Laboratori IDI: OpenGL, bloc 3

Professors d'IDI, 2012-13.Q2

22 d'abril de 2013

Igual que els blocs 1 i 2, aquesta part està pensada per a que la feu experimentant amb l'efecte de les diferents crides i paràmetres. Per a això us suggerim una sèrie d'experiments, però a més esperem de vosaltres que afegiu els propis, fins a convèncer-vos que enteneu per què passen les coses que passen en el vostre programa. En cas de dubtes, aprofiteu el professor de laboratori, que us donarà pistes de què mirar o on buscar. Podeu consultar internet per cercar informació, documentació,... Però no cerqueu codi i retalleu i enganxeu als vostres programes. Fent-lo no aprendreu gaire, i durant les proves de laboratori no hi ha accés a internet!

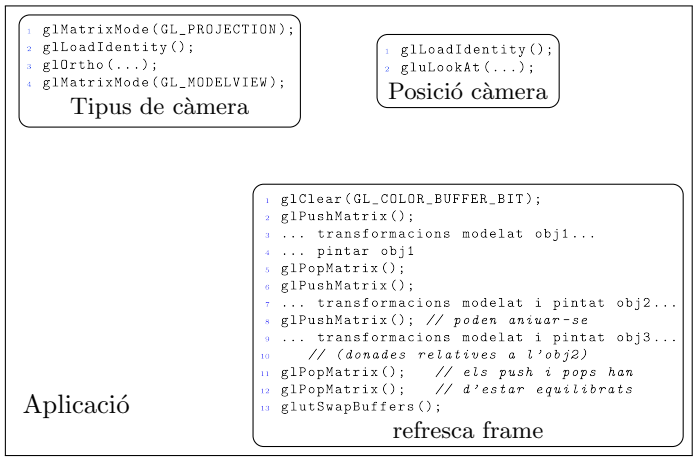
Hem intercalat el signe '►' per a senyalar punts específics en què es planteja un exercici o quelcom que necessita d'experimentació o proves per part vostra. No us estigueu de fer altres experiments, naturalment!

Com als blocs anteriors, cal que mireu de produir codi net, que més endavant pugueu aprofitar. Mireu de deixar una aplicació "neta" al final del bloc, en què es puguin activar les diferents funcionalitats (o almenys una versió de cadascuna) via tecles o botons del ratolí (amb modificadors). En la darrera secció d'aquest document, us indiquem les funcionalitats mínimes de l'aplicació que haureu de lliurar.

1 Introducció

L'objectiu d'aquest bloc és que entengueu les inicialitzacions de càmera en l'àmbit d'OpenGL i algunes tècniques bàsiques per a la seva manipulació. Us recomanem que repasseu els conceptes i problemes analitzats en classes de teoria.

Amb les modificacions que introduireu en aquesta pràctica, la vostra aplicació tindrà aquestes parts (el codi inclòs és a títol d'exemple; no el preneu literalment!):



On cada bloc de codi sols s'executarà quan sigui necessari: el de dalt a l'esquerra, quan canviem el tipus de càmera (per exemple si passem d'ortogonal a perspectiva, o si es canvien les mides de la finestra a pantalla); el de dalt a la dreta, quan es vol reposicionar la càmera; el de sota, cada cop que cal refrescar l'escena. Això és important per a que després pugueu afegir nous aspectes i millores al vostre codi. També és important que entengueu quan convé fer cada una d'aquestes operacions. La inicialització dels paràmetres de la càmera inicial les podeu calcular (i realitzar) en un bloc indepentent.

Podeu agafar com a aplicació de partida la dels planetes del bloc2, tanmateix us recomanem que pinteu una escena formada per un terra (quadrat d'aresta 10 centrat a l'origen de coordenades en el pla XZ) i, com a mínim, un objecte OBJ qualsevol ubicat a sobre de terra. La mida més gran de capsa contenidora de l'objecte OBJ ha de ser 4.

2 Càmeres axonomètriques

En el bloc 1 vàreu aprendre que en redimensionar la finestra calia adaptar el *viewport* per a evitar deformacions. Però tal com ho vàrem fer aleshores, amb les eines conceptuals de què disposàvem, es desaprofitava potencialment molta part de la finestra, o es perdia part de l'escena. A més, fins ara estàvem obligats a posar tota la geometria a dibuixar dins d'un volum determinat (l'anomenat volum de visió) fixat a $[-1, 1]^3$.

OpenGL ofereix mecanismes per a solucionar aquesta darrera restricció, i pal·liar el problema de l'aprofitament de la finestra. La crida `glOrtho` permet definir un volum de visió hexaèdric però entre altres límits. Ara bé, per a poder-ho fer (el definir un altre volum de visió) caldrà que primer coneguem més en detall els mecanismes de transformacions d'OpenGL. En realitat, OpenGL emmagatzema més d'una transformació. Fins ara, les transformacions que hem fet servir (que bellugaven i transformaven els models) modificaven la transformació que OpenGL guardava com a `ModelViewMatrix` (resultat de concatenar les transformacions de modelat i "view"). Per a modificar el volum de visualització, volem alterar la `ProjectionMatrix`. Per a fer-ho, abans de cridar a `glOrtho` caldrà indicar-ho:

```
1 glMatrixMode(GL_PROJECTION);
2 glLoadIdentity();
3 glOrtho(...);
4 glMatrixMode(GL_MODELVIEW);
```

Un cop modificada la matriu convenientment, la línia 4 d'aquest tros de codi ens retorna al mode en que estàvem. Això és convenient perquè és molt més freqüent que l'aplicació hagi de modificar la matriu de modelat o la de visualització que no pas la que defineix el volum de visió. D'altra banda és més fàcil desenvolupar un codi si adoptem la convenció que OpenGL està en un "`MatrixMode`" determinat i fix sempre que no es modifiqui localment.

►Agafant la vostra aplicació i l'escena que us hem recomanat o la dels planetes —per exemple— com a punt de partida, mireu d'incloure al *callback* de *reshape* crides a `glViewport` i a `glOrtho` de manera que s'aprofiti al màxim la finestra, no importa com es redimensioni, però sense tenir deformacions. No cal que la solució sigui un màxim —quant a l'aprofitament de l'espai— sinó un bon compromís entre aprofitar l'espai i calcular fàcilment els paràmetres de les crides. Aprofiteu la noció de *d'esfera contenidora* vista a classe, i assegureu que l'esfera es veu sencera al viewport. Si heu de fer servir al màxim la finestra en pantalla, òbviament ara el viewport que declareu a OpenGL haurà de cobrir-ne la totalitat, i per tant haureu d'evitar la deformació donant la mateixa *relació d'aspecte* al volum de visió —bé, a la cara anterior del volum de visió, és a dir al *window*—.

Recordeu que teniu la posició i orientació de la càmera per defecte d'OpenGL.

3 Posicionament de la càmera:`gluLookAt`

Fins a aquest moment, el nostre observador (o la nostra càmera) ha estat quiet a l'origen de coordenades. Sovint aquesta manera de pensar no és òptima, i en comptes d'això preferim poder col·locar-lo en qualsevol altra posició. Hi ha dues maneres principals de fer-ho: utilitzar `gluLookAt` o ubicar els objectes respecte la càmera de defecte.

La primera d'elles és amb la crida a `gluLookAt`. ►Busqueu la plana del manual corresponent per a veure els seus paràmetres, que designen la posició de l'observador (la càmera), un punt on l'observador centra la seva mirada (anomenat VRP, acrònim de l'anglès "view reference point"), i un vector que designa una direcció que l'observador veuria com vertical. Caldrà que afegiu l'*include* `GL/glu.h`, i que afegiu la llibreria corresponent a la instrucció de muntatge (-lGLU).

Cal que la posició de la càmera (crida a `gluLookAt`) es defineixi fora de la funció de refresc, i que a la funció de refresc **ja no es cridi a** `glLoadIdentity`. En canvi, per a evitar l'acumulació de transformacions, parentitzeu les transformacions geomètriques que us calguin entre `glPushMatrix` i `glPopMatrix`. En altres paraules, cal que no es cridi a gluLookAt cada vegada que es vol repintar, sinó sols quan hom mogui la càmera. Això pot semblar una complicació però permetrà després un millor control de l'aplicació, a més de ser més eficient car no cal definir repetidament unes variables tot donant-les-hi sempre el mateix valor...)

Observeu que ara tenim desacoblades dues transformacions: unes defineixen la posició de la càmera (la crida a `gluLookAt`), i altres modifiquen els models (les crides que tingueu a `glScale, glRotate`, etc). En realitat totes estan composant-se en una única transformació que s'emmagatzema en la matriu de ModelView. Cal doncs que l'assignació de la posició de la càmera es defineixi primer, i les altres transformacions s'afegeixin després. Per aquest motiu fem el `LoadIdentity` just abans de la crida a `gluLookAt`:

```
1 glLoadIdentity();
2 gluLookAt(...);
```

►Afegiu la inicialització d'una càmera que al posar en funcionament l'aplicació mostri una imatge on la càmera mira cap al centre de la vostra escena i permeti veure-la en planta. Després proveu que la mostri des d'una posició arbitrària. Proveu variar el vector "up" per a entendre el seu funcionament.

4 Càmeres perspectives

Per tal d'augmentar el realisme, afegint l'empetitiment perspectiu (és a dir el fet que les coses llunyanes es veuen més petites que les properes), podem fer servir una càmera perspectiva, en comptes d'una càmera axonomètrica (també anomenada ortogonal).

Per a programar la càmera perspectiva caldrà substituir la crida a `glOrtho` per una crida a `gluPerspective(fovy, aspect, near, far)` (per tant amb les mateixes condicions de fer-se en "`MatrixMode`" `GL_PROJECTION`, com al tros de codi de la secció anterior).

Aquest model de càmera s'assembla més al d'una càmera real, i per tant les escenes es veuran més fidelment, especialment en quan a la seva profunditat. Tanmateix, heu de tenir algunes consideracions en compte:

- el paràmetre `fovy`, que mesura l'angle entre el pla de retall superior i el pla de retall inferior, s'ha de donar en graus sexagesimals. Tingueu-ho en compte si feu servir funcions trigonomètriques en el seu càlcul, car aquestes operen en radians.
- `aspect` és la relació d'aspecte del *window*. Es defineix com l'amplada dividida per l'alçada. Però alerta que quan feu la divisió es faci en coma flotant, no entre enters!
- els paràmetres `near` i `far` han de ser estrictament més grans que zero. Representen distàncies a la càmera. Tot el que sigui a menys de `near` o a més de `far` en la direcció de les z es retallarà, i no apareixerà en pantalla.
- L'anterior implica que ara, l'observador no pot trobar-se dins del volum de visió. Per tant, caldrà posicionar la càmera de manera adient amb `gluLookAt`. Quan posicioneu la càmera amb angles d'Euler—secció següent—, caldrà recordar que en la càmera per defecte l'observador segueix essent a l'origen, per tant caldrà moure la geometria que voleu veure a una distància superior a `near` per tal que el retallat no l'elimini.
- cal que aprofiteu l'espai el millor possible (és a dir que cal fer que l'escena es vegi tant gran com es pugui, sense retallar-ne cap part; tanmateix, si aquest òptim és costós de calcular exactament, és més raonable adoptar una bona aproximació que sigui fàcil de calcular. Però no és acceptable una que es basi en "factors de seguretat" i pugui deixar l'escena empetitida al mig del viewport...).
- Com en els altres casos, cal fer un tractament diferenciat quan `aspect` $>= 1$ i quan `aspect` < 1 per a evitar deformacions.

►Inicialitzeu una càmera perspectiva que permeti veure tota l'escena, aprofitant l'espai del viewport (que sempre serà tota la finestra gràfica) i assegurant que no hi ha deformacions en fer un `resize`.

►Programeu una tecla (per exemple la `'p'`) que passi a càmera perspectiva en la vostra aplicació, i una altra (per exemple la `'x'`) que torni a la càmera axonomètrica de l'apartat anterior. Quan tingueu programada la càmera perspectiva, mireu d'apreciar les diferències.

5 Posicionament de la càmera amb transformacions geomètriques (angles d'Euler)

En comptes de fer servir `gluLookAt`, podem posicionar la càmera fent servir transfomacions geomètriques, ja que el que importa **és la posició relativa de la càmera i l'escena**, o sigui que les transfomacions geomètriques es poden interpretar com actuant sobre els objectes de l'escena **o actuant (de la manera oposada) sobre la càmera**.

►Programeu que es pugui commutar entre una i altra definició de càmera mitjançant dues tecles (per exemple `'a'` per `gluLookAt` i `'e'` per a angles d'Euler).

►Substituïu la crida a `gluLookAt` per un seguit de transformacions geomètriques. Per a fer-ho, s'acostuma a traslladar el VRP a l'origen, de forma que les rotacions que es facin a continuació girin al seu entorn. Aleshores s'apliquen tres rotacions al voltant dels eixos y, x, i z (en aquest ordre), que corresponen a la direcció (del pla x-z) des de la que es mira, l'elevació (o la inclinació de la càmera "cap avall"), i la rotació de la càmera al voltant del seu propi eix. Finalment, cal allunyar el VRP a la distància que li correspon de la càmera:

```
1 glTranslated(0., 0., -dist);
2 glRotated(-anglez, 0., 0., 1.);
3 glRotated(anglex, 1., 0., 0.);
4 glRotated(-angley, 0., 1., 0.);
5 glTranslated(-VRP.x, -VRP.y, -VRP.z);
```

►Experimenteu amb diferents angles i distàncies, i canviant el VRP.

►Afegiu el codi necessari per a que es puguin canviar els angles d'Euler interactivament arrossegant el ratolí, associant el moviment horitzontal al gir al voltant de l'eix y (canviat de signe), i el moviment vertical al gir al voltant de l'eix x

►Mireu d'emular l'acció de `gluLookAt` completament amb aquestes crides al seu lloc. Hauríeu d'aconseguir que prement-les alternativament no es veiés cap canvi a la pantalla. Penseu primer quin ha de ser `dist`. Després com calcular l'`angley` a partir de les coordenades de l'observador i del VRP. Finalment, com calcular l'`anglex` a partir de l'observador i del VRP. L'`anglez` depèn del vector "up"; si poseu `vup` $= (0, 1, 0)$, no us hauria de caldre aquest gir. ►Però poseu-lo amb diferents valors (o fent que es pugui variar amb el ratolí o amb una tecla, per exemple) només per a veure quin és el seu efecte en la imatge final.

►Actualitzeu el help de la vostra aplicació per a que mostri totes les tecles disponibles.

6 Zoom, Pan

A banda de les rotacions, les aplicacions sovint necessiten altres modificacions de les vistes. En aquest apartat aprendrem a implementar dues, el zoom i el pan, associades (en sentit laxe) a translacions.

El zoom —en càmeres perspectives— correspon a apropar-se a l'escena. Pot implementar-se de dues maneres diferents: com un canvi efectiu de la posició de la càmera, fent-la avançar (o retrocedir) al llarg de l'eix de visió, o bé com un "canvi en l'òptica de la càmera", que correspon més exactament amb l'operació de fer zoom amb una càmera real.

La primera opció (OPTATIVA), acostar o allunyar la càmera, és fàcil d'implementar en totes les formes de posicionar la càmera que hem vist. Si fem servir `gluLookAt`, mourem l'observador al llarg de l'eix OBS–VRP. ►Proveu de fer-ho, controlant-lo interactivament amb el ratolí (podeu fer servir modificadors, de manera que per exemple majúscules+arrossegar el ratolí correspon a fer zoom, o fer servir tecles per a modificar l'estat de la interfície).

►Si en canvi la càmera està posicionada amb angles d'Euler, com ho faríeu per a aconseguir el mateix?

►Finalment, proveu d'implementar el zoom canviant l'óptica de la càmera, és a dir modificant el paràmetre `fovy` de la crida a `gluPerspective`, o els paràmetres de la `glOrtho` si esteu fent servir una càmera axonomètrica. Aquest, per cert, és l'únic mecanisme de zoom en cas de càmeres axonomètriques (►Veieu per què?). ►Per les càmeres perspectives, veieu alguna diferència entre el zoom obtingut movent la càmera i l'obtingut modificant `fovy`? Sabeu explicar el perquè?

Aquesta part és OPTATIVA.

El "pan" correspon en canvi a una translació de la càmera en una direcció **perpendicular** a l'eix de visió. ►Per càmeres perspectives, això es pot implementar movent efectivament la càmera en aquella direcció (►què passa si feu el mateix amb una càmera axonomètrica? I com podeu obtenir un efecte de pan en aquest cas?).

Per a obtenir les direccions en què us hauríeu de moure en la vostra escena per a aconseguir un desplaçament horitzontal o vertical de la càmera (un desplaçament horitzontal o vertical de la vista), podeu recórrer altre cop a la funció `glGetDouble(GL_MODELVIEW_MATRIX,...)`, i fer servir per a aquestes direccions les primeres dues files (ignorant la quarta component); alerta però, perquè OpenGL guarda les matrius per columnes i no per files.

7 Navegació

Els mecanismes per a anar modificant la càmera interactivament sovint es denominen genèricament mecanismes de navegació per l'escena. Però n'hi ha de diferents, i poden ser més o menys apropiats segons la situació i el tipus d'aplicació, així com també per preferències personals dels usuaris. A continuació veurem alguns d'aquests mecanismes de navegació i la seva implementació. **En la discussió següent suposem sempre una càmera perspectiva.**

7.1 Inspect

El mode "inspecció" més simple correspon (possiblement amb petites variants) al que heu estat implementant en els apartats anteriors modificant els angles d'Euler. L'escena es veu com un objecte d'interès que volem inspeccionar, mirant-lo des de direccions diferents, i acostant-nos als detalls rellevants (és com si el sostinguéssim a la mà i el moguéssim per a estudiar-lo). Aquest mode, per tant, ja l'hem implementat. Tanmateix, el girar l'escena a inspeccionar amb angles d'Euler no dóna girs intuïtius (en alguns casos). Vegem ara com millorar-ho.

7.1.1 Modificació interactiva en coordenades d'ull

En el Bloc 2 ja vàrem aprendre que, per a inspeccionar un model, resultava més intuïtiu concatenar les transformacions per l'esquerra en comptes de per la dreta, de manera que sempre girés, si canviàvem interactivament la seva orientació, de la manera esperada.

►També ara podeu integrar aquesta millora a un posicionament de la càmera efectuat per qualsevol dels mecanismes estudiats (glLookAt() o angles d'euler). Per això, posicioneu d'antuvi la càmera amb el mètode de la vostra preferència, però a partir d'allí, modifiqueu la seva posició concatenant transformacions per l'esquerra (via `glGetDoublev...`). El mètode de posicionament "absolut" tant sols s'ha de tornar a cridar si l'usuari vol fer un *reset* de la càmera (cosa per la qual és bo proporcionar-li algun mecanisme. ►Afegiu alguna tecla que en ésser premuda retorni la càmera a una posició coneguda). Aquests canvis han de succeir en el bloc de codi de posicionament de càmera, no en el moment de refrescar un frame. D'aquesta manera preserveu el desacoblament de la gestió de la càmera i del model. A més, donat que el mètode de composar els girs per l'esquerra és incremental, es facilita tot plegat, ja que tant sols s'executa aquest codi quan es vol modificar la posició de la càmera.

7.2 Fly-through

Observació inicial: aquest mètode és OPTATIU.

En aquest cas, en comptes de sostenir a la mà el model, hem d'imaginar que estem a un avió que el sobrevola. El programa actualitza constantment la posició de l'avió (fent-lo avançar, tal com fèieu a un dels modes de zoom), i l'usuari controla a través del teclat i del ratolí la velocitat, així com els girs. Per donar realisme, els girs a dreta i esquerra s'haurien d'acompanyar d'un gir al voltant de l'eix de visió (els avions no giren perfectament horitzontals. Aquest gir al llarg de l'eix de visió s'anomena "roll"). També pot controlar l'alçada del vol, canviant la inclinació de l'avió (donada pel gir en x en angles d'Euler, anomenat en aquest context "pitch").

►Mireu d'implementar una navegació amb aquest paradigma per a la vostra aplicació. Aprofitareu el *callback* de `glutIdleFunc` per a cada *frame* actualitzar la posició de la càmera, en comptes (o a més) de modificar l'escena. Fixeu-vos que el que veiem, és el que veuria una càmera fixada al nas del nostre avió imaginari, no el pilot. El pilot, en realitat, té més graus de llibertat, perquè pot triar no mirar recte cap endavant. ►Com implementaríeu un model més detallat que a més de moure's al llarg de la trajectòria de l'avió com abans, permeti al pilot "moure el cap"?

7.3 Walk

►En aquest cas, el model és una persona caminant, o recorrent l'escena en un vehicle. Com en el cas del fly-through de l'apartat anterior, l'aplicació actualitza constantment la posició de l'observador, i l'usuari pot modificar la velocitat i la direcció. En aquest cas, però, no s'inclina la càmera quan es gira, ni es pot canviar l'alçada (no hi ha "pitch"; en aplicacions avançades hi haurà una noció de terra, i si aquest és inclinat, la càmera pujarà o baixarà en recórrer-lo), ni.

►Podeu aconseguir una versió més avançada (OPTATIVA) si desacobleu la direcció de visió de la d'avançament (és a dir li deixem caminar mirant lleugerament a un costat, per exemple).

8 Aplicació a lliurar

Cal que al començament de la primera sessió de laboratori del bloc 4, lliureu una aplicació amb les següents funcionalitats:

- A l'iniciar l'aplicació mostri una escena des d'una posició arbitrària (sense retallar), amb una càmera perspectiva i aprofitant la grandària del viewport (definit com tota la finestra gràfica).
- Permeti communtar entre les càmeres perspectiva i axonomètrica. En cap cas ha de haber-hi deformació si es fa un "resize".
- Permeti fer zoom modificant l'òptica de les càmeres.
- Permeti definir la càmera inicial amb glLookAt() o angles d'Euler.
- Permeti inspeccionar l'escena modificant els angles d'Euler i en coordenades d'ull
- Permeti inspeccionar l'escena amb "walk".
- Permeti fer un "reset" per tornar a la visualització de l'escena des de la càmera inicial.
- Hi hagi un help que mostri com activar les diferents funcionalitats.