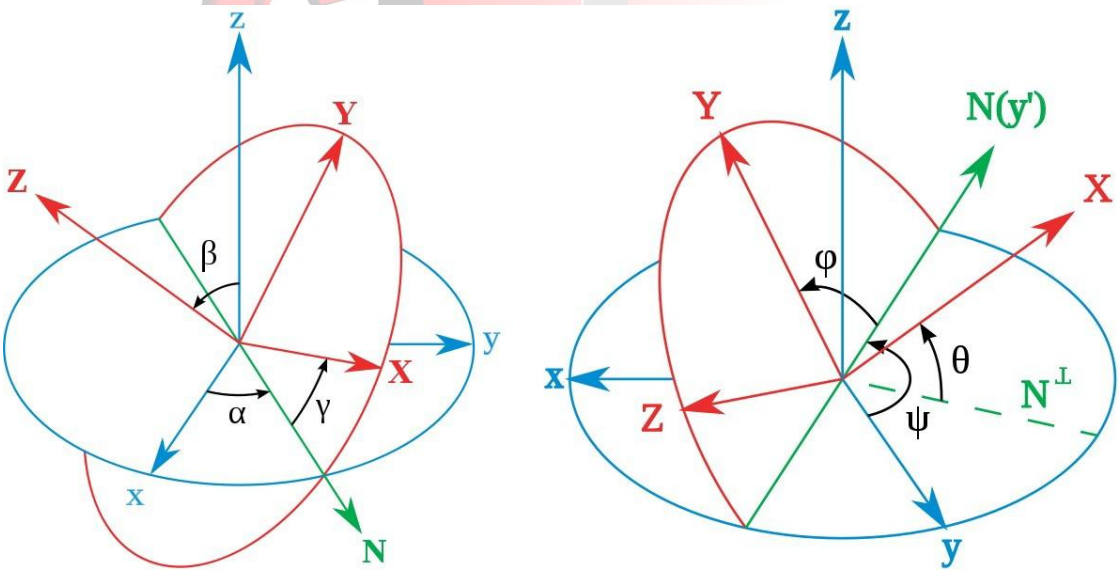
New_Observator = Observator + Distanta*Forward – Distanta*New_Forward

Mai exista si alte tipuri de camere, care nu vor fi prezentate din lipsa spatiului si a timpului.

3. Posibile probleme

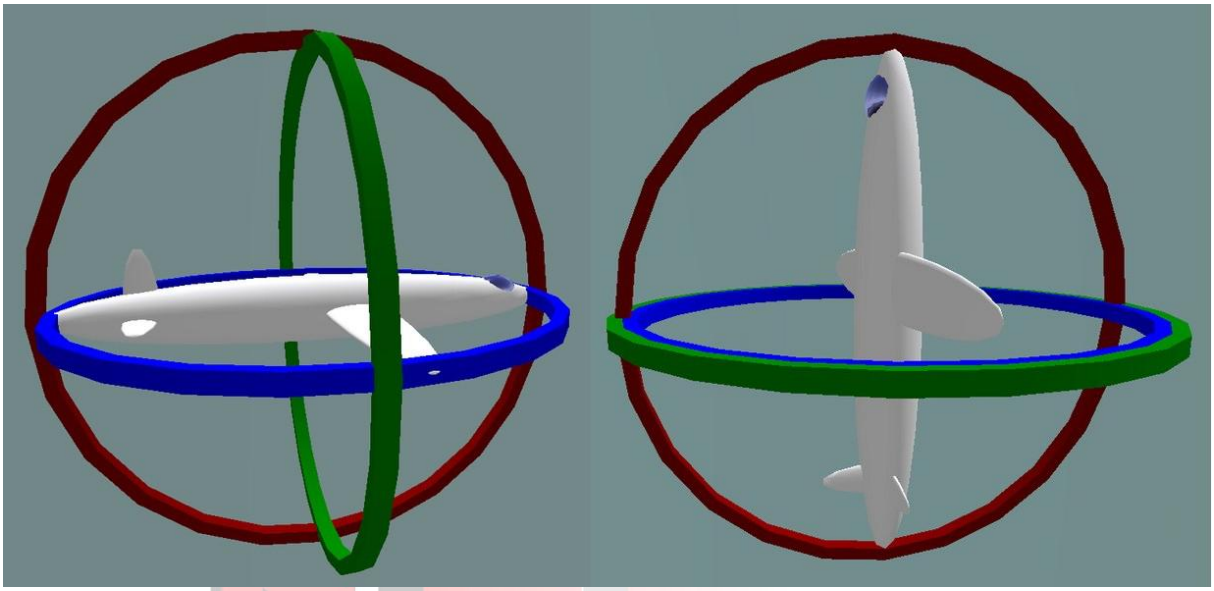
Fie ca folosim vectori sau ca folosim doar unghiuri de tip Euler este vital de precizat ca poate aparea o problema numita Gimbal Lock. Aceasta problema apare in momentul in care doua axe ale sistemului de axe se suprapun, privand sistemul de axe de un grad de libertate. Dar cum se poate intampla asa ceva cand sistemul este ortonormat?

Raspunsul este unul simplu, ***Intr-un spatiu eulerian conteaza ordinea operatiilor.*** Exista mai multe ordini de legare a rotatiilor fata de cele 3 axe, in imaginile urmatoare vor fi prezentate doar doua, xyz (intai rotatia fata de x, apoi fata de y si apoi fata de z) si zyx.



Mai mult, cum in general o rotatie nu se rezuma la un singur plan Gimbal Lock poate aparea cu usurinta daca efectuam pe rand, axial, operatiile de rotatie. De aceea trebuie pastrata consistenta sistemului, metoda variind in functie de conventia utilizata.

Gimbal lock in actiune:



Daca rotim avionul fata de axa albastra cu 90 se poate observa in figura de mai sus cum axa verde si axa albastra ajung sa se suprapuna si vor ramane suprapuse. Astfel se pierde un grad de libertate. Cum s-a ajuns la acesasta situatie? Raspuns: din cauza ca la unghiurile Euler pozitionarea se face in functie de ordinea de aplicare a rotatiilor fata de axe, deci se poate ajunge la situatia de mai sus daca printr-o rotatie fata de o axa A se ajunge peste axa B, in conditiile in care rotatia fata de A se face inainte rotatie fata de B. Dupa ce am rotit fata de A rotatia fata de B este tot doar o rotatie fata de A.