



Piscina PHP

Ziua 06

Staff Academy+Plus contact@academyplus.ro

Résumé:

Acest document este subiectul zilei 06 a piscinei PHP din cadrul Academy+Plus.

Table des matières

I	Instructiuni	2
II	Preambul	3
III	Introducere	4
IV	Instructiuni suplimentare valabile toata ziua	6
V	Exercitiu 00 : Clasa Color	8
VI	Exercitiu 01 : Clasa Vertex	10
VII	Exercitiu 02 : Clasa Vector	12
VIII	Exercitiu 03 : Clasa Matrix	15
IX	Exercitiu 04 : Clasa Camera	19
X	Exercitiu 05 : Clasele Triangle (triunghi) si Render	24
XI	Exercitiu bonus 06 : Clasa Texture	29

Chapitre I

Instructiuni

- Folositi doar aceste pagini ca referinta : nu plecati urechea la zgomotul de pe culoar.
- Subiectul se poate modifica pana cu cel mult o ora inainte de intalnire.
- Doar rezultatul muncii voastre, livrat in directorul vostru de lucru/livrare, va fi luata in considerare la corectare.
- Ca si in timpul piscinei C, exercitiile voastre vor fi corectate de catre colegii de piscina SI/SAU de catre Moulinette.
- Moulinette este foarte stricta la corectare. Ea este total automatizata. E imposibil sa discutati pe seama notarii facute de aceasta. Fiti foarte rigurosi pentru a evita surprizele.
- Utilizarea unei functii interzise se considera ca fiind o tentativa de trisare. Orice tentativa de acest fel va fi sanctionata cu nota -42.
- Exercitiile sunt foarte precis ordonate, de la cele simple la cele complexe. In niciun caz nu va fi luat in considerare un exercitiu complex rezlvat inaintea unuiuia simplu nerezolvat complet si bine.
- Nu trebuie sa lasati in directorul vostru de lucru niciun alt fisier, decat cele explicit mentionate de enuntul exercitiului.
- Aveti intrebari ? Intrebat-i pe vecinii vostri din dreapta sau din stanga.
- Manualul vostru de referinta se numeste **Google**.
- Puteti discuta pe forum. Solutia la problema voastra este deja probabil, gasita. In caz contrar, va revine sarcina sa demarati investigarea.
- Cititi cu atentie exemplele. Ele pot avea solutii ce nu sunt precizate atltfel prin enuntul subiectului ...

Chapitre II

Preambul

Nimic pe moment.

Chapitre III

Introducere

Azi incepe lunga si pasionanta vostra aventura in invatarea programarii orientate pe obiecte (OOP), in particular cu ajutorul PHP. La cursurile din semestrul trecut ati invatat bazele sintactice si semantice ale acestui limbaj si ne asumam ca sunteti capabili sa scrieti cod viabil in PHP.

Partea obiectuala a unui limbaj are ca principala utilitate inlesnirea decuparii semantice fata de cod si spre acest punct de vedere vom focaliza exercitiile de azi. Sintaxa obiectului in PHP fiind foarte simpla, exercitiile nu vor urma un model clasic de piscina, adica un exercitiu == o notare particulara. Din contra, exercitiile va vor ghida in realizarea unui mic program amuzant si va revine voua sa deduceti informatiile ce va sunt date, ce notatii se folosesc pentru rezolvarea exercitiului. Da, va trebui sa urmati rezultatul propriei reflectii si sa-l confruntati cu ideile voastre.

Pentru cei dintre voi ce au experienta in OOP, nu trebuie sa omiteti ca videoclipul zilei nu trateaza decat o parte din OOP in PHP, si anume programarea modulara. Bineinteles, restul notarilor vor fi abordate maine. In ceea ce priveste exercitiile de azi, TREBUIE sa folositi ceea ce ati vazut pana acuma. Fara exceptii de mosteniri, proprietati si alte interfete. Astea vor fi pentru maine si poimaine.

Pe parcursul acestei piscine precum si in proiectul urmator, veti avea propriul lot de site-uri si alte aplicatii web de realizat. Atunci de ce nu e mai bine ca azi sa faceti altceva, ca de exemplu grafica 3D? ati realizat un Raytracer nu de multa vreme, iar acum veti face un Rasterizer.

Sunt mai multe lucruri de spus inainte de a incepe. In primul rand aceasta zi a piscinei va fi evaluata in mod unic prin corectarea peer. Asa ca nicio grija in privinta moulinette-i. Toate rezultatele exemplelor nu trebuie respectate intocmai in privinta dimensiunii paginii ori a pixelilor, chiar daca prin enunt sunteti incurajati la un anumit format pe care va invit sa-l respectati. Ceea ce intereseaza este rezultatul si organizarea codului.

Apoi, va trebui sa efectuati activitati de studiu individual pentru a putea aborda exercitiile zilei. Multe notiuni tehnice legate de PHP sunt descrise in videoclipurile ce insotesc subiectul, precum si vocabularul si notiunile legate de grafica 3D trebuie sa le descoperiti singuri.

În al treilea rând, contrar celor ce pot fi presupuse conform enunțului, noțiunile de matematică ce trebuie cunoscute pentru a realiza această zi a piscinei, sunt minime. Dacă va pierdeți în speculații matematice înseamnă că ați luat-o pe un drum gresit. Nu vi se cere să știți să creați o matrice complicată cu toate explicațiile aferente, demonstrații etc. Vi se cere doar să știți să manipulați un tablou cu două dimensiuni în PHP și să faceți adunări și înmulțiri pentru aceste cazuri.

Și în ultimul rând, dacă ceva vi se pare foarte vag, nu foarte explicit sau subiectul e dezbătut într-un enunț, există două soluții : fie nu ați înțeles, fie trebuie să luați o decizie pe care va trebui să o apărați. Baremul ține cont de asta și țineți minte că chiar și o decizie bine apărată poate fi greșită.

În ultimul rând, vi s-a cerut să știți cum funcționează **OpenGL** ? Ei bine decoperiți voi împreună, azi.

Chapitre IV

Instructiuni suplimentare valabile toata ziua


Instructiunile urmatoare sunt valabile pentru toate exercitiile zilei. Descrierea fiecarui exercitiu va va reaminti. Daca aveti dubii, nu exista niciodata exceptie. Aceste instructiuni se aplica sistematic la ce poate fi interpretat prin lectura. Nerespectarea uneia dintre acestea duce la penalizarea cu 0 a exercitiului si la incheierea evaluarii muncii voastre cu ajutorul baremului.

- Scrieti o singura clasa intr-un fisier.
- Un fisier ce contine definitia unei clase nu trebuie sa mai contina nimic altceva, in afara de `require` sau `require_once` cand este necesar.
- Un fisier ce contine o clasa trebuie **INTOTDEAUNA** denumit astfel `ClassName.class.php`.
- O clasa trebuie **INTOTDEAUNA** sa fie insotita de un fisier de documentatie a carui nume trebuie sa fie **INTOTDEAUNA** de forma `ClassName.doc.txt`.
- Documentatia unei clase trebuie **INTOTDEAUNA** sa fie utila si corespunzatoare implementarii. O clasa fara documentatie nu foloseste la nimic si o documentatie incerta, perimata sau incompleta nu e folositoare. Nu explicati functionalitatea interna a claselor. Documentatia e utila pentru a sti cum se foloseste clasa si nu cum ati implementat-o. Acest ultim lucru este evident din scrierea codului.
- O clasa trebuie **INTOTDEAUNA** sa aiba un atribut static boolean `verbose` ce permite afisarea informatiilor practice pentru debug si la sustinere.
- O clasa trebuie **INTOTDEAUNA** sa aiba o metoda staica `doc` ce returneaza continutul fisierului de documentatie asociat unei clase, sub forma unui sir de caractere.
- Codul pe care il veti scrie pentru fiecare exercitiu se va adauga celui scris pentru exercitiile precedente pentru a tinde spre un program complet. Pentru a usura dezvoltarea si sustinerea, livrarea fiecarui exercitiu trebuie sa contina o copie a fisierelor livrate in exercitiul anterior. In acelasi timp, anumite exercitii va ofera libertatea sa modificati codul claselor pe care le-ati realizat in exercitiile precedente. In acest caz, exercitiul curent va contine codul modificat in locul versiunii exercitiului precedent si asa in continuare. In incheiere, aveti posibilitatea sa va adaugati propriile clase de la un moment dat, fisierele corespunzatoare trebuind

sa fie prezente dintr-un director de livrare in celalalt. Nu exista nicio capcana.

Chapitre V

Exercitiu 00 : Clasa Color

	Exercice : 00
Clasa Color	
Dossier de rendu : <i>ex00/</i>	
Fichiers à rendre : <code>Color.class.php</code> , <code>Color.doc.txt</code>	
Fonctions Autorisées : Tot ce a fost deja vazut de la inceputul piscinei si toate bibliotecile standard PHP.	
Remarques : n/a	

Sa incepem cu o mica clasa foarte simpla : clasa `Color`. Aceasta clasa ne va permite sa reprezentam culorile si sa facem cateva operatiuni simple pe componentele sale.

- Clasa `Color` trebuie sa aiba trei attribute publice intregi `red`, `green` si `blue` care vor servi la reprezentarea componentelor unei culori.
- Constructorul clasei ia ca parametru un tablou. O instanta trebuie sa poata fi construita, fie prin transmiterea unei valori pentru cheia `'rgb'` ce va fi descompusa in trei componente rosu, verde si albastru, fie prin transmiterea unei valori pentru cheile `'red'`, `'green'` si `'blue'` ce reprezinta direct cele trei componente. Fiecare din valorile pentru cele patru chei posibile vor fi convertite in intregi inainte de utilizare.
- Clasa `Color` trebuie sa aiba o metoda `__toString`. Vedeti exemplul pentru a deduce formatul.
- Clasa trebuie sa aiba un atribut **verbose** **statique** de tip boolean ce permite controlul afisajului legat de utilizarea clasei. Acest atribut este setat initial pe valoarea `False`.
- Daca si numai daca atributul static **verbose** este adevarat, atunci constructorul si destructorul clasei produc un rezultat. Vezi rezultatul exemplului pentru deducerea formatului.
- Clasa trebuie aiba o metoda statica `doc` ce returneaza o documentatie scurta a clasei sub forma unui sir de caractere. Continutul documentatiei trebuie citita din fisierul `Color.doc.txt`. Vezi rezultatul exemplului pentru deducerea formatarei


documentatiei si continutul fisierului.

- Clasa trebuie sa aiba o metoda **add** ce permite adaugarea componentelor instantei curente la componentele unei instante transmise ca parametru. Culoarea rezultata este o noua instanta a clasei.
- Clasa trebuie sa aiba o metoda **sub** ce permite scaderea componentelor unei instante transmise ca parametru din componentele instantei curente. Culoarea rezultata este o noua instanta a clasei.
- Clasa trebuie sa aiba o metoda **mult** ce permite inmultirea componentelor instantei curente cu un factor transmis ca parametru. Culoarea rezultata este o noua instanta a clasei.

Pentru a rezuma, trebuie sa scrieti o clasa **Color** intr-un fisier numit **Color.class.php** ce permite unui script **main_00.php** dat in anexa acestui subiect sa genereze rezultatul prezentat in fisierul **main_00.out**, de asemenea dat in anexa subiectului. Documentatia clasei se va afla in fisierul denumit **Color.doc.txt**, fisier ce trebuie livrat de asemenea.

Chapitre VI

Exercitiu 01 : Clasa Vertex

	Exercice : 01
Clasa Vertex	
Dossier de rendu : <i>ex01/</i>	
Fichiers à rendre : <code>Vertex.class.php</code> , <code>Vertex.doc.txt</code>	
Fonctions Autorisées : Tot ce a fost vazut de lainceputul acestei piscine si toata biblioteca standard PHP.	
Remarques : n/a	

In continuare vom continua cu reprezentarea unui punct in spatiu : "vertex". Vom reprezenta un vertex pe baza a cinci caracteristici :

- Abscisa sa **x**
- Ordonata sa **y**
- Adancimea sa **z**
- O coordonata noua si nelinistitoare **w**. Cautati pe **Google** "coordonate omogene". In practica, aceasta coordonata are valoarea obisnuita 1.0 si simplifica calculul matriceal in exercitiile urmatoare.
- O culoare reprezentata sub forma unei instante a clasei **Color** din exercitiul precedent.

Clasa Vertex este foarte simpla. Inca nu se pune problema intelegerii modului cum sunt date coordonatele unui vertex. Aceasta clasa are coordonatele incapsulate si ofera accesul la citirea si scrierea atributelor coespunzatoare.


- Clasa **Vertex** sa aiba attribute private pentru reprezentarea celor cinci caracteristici precedente. Va reamintesc ca, prin conventie, identificatorii atributelor private incep cu caracterul '**_**' (underscore).
- Culoarea vertex-ului va fi intotdeauna o instanta a clasei **Color** din exercitiul precedent.

- Clasa **Vertex** trebuie sa permita accesul la citirea si scrierea celor cinci atribute.
- Constructorul clasei ia ca parametru un tablou. Cheile asteptate sunt urmatoarele :
 - 'x' : abscisa vertex-ului, obligatorie.
 - 'y' : ordonata vertex-ului, obligatorie.
 - 'z' : adancimea vertex-ului, obligatorie.
 - 'w' : optionala, valoarea implicita 1.0.
 - 'color' : optionnena, valoarea unei instante noi de culoare alba, implicit.
- Clasa trebuie sa contina un atribut **verbose static** de tip boolean ce permite controlul afisarii legat de folosirea clasei. Acest atribut este initializat la valoarea **False**.
- Clasa **Vertex** trebuie sa aiba o metoda **__toString**. Vedeti rezultatul exemplului de deducere a formatatii. Notati ca valoarea culorii vertex-ului nu este nu este adaugata la sir decat daca si numai daca atributul static **verbose** este adevarat.
- Daca si numai daca atributul static **verbose** este adevarat, atunci constructorul si destructorul clasei vor produce un rezultat. Vedeti rezultatul din exemplul pentru deducerea formatarii.
- Clasa trebuie sa aiba o metoda statica **doc** ce returneaza o documentatie scurta a clasei sub forma unui sir de caractere. Continutul documentatiei trebuie sa fie citita din fisierul **Vertex.doc.txt**. Din acest exercitiu documetatie este lasata la latitudinea voastra. Singura obligatie este ca aceasta documetatie sa fie pertinena si utila. Imaginati-va un programator ce vrea sa foloseasca clasa voastra si care nu are aceasta documentatie pentru a intelege functionarea sa. Acest aspect va fi verificat la sustinere.

Pentru a rezuma, trebuie sa scrieti clasa **Vertex** intr-un fisier denumit **Vertex.class.php** ce p[ermite unui script **main_01.php** dat in anexa acestui subiect sa produca un rezultat prezent in fisierul **main_01.out** de asemenea dat in anexa acestui subiect. Documentatia clasei se va gasi intr-un fisier denumit **Vertex.doc.txt**, ce de asemenea trebuie livrat.

Chapitre VII

Exercitiu 02 : Clasa Vector

	Exercice : 02
Clasa Vector	
Dossier de rendu : <i>ex02/</i>	
Fichiers à rendre : <code>Vector.class.php</code> , <code>Vector.doc.txt</code>	
Fonctions Autorisées : Tot ce s-a vazut de la inceputul piscinei si toata biblioteca standard PHP.	
Remarques : n/a	

Acum ca avem nodurile, acestea pot fi plasate in spatiu si chiar sa li se atribuiie o culoare simpla. E frumos, dar vectorii sunt de asemenea practici pentru reprezentarea directiilor sau a deplasarilor, de exemplu.

Clasa `Vector` ne va permite introducerea unei conventii. Pentru orientarea 3D, exista optiunea unui reper numit "mana stanga" sau "mana dreapta". Cautati pe **Google** semnificatia acestora si considerati ca incepand de acum, vom lucra raportat la "mana dreapta".

Daca aveti cunostinte solide despre coordonatele omogene in pentru realizarea clasei `Vertex`, ati descoperit ca aceasta reprezentare permite simplificarea masiva a anumitor calcule. Vom folosi de asemenea un sistem de coordonate omogene pentru vectori, dar de data asta, componenta `w` va avea intotdeauna valoarea 0.0 si va fi considerata ca o componenta a unui vector, pe parcursul calculelor, la fel ca `x`, `y` si `z`.

Un vector este reprezentat prin urmatoarele caracteristici :

- Marimea sa pe `x`
- Marimea sa pe `y`
- Marimea sa pe `z`
- Coordonata `w`

Clasa `Vector` este putin mai complexa decat clasa `Vertex`. Unele metode vor necesita

calcule foarte simple ce sunt predate in primi ani de liceu. Internetul este plin de tutoriale despre vectori si nu aveti decat sa le adaptati pentru scrierea acestei clase.

- Clasa **Vector** trebuie sa aiba attribute private pentru a reprezenta cele patru caracteristici descrise mai sus. Va reamintesc ca prin conventie, identificatorii atributelor private sunt precedate de `'_'` (underscore).
- Clasa **Vector** trebuie sa ofere acces doar pentru citirea celor patru attribute.
- Constructorul clasei ia ca parametru un tablou. Cheile asteptate sunt urmatoarele :
'dest' : vertex (punctul) de destinatie al vectorului, obligatoriu.
'orig' : vertex (punctul) de origine al vectorului, optional, avand valoarea implicita a unei instante noi de vertex `x=0, y=0, z=0, w=1`.
- Clasa trebuie sa contina atributul **verbose static** de tip boolean ce permite controlul afisarii legat de utilizarea clasei. Acestui atribut i se atribuie initial valoarea **False**.
- Clasa **Vector** trebuie sa aiba o metoda `__toString`. Vezi rezultatul de la exemplul pentru deducerea formatarei.
- Daca si numai daca atributul static **verbose** este adevarat, atunci constructorul si destructorul clasei produc un rezultat. Vezi rezultatul de la exemplul pentru deducerea formatarei.
- Clasa trebuie sa ai ba o metoda statica **doc** ce returneaza documentatie scurta a clasei sub forma unui sir de caractere. Continutul documentaiei trebuie citita dintr-un fisier `Vector.doc.txt` si va este lasata la dispozitie, ca in exercitiul precedent.
- O metoda a clasei **Vector** nu trebuie niciodata sa modifice instanta curenta. Acest comportament este intarit de absenta setter-elor. Odata un vector instantiat, starea lui este definitiva.
- Clasa **Vector** trebuie sa aiba cel putin urmatoarele metode publice. Existenta metodelor private nu va priveste decat pe voi. E vorba de adunari si inmultiri.

float magnitude() : returneaza lungimea (sau "norma") vectorului.

Vector normalize() : returneaza vectorul normalizat. Daca vectorul este deja normalizat, returneaza o copie noua a vectorului.

Vector add(Vector \$rhs) : returneaza vectorul suma a doi vectori.

Vector sub(Vector \$rhs) : returneaza vectorul diferenta a doi vectori.

Vector opposite() : returneaza vectorul opus.

Vector scalarProduct(\$k) : returneaza produsul dintre un vector si un scalar.

float dotProduct(Vector \$rhs) : returneaza produsul scalar a doi vectori.


float cos(Vector \$rhs) : returneaza cosinusul unghiului dintre doi vectori.

Vector crossProduct(Vector \$rhs) : returneaza produsul vectorial a doi vectori (mana dreapta!)

Pentru a rezuma, trebuie sa scrieti o clasa **Vector** intr-un fisier ce trebuie livrat, denumit **Vector.class.php** ce permite script-ului **main_02.php** dat in anexa acestui subiect genereze rezultatul prezent in fisierul **main_02.out**, de asemenea dat in anexa. Documentatia clasei se va afla intr-un fisier denumit **Vector.doc.txt**, fisier ce va trebui de asemenea livrat.

Chapitre VIII

Exercitiu 03 : Clasa Matrix

	Exercice : 03
Clasa Matrix	
Dossier de rendu : <i>ex03/</i>	
Fichiers à rendre : <code>Matrix.class.php</code> , <code>Matrix.doc.txt</code>	
Fonctions Autorisées : Tot ce a fost vazut de la inceputul piscinei si toata biblioteca standard PHP.	
Remarques : n/a	

Incepand cu acest exercitiu, implementarea claselor va deveni tot mai libera. Asta inseamna ca puteti folosi attributele si metodele pe care le considerati adecvate, atata vreme cat clasa raspunde exigentelor cerute. Fiti constienti de vizibilitatea mebrilor. Un atribut sau o metoda publice care n-ar trebui sa fie constituie o eroare.

Folosind nodurile se pot pozitiona punctele in spatiu. Cu vectorii se poate reprezenta deplasarea in spatiu. Cu ajutorul matricilor, se vor putea face transformari, ca aplicarea unei schimbări de scara, o translatie sau o rotatie in raport cu una sau mai multe noduri. De obicei mai multe, altfel jocurile video fiind plictisitoare.

Internetul este plin de tutoriale in legatura cu matricile, si mai ales pentru domeniul 3D. E inutil sa invatati toata teoria matematica legata de matrici, ceea ce trebuie sa stim aici este : ce este o matrice 4x4 ? Si inmultirea a doua matrici ?

În 3D, o matrice 4x4 poate fi văzută ca o reprezentare a unui reper ortonormal, și anume 3 vectori pentru cele 3 axe și un vertex pentru originea reperului. Deci, vom reprezenta toate matricile, indiferent de utilitatea lor :

```

    vtcX vtcY vtcZ vtx0
x      .      .      .      .
y      .      .      .      .
z      .      .      .      .
w      .      .      .      .

```

Sub ochi voștri uimiți, descoperiți mai jos matricea ce reprezintă reperul fiecărui sistem de axe, vectorul unitate pe direcția sa și a cărui origine este vertex-ul origine :

```

    vtcX vtcY vtcZ vtx0
x  1.0  0.0  0.0  0.0
y  0.0  1.0  0.0  0.0
z  0.0  0.0  1.0  0.0
w  0.0  0.0  0.0  1.0

```

Dacă din întâmplare ați mai întâlnit o matrice în viața voastră, veți recunoaște bineînțeles matricea identitate. Dacă nu, bună dimineața.

Luati un moment de respirație. Dacă la un moment dat acest exercițiu cu matrici vi se pare imposibil de înțeles, este pentru că vă concentrați pe aspectul matematic al problemei în loc să vă concentrați pe aspectul informatic. O matrice este un tablou și nimic mai mult sau mai puțin. În acest exercițiu va trebui să înmulțiți două matrici între ele și să înmulțiți o matrice și un vertex. Algoritmii pentru aceste operații sunt disponibili pe internet și constau în doar în a aduna sau înmulți două celule ale unui tablou, într-o anumită ordine. Nu considerați necesar să înțelegeți de ce și cum funcționează aceste calcule. Aplicați-le doar.

Matricile noastre sunt formate **INTOTDEAUNA** din patru linii și patru coloane. Puteti uza și abuza de această caracteristică în calculele voastre. Pentru a simplifica și mai mult problema noastră, clasa **Matrix** Nu va reprezenta orice matrice 4x4. Ne vom concentra pe matricile 4x4 următoare :

- Matricea identitate
- Matricea de translație ("translate")
- Matricea de schimbare a scării ("scale")
- Matricea de rotație ("rotate")
- Matricea proiecție ("project")

Dacă vreți să știți mai multe despre aceste matrici, cautați *recherchez les "3D transformation matrix"* pe **Google**. Dacă nu va pasa, mulțumiți-vă să cautați cum se construiește.

Matricea proiecție e cea mai dificil de calculat. Această [documentation](#) este perfectă pentru tine. Va lăsa să alegeți între posibilitatea de a copia matricea sau de a înțelege ce reprezintă. Vom studia componenta sa în detaliu în exercitiul următor. Celelalte matrici sunt prea simple pentru a fi explicate. Chiar și așa vă încurajez să partajați documentația

si analizati problema.

Sa definim acum o clasa **Matrix** pentru a reprezenta matrici 4x4. Matricile noastre vor avea intotdeauna dimensiunea 4x4, fara exceptie.

- Mai multe comportamente ale clasei **Matrix** sunt deduse din cod si din rezultatul ce urmeaza acestor explicatii. Restul este la dispozitia voastra.
- Clasa **Matrix** trebuie sa aiba sapte constante : **IDENTITY**, **SCALE**, **RX**, **RY**, **RZ**, **TRANSLATION** si **PROJECTION**.
- Constructorul clasei asteapta un tablou. Cheile asteptate sunt urmatoarele :
'preset' : tipul matricei de construit, obligatorie. Valoarea trebuie sa fie una din constantele clasei precedente.
'scale' : factorul de scara, obligatorie cand **'preset'** ia valoarea **SCALE**.
'angle' : unghi de rotatie in radiani, obligatorie cand **'preset'** ia valoarea **RX**, **RY** sau **RZ**.
'vtx' : vector de translatie, obligatorie cand **'preset'** ia valoarea **TRANSLATION**.
'fov' : camp de vedere a proiectiei in grade, obligatorie cand **'preset'** ia valoarea **PROJECTION**.
'ratio' : rata imaginii proiectate, obligatorie cand **'preset'** ia valoarea **PROJECTION**.
'near' : plan de sectionare aproape de proiectie, obligatorie cand **'preset'** ia valoarea **PROJECTION**.
'far' : plan de sectionare indepartat de proiectie, obligatorie cand **'preset'** ia valoarea **PROJECTION**.
- Clasa trebuie sa contina un atribut **verbose static** de tip boolean ce permite controlul afisarii referitor la utilizarea clasei. Acest atribut este initializat la valoarea **False**.
- Clasa **Matrix** trebuie sa aiba o metoda **__toString**. Vedeti rezultatul din exemplul pentru deducerea formatarei.
- Daca si numai daca atributul static **verbose** este adevarat, atunci constructorul si destructorul clasei vor produce un rezultat. Vedeti rezultatul din exemplul pentru deducerea formatarei.
- Clasa trebuie sa aiba o metoda statica **doc** ce returneaza o documentatie scurta a clasei sub forma unui sir de caractere. Continutul documentatiei trebuie sa fie citit din fisierul **Matrix.doc.txt** si este la dispozitia voastra, ca in exercitiul precedent.
- O metoda a clasei **Matrix** nu trebuie niciodata sa modifice instanta curenta. Odata matricea instantiata, starea sa e definitiva.
- Organizarea si codul clasei **Matrix** sunt la dispozitia voastra. Fiti destepti, eficienti

si ordonati. Fiti atenti cu vizibilitatea membrilor.

- Clasa **Matrix** trebuie sa aiba urmatoarele metode. Daca vi se pare greu de codat e posibil sa fiti pe o cale gresita.


Matrix mult(Matrix \$rhs) : returneaza o matrice noua, ca rezultat a inmultirii a doua matrici.

Vertex transformVertex(Vertex \$vtx) : returneaza un nou vertex, rezultat al transformarii vertex-ului de catre matrice.

Pentru a rezuma, trebuie sa scrieti oclasa **Matrix** intr-un fisier denumit **Matrix.class.php** ce permite script-ului **main_03.php** dat in anexa acestui subiect sa genereze rezultatul prezent in fisierul **main_03.out** din aceeași anexa. Documentatia clasei va fi intr-un fisier denumit **Matrix.doc.txt**, ce trebuie de asemenea livrat.

Chapitre IX

Exercitiu 04 : Clasa Camera

	Exercice : 04
Clasa Camera	
Dossier de rendu : <i>ex04/</i>	
Fichiers à rendre : <code>Camera.class.php</code> , <code>Camera.doc.txt</code> , <code>*.class.php</code> , <code>*.doc.txt</code>	
Fonctions Autorisées : Tot ce a fost vazut de la inceputul piscinei si toata biblioteca standard PHP.	
Remarques : n/a	

Incepand cu acest exercitiu, sunteti liberi sa modificati si sa completati clasele exercitiilor precedente daca considerati necesar acest lucru. Puteti, in aceeași masura sa adaugati clase noi. In același timp va trebui sa respectati regula unei singure clase per fisier, sa respectati conventiile de nume ale fisierelor si sa livrati fisierul de documentatie pentru fiecare clasa. Fiti atenti la vizibilitatea membrilor claselor.

Concluzii : Sunteti capabili sa modificati formele in 3D cu ajutorul nodurilor. Cu ajutorul vectorilor si matricilor se pot transforma aceste forme (schimbarea dimensiunii, deplasarea, rotirea). Ne mai lipseste ceva important : camera pentru a "vedea" scenele.

O imagine 3D este rezultatul transformarii succesive a nodurilor de la un reper la altul. E vorba de "pipeline de transformare" pentru a trece de la o scena la o imagine afisabila. Pipeline-ul unei biblioteci precum `OpenGL` fiind foarte complex, vom folosi acest pipeline doar pentru nevoile noastre :

```

Local vertex
|
| Model matrix
V
World vertex
|
| View matrix
V
Cam vertex
|
| Projection matrix
V
NDC vertex
|
| Transformation (no matrix involved)
V
Screen vertex (pixel)

```

În general, se combina prin înmulțire matricile scalei de translație și de rotație a unui ansamblu de noduri într-o matrice rezultantă numită "model matrix". Această matrice permite transformarea unui obiect de la reperul său local spre un reper "monde". Asta înseamnă așadar plasarea obiectului acolo unde dorim în scenă, cu orientarea și dimensiunea dorite.

Pentru a "vedea" lumea noastră, este necesară plasarea unei camere undeva, cu orientarea dorită. Camera va avea deci, coordonate în reperul "monde". Pentru determinarea poziției obiectelor în reperul "camera" (ceea ce "vede" camera), trebuie calculată o matrice care permite trecerea din reperul "monde" spre reperul "camera". Această matrice se numește în general "view matrix".

Cum se calculează "view matrix"? Bună întrebare, care necesită puțină logică. Să începem caracterizarea unei camere :

- Poziția sa în "monde" (lume) cu un vertex.
- Orientarea sa în "monde" (lume) cu o matrice de rotație, adică spre "ou la camera regarde" (unde privește camera).

Poziția camerei în lume ne permite să calculăm o matrice de translație. Avem deci o cameră definită de o matrice de translație și o matrice de rotație ce permit situarea camerei în reperul "monde". Acum că avem aceste informații, calculați "view matrix" fiind posibil prin calcularea matricei de transformare inverse. acordati-va timp pentru cautarea pe net.

- Fie T matricea de translație construită pentru vectorul v . Se calculează $oppv$ vectorul opus vectorului v și se construiește o matrice de translație inversă tT .
- Fie matricea de rotație R . Se trasează o axă de simetrie diagonală (coordonata x devine y în tablou și vice-versa) și se obține o matrice tR .
- În ultima etapă se înmulțesc $tR \rightarrow mult(tT)$ și paf, asta e o "view matrix" pentru camera.

În acest stadiu camera poate "vedea". Dar coordonatele sunt întotdeauna în 3 dimen-

siuni, astfel incat ceea ce dorim este o imagine in 2 dimensiuni. Scena privita trebuie sa fie "proiectata" pe un plan. Pentru asta su va folosi matricea de proiectie a exercitiului precedent, definita de caracteristicile urmatoare :

ratio : Rata imaginii finale, adica raportul dintre lungimea si inaltimea imaginii. Ati auzit vorbindu-se de formatele 4/3 sau 16/9? Ei bine acestea sunt ratele imaginii.

fov : Campul de vedere al imaginii proiectate, in grade. Cautati "field of view 3D" pe **Google** daca vreti sa stiti mai multe. In practica 60 de grade este o valoare arbitrara corecta. Ea corespunde aproximativ cu unghiul dintre nasul vostru si cele doua margini ale ecranului din acel momemt. Aceasta valoare va putea fi modificata pentru a vedea o portiune mai mare sau mai mica a scenei.

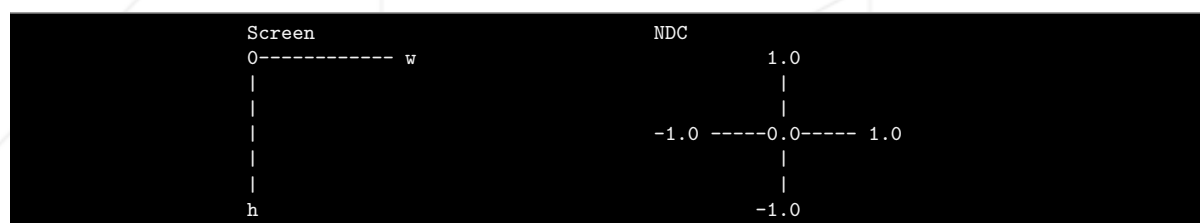
near : Planul de clipping apropiat. Notiune putin mai complexa, este distanta camerei de la care un obiect este vazut. **Google** va explica.

far : Planul de clipping indepartat. Pentru cei pe care ii intereseaza, aceste doua planuri permit calcularea z-buffer-ului unei scene, notiune din afara scopului acestui exercitiu.

Ceea ce e sigur este ca ati inteles modul in care este construita matricea de proiectie sau nu, un vertex (nod) in coordonate camera transformat de aceasta matrice va fi acum in 2D! Si cine spune 2D afirma ca se poate face o imagine!

Nu am terminat inca. Numelereperului in care se gaseste un vertex (nod) dupa transformarea de catre matricea de proiectie poarta numele ciudat "normalized device coordinates", sau "NDC" pentru amicii (**Google** it now!). Acest reper corespunde unei imagini al carei centru are coordonatele 0.0, 0.0, coltul inferior stang -1.0, -1.0 si coltul superior dreapta 1.0, 1.0. x-ii si y-ii fiecarui vertex (nod) sunt deci cuprinsi intre -1.0 si 1.0 (z-ul corespunde pozitiei vertex-ului din z-buffer (inutil pentru azi). La ce serveste asta? Ei bine e foarte simplu pentru generarea unei imagini de dimensiunea pe care o doriti astfel ca ea sa respecte proportiile (aspect ratio)! Cunoasteti rezolutia din jocul video? Ei bine asta e ce avem de facut pentru a o schimba.

Cum se trece de la un vertex NDC la un vertex ecran (un pixel)? Réfléchissez!



Va trebui sa scrieti codul chiar acum. Din fericire codul este mult mai scurt ca acest text.

- Constructorul clasei ia ca parametru un tablou. Cheile sunt urmatoarele :

'origin' : Vertex (nod) ce pozitioneaza camera in reperul monde. Datorita lui se poate calcula un vector si apoi o matrice de translatie.

'orientation' : Matrice de rotatie ce orienteaza camera in reperul monde.

'width' : Latimea in pixeli a imaginii dorite. Folosita la calcularea aspect ratio. Incompatibila cu cheia **'ratio'**.

'height' : Inaltimea in pixeli a imaginii dorite. Folosita la calcularea aspect ratio. Incompatibila cu cheia **'ratio'**.

'ratio' : Aspect ratio-ul imaginii. Incompatibila cu cheia **'width'** si **'height'**.

'fov' : Campul de vedere al imaginii proiectate, in grade.

'near' : Planul de clipping apropiat.

'far' : Planul de clipping departat.

- Clasa trebuie sa contina un atribut **verbose static** de tip boolean ce permite controlul afisarilor legate de utilizarea clase. Acest atribut este initializat cu valoarea **False**.
- Clasa **Camera** trebuie sa aiba o metoda **__toString**. Vezi rezultatul din exemplul pentru deducerea formatarei.
- Daca si numai daca atributul static **verbose** este adevarat, atunci constructorul si destructorul clasei produc un rezultat. Vedeti rezultatul exemplului pentru deducerea formatarei.
- Clasa trebuie sa aiba o metoda statica **doc** ce returneaza o documentatie scurta a clasei sub forma unui sir de caractere. Continutul documentatiei trebuie sa fie citita din fisierul **Camera.doc.txt** si este lasat la latitudinea voastra, ca in exercitiul precedent.
- Organizarea si codul clasei **Camera** sunt la dispozitia voastra.

- Clasa `Camera` trebuie sa aiba metoda urmatoare :


`Vertex watchVertex(Vertex $worldVertex)` : Transforma un vertex in coordonate "monde" intr-un vertex in coordonate "ecran" (in pixeli).

Aveti in prezent un pipeline de transformare complet ! Felicitari !

Pentru a rezuma, trebuie sa scrieti o clasa `Camera` intr-un fisier numit `Camera.class.php` ce permite script-ului `main_04.php` dat in anexele acestui subiect sa genereze rezultatul prezentat in fisierul `main_04.out`, de asemenea din anexa. Documentatia clasei va fi plasata intr-un fisier numit `Camera.doc.txt`, ce trebuie livrat de asemenea. Aveti dreptul sa modificati clasele precedente sau sa le completati daca considerati necesar. Trebuie sa livrati cate un fisier de documentatie a claselor suplimentare si aceste clase trebuie sa aiba o metoda statica `doc` precum celelalte.

Chapitre X

Exercitiu 05 : Clasele Triangle (triunghi) si Render

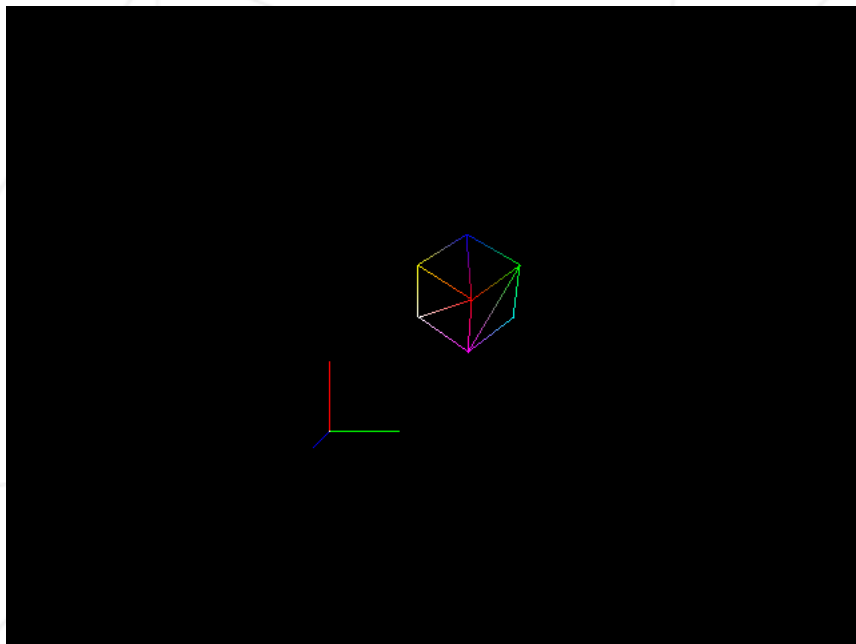
	Exercice : 05
Clasele Triangle (triunghi) si Render	
Dossier de rendu : <i>ex05/</i>	
Fichiers à rendre : <code>Render.class.php</code> , <code>Render.doc.txt</code> , <code>Triangle.class.php</code> , <code>Triangle.doc.txt</code> , <code>*.class.php</code> , <code>*.doc.txt</code>	
Fonctions Autorisées : Tot ce s-a vazut pana acuma de la inceputul piscinei, toata biblioteca standard PHP si biblioteca GD.	
Remarques : n/a	

In acest exercitiu vom genera in sfarsito imagine a scenei. Scopul eate de a putea randa in 3 moduri :

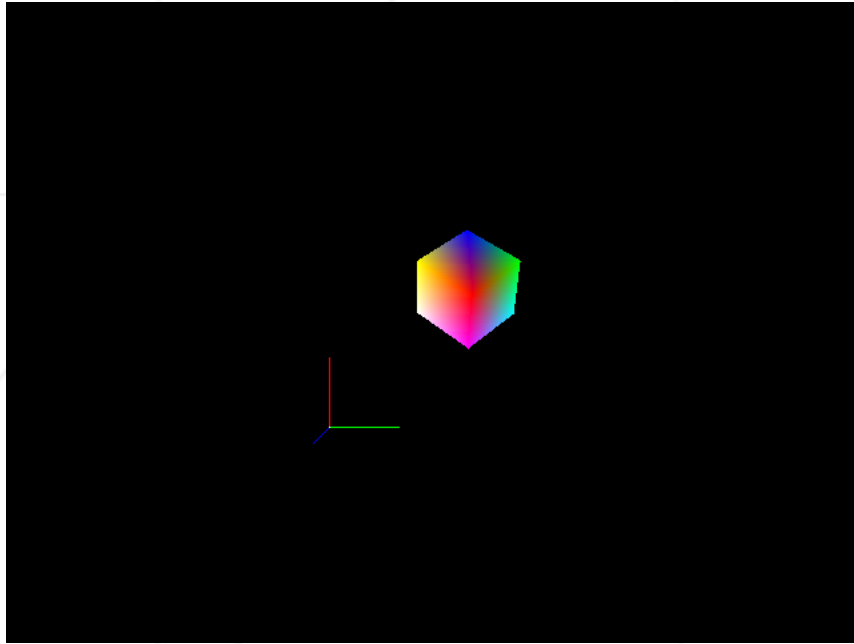
- Vertex :



- Edge :



- Raster :



Imaginea generata trebuie sa in format **png**. Pentru asta, intreaga biblioteca **GD** a **PHP** va sta la dispozitie.

Implementarea acestui exercitiu este foarte libera. Trebuie sa scrieti o clasa **Triangle** si o clasa **Render**. Puteti adauga clase noi si/sau modifica clasele precedente. Respectati urmatoarele instructiuni.

In privinta clasei **Triangle** :

- Constructorul clasei **Triangle** asteapta ca parametru un tablou. Cheile asteptate sunt urmatoarele :

'A' : Vertex-ul (nodul) primului punct al triunghiului, obligatoriu.

'B' : Vertex-ul (nodul) celui de-al doilea punct al triunghiului, obligatoriu.

'C' : Vertex-ul (nodul) celui de-al treilea punct al triunghiului, obligatoriu.
- Clasa **Triangle** trebuie sa contina un atribut **verbose static** de tip boolean ce permite controlul afisajului legat de folosirea clasei. Acest atribut este initializat la valoarea **False**.
- Daca si numai daca atributul static **verbose** este adevarat, constructorul si destructorul clasei vor produce un rezultat.
- Clasa trebuie sa aiba o metoda statica **doc** ce returneaza o documentatie scurta a clasei, sub forma unui sir de caractere. Continutul documentatiei trebuie citita din fisierul **Triangle.doc.txt** si este la dispozitia voastra , ca in exercitiul precedent.
- Nu sunt metode obligatorii pentru clasa **Triangle**. Cu toate astea, scrierea unora va poate fi utila. Cred de exemplu ca pentru a putea itera sau mapa nodurile triunghiului, itera muchiile lui, a putea sorta nodurile sau muchiile conform unor conditii etc.

Referitor la clasa **Render** :

- Constructorul clasei **Render** ia ca parametru un tablou. Cheile asteptate sunt urmatoarele :

'width' : Lungimea imaginii generate, obligatorie.

'height' : Inaltimea imaginii generate, obligatorie.

'filename' : Numele fisierului in care va fi salvata imaginea **png** creata, obligatoriu.
- Clasa **Render** trebuie sa contina trei constante ale claselor **VERTEX**, **EDGE** si **RASTERIZE** care vor servi la alegerea modului de randare.
- Clasa **Render** trebuie sa contina un atribut **verbose static** de tip boolean ce permite controlul afisarii legat de folosirea clasei. Acest atribut este initializat cu valoarea **False**.

- Daca si numai daca atributul static **verbose** este adevarat, atunci constructorul si destructorul clasei vor produce un rezultat.
- Clasa trebuie sa contina o metoda statica **doc** ce returneaza o documentatie scurta a clasei, sub forma unui sir de caractere. Continutul documentatiei trebuie citita din fisierul **Render.doc.txt** si este lasata la discretia voastra, ca la executiul precedent.
- Clasa **Render** trebuie sa aiba urmatoarele metode :

void renderVertex(Vertex \$screenVertex) : Afiseaza un vertex (nod) in coordonate "ecran" pe imaginea generata (un pixel).

void renderTriangle(Triangle \$triangle, \$mode) : Afiseaza un triunghi in coordonate "écran" pe imaginea generata conform modului dorit. Modul este una din cele trei constante de clasa.

void develop() : Salveaza imaginea **png** generata pe harddisk folosind numele de fisier furnizat in constructor.

Pentru a va simplifica munca, va sfatuiesc sa adaugati cateva functionalitati foarte simple anumitor clase din exercitiile precedente sau sa adaugati propriile voastre clase. Asta bineinteles ca nu este obligatorie. Vedeti mai jos cateva piste fara a fi exhaustive :

- O metoda **toPngColor** pentru clasa **Color** a carei pseudocod este urmatorul, cu ajutorulbibliotecii GD :


```
color = getColorAlreadyAllocatedInPNGImage( img, r, g, b )
IF color == -1
IF numberOfColorsInPNGImage( img ) >= 255
color = getPNGImageClosestColor( img, r, g, b )
ELSE
color = allocateNewColorInPNGImage( img, r, g, b )
RETURN color
```

- Suport al trunghiurilor in clasele **Matrix** si **Camera**.
- Adaugarea unei metode **bool isVisible(Triangle \$stri)** clasei **Camera** pentru a determina daca un triunghi este vizibil sau nu pentru a nu afisa partile din dosul obiectului. Puteti folosi z-buffer sau un BSP daca doriti, dar exista un algoritm mult mai scurt si mai simplu de utilizat pentru un simplu exercitiu de piscina. CAutati "backface culling".
- Adaugarea unei clase **Mesh** ce permite reprezentarea unui model 3D compus din triunghiuri ce permite manipularea usoara a nodurilor, a muchiilor si a trunghiurilor modelului. In mod evident, adaugand suportul acestei clase la clasele **Matrix** si **Camera** este extrem de practic!
- ...

Veti gasi un exemplu de utilizare a acestei clase in fisierul **main_05.php** din anexele subiectului ce a servit la generarea celor trei imagini precedente.

Chapitre XI

Exercitiu bonus 06 : Clasa Texture

	Exercice : 06
Clasa Texture	
Dossier de rendu : <i>ex06/</i>	
Fichiers à rendre : <code>Texture.class.php</code> , <code>Texture.doc.txt</code> , <code>*.class.php</code> , <code>*.doc.txt</code> , vos fichiers de textures	
Fonctions Autorisées : Tot ce a fost vazut de la inceputul piscinei, toata biblioteca standard PHP si biblioteca GD.	
Remarques : n/a	

Exercitiu pentru cei tari ce permite depasirea celor 20 de puncte in cadrul sustinerii. Adaugati suportul texturilor la rasterizatorul vostru. Puteti adauga si modifica tot ce doriti.

Respectati instructiunile urmatoare :

- Clasa **Texture** trebuie sa contina un atribut **verbose static** de tip boolean ce permite controlul afisarii legat de utilizarea clasei. Acest atribut este initializat la valoarea **False**.
- Daca si numai daca atributul static **verbose** este adevarat, constructorul si destructorul clasei produc un rezultat.
- Clasa trebuiesc aiba metoda statica **doc** ce returneaza o documentatie scurta a clasei sub forma unui sir de caractere. Continutul documentatiei trebuie citita din fisierul `Texture.doc.txt` si este lasat la dispozitia voastra, ca in exercitiile precedente.

Pentru a va ajuta in cautarile voastre, notiunile utile in acest caz sunt "coordonate **u** si **v** ale unui vertex" si coordonate baricentrice (barycentriques) ale unui triunghi ...



Dam o bere celui ce termina primul. - Ramona si Alex