Laboratori IDI: OpenGL, bloc 4

Professors d'IDI, 2013-14.Q2

31 de març de 2014

Igual que els blocs 1, 2, i 3 aquesta pràctica està pensada per a què la feu pausadament, experimentant amb l'efecte de les diferents crides i paràmetres. Disposareu de **dues sessions** de laboratori per a completar-la. Orientativament, en la primera sessió heu de completar, com a mínim, la Secció 2. Haureu de lliurar aquest bloc, a través del racó, com a molt tard, el dilluns 21 d'Abril a les 23:59.

També com en laboratoris anteriors, hem intercalat el signe '►' per indicar punts específics en què es planteja un exercici fonamental o quelcom que requerireu per a l'aplicació final. El signe '►' indica altres punts 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!

Insistim també que 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). Aquesta aplicació servirà de base per a l'examen de laboratori.

1 Introducció

En aquest bloc anem a intentar veure les escenes amb cert realisme. Això comporta pintar l'escena amb parts amagades, amb il·luminació (focus de llums) i considerant els materials dels objectes.

Feu servir l'escena de l'aplicació del bloc 3, amb totes les funcionalitats que tingueu implementades. Per entendre els conceptes d'aquest bloc, com a mínim cal poder carregar l'escena, i definir una càmera arbitraria que veu tota l'escena i que es pot moure interactivament.

Incorporeu, si no ho heu fet encara, les instruccions del bloc 3 per a activar l'elimació de parts amagades i per pintar les cares amb omplert de polígons.

2 Materials

Per a poder dibuixar objectes amb més realisme que fins ara, cal tenir en consideració les propietats de què estan fets dels materials. OpenGL pot dibuixar polígons il·luminats amb els models empírics d'il·luminació que heu estudiat a classe, però per a que ho faci haurem d'afegir un conjunt d'instruccions per a la descripció dels materials, indicar la normal de les cares, encendre llums, posicionar llums,... No podreu valorar l'efecte final fins que tingueu correctament incorporades, al menys, les instruccions que us indiquem en aquesta secció.

Tingueu present que quan dibuixem, la il·luminació d'OpenGL pot estar en un de dos estats: activada o desactivada. Aquests estats es canvien mitjançant les funcions `glEnable(GL_LIGHTING)` i `glDisable(GL_LIGHTING))`.[1]

►Activeu la il·luminació, per exemple, a la inicialització de l'estat d'OpenGL. Implementeu una tecla 'i' que permeti desactivar/activar la il·luminació.

►A més caldrà que encengueu alguna llum. Afegiu la crida `glEnable(GL_LIGHT0)` a la inicialització. Tindreu la llum de defecte encesa, **mireu en el manual** les seves caraterístiques. No cal que feu el corresponent `glDisable()`, llevat que vulgueu apagar aquest llum.

Quan OpenGL té activat el mode `GL_LIGHTING`, fa servir els models empírics d'il·luminació per a calcular els colors dels vèrtexs; en canvi, si està desactivat, fa servir el darrer color que haguem definit amb `glColor*()`. Si activeu la il·luminació, per tant, les crides a `glColor*()` deixaran de tenir efecte (fins que la torneu a desactivar). **En altres paraules**, fins que no completeu l'especificació dels materials no veureu correctament els colors dels objectes. ►Ho podeu provar ....

[1] `glEnable` i `glDisable()` admeten una multitud d'altres valors del paràmetre, que activen i desactiven diferents opcions i processos que pot realitzar el servidor d'OpenGL. Consulteu la pàgina del manual d'aquestes funcions per a saber més, però tingueu present que moltes d'aquelles opcions no us seran significatives en aquest punt.

Per a fer servir la il·luminació, i aprofitar els materials descrits als arxius `OBJ` que carreguem, haureu d'afegir a la vostra aplicació les crides convenients dins del bucle de dibuix/pintat. La manera en què això funciona és com sol ser en OpenGL: disposem d'unes funcions (`glMaterial*()`) que modifiquen l'estat d'OpenGL. Quan enviem pel *pipeline* un vèrtex, aquest s'il·luminarà d'acord amb els materials que hi hagi en aquell moment al context gràfic. Per tant, abans d'enviar cada vèrtex, hem d'assegurar-nos que el material actiu és el correcte. Per a assignar un nou material actiu, cal fer un cert nombre de crides a `glMaterial*()`, una per cada propietat que volem modificar. Els paràmetres d'aquesta crida són el tipus de cares que afecten (igual que en el cas de `glPolygonMode()`, poden ser `GL_FRONT`, `GL_BACK` o `GL_FRONT_AND_BACK`), la propietat del material que volem assignar (que pot ser `GL_AMBIENT`, `GL_DIFFUSE`, `GL_SPECULAR` o `GL_SHININESS`, entre d'altres), i els nous valors. En el cas de les primeres tres propietats, que corresponen al color ambient, difús i especular, el valor serà un vector de quatre `float`s (i fareu servir `glMaterialfv()`). Per a assignar el quart paràmetre, `GL_SHININESS`, que dóna l'exponent de Phong a fer servir en el càlcul de la il·luminació, podeu fer servir enters o `float`s; el valor, però, ha d'estar entre 0 i 128.

Recordeu que tal com s'emmagatzemen els models a la classe `Model` que us proporcionem, els **materials estan associats a les cares** del model. ▶Repasseu la classe `Model` i penseu com accedir a la informació dels materials.

▶Inclou en la teva aplicació les crides a `glMaterial*()` per a assignar a les cares el material especificat en el model. Abans d'enviar pel pipeline els vèrtexs d'una cara, haureu de fer les quatre crides a `glMaterial*()` per a assignar el material corresponent. Com que el que emmagatzemem per a cada cara és, en realitat, un enter que indica la posició al vector materials de les dades –del seu material–, podeu fer una petita optimització incloent aquestes crides sols quan el material d'una cara és diferent del de l'anterior, amb el simple mecanisme de guardar un enter que indica l'índex del material anterior, i comparar-lo amb l'actual.

Altres objectes de la vostra escena (parets, terra, ninots de neu) no són instàncies de cap model `obj` i no tenen materials pre-assignats. Tanmateix cal definir un material per les seves cares si les volem veure correctament.

▶Inclou les següents especificacions de material. El terra haurà de ser d'un material brillant blau (ha de tenir reflexió especular). Les parets d'un material verd mate i els ninots de neu han de ser també brillants (cap i cos de color blanc i nas taronja). Com haureu intuït, en el cas dels objectes `glut`,com `glutSolidSphere()`, no es possible especificar un material per a cada cara, sinó que cal fer-ho a nivell d'objecte.

Com que els models empírics d'il·luminació fan servir la normal a la superfície en què incideix la llum, també l'haurem de proporcionar a OpenGL. Per tant, haureu d'afegir, abans de cada crida a `glVertex*()`, una crida a `glNormal*()` per a emmagatzemar la normal actual al context gràfic. El model que heu llegit de disc **tindrà sempre la normal per cara definida** (i compartides pels tres vèrtexs de la cara, pel que no cal tornar a enviar la normal per cada vèrtex); però només tindrà definides normals per vèrtex si el model `obj` les contenia. Amb normal per vèrtex, el model tindrà una aparença més suau. ▶Per què? ▶Analitzeu com accedir a la normal per cara i per vèrtex del model.

Observacions: El `patricio.obj`, utilitzat en l'aplicació del bloc 3, conté, a més de normal per cara, normal per vèrtex (també les contenen els models `f-16.obj`, `cow.obj` i `porsche.obj`). Les funcions de `glut` de pintat, declaren normal per vèrtex en `glutSolidSphere()`i per cara en `gluSolidCube()`.Si heu creat altre polígons (per exemple, per crear el terra), recordeu que també els hi heu d'assignar normals –per cara– abans de pintar.

▶Incorporeu les crides requerides a `glNormal*()`. Prement la tecla 'n' s'ha de poder communtar entre normal per cara i normal per vèrtex. Observeu les diferències resultants.

OpenGL parteix de la hipòtesi de què les normals estan normalitzades quan s'efectúa el càlcul de la il·luminació en cada vèrtex. Tanmateix, encara que estiguin normalitzades en el model, no podem garantir que ho estiguin quan OpenGL calcula la il·luminació en cada vèrtex (recordeu que es calcula en coordenades d'observador, després de multiplicar la normal per la matriu activa de `GL_MODELVIEW`). ▶Per què? Per aquest motiu, us recomanem que ▶poseu en la inicialització del context d'OpenGL `glEnable(GL_NORMALIZE)` que indica a OpenGL que abans d'utilitzar un vector el normalitzi (només cal fer la crida un cop en tot el programa!).

Per a què tot el vostre esforç doni fruits, però, recordeu que caldrà que estigui activat el *flag* `GL_LIGHTING` (cosa que podeu fer en la inicialització, fins que implementeu alguna tecla per a canviar d'un mode a l'altre).

Després d'aquesta feinada, hauríeu de veure els models que carregueu amb els colors que els seus dissenyadors hagin assignat (si ho han fet).

▶Proveu, si voleu, altres materials pels objectes `glut`. Proveu passar al mode de primera persona.

▶**Opcional però interessant**. El context de defecte d'OpenGL pinta les cares fent *smooth shading*, proveu que pinti amb *flat shading* i observeu les diferències. Cal utilitzar la instrucció `glShadeModel()` amb `GL_FLAT` (o `GL_SMOOTH` per tornar al mode *smooth shading*). Proveu l'efecte tenint normal per cara o normal per vèrtex.

3 Llums

Per a què tot l'anterior funcioni, cal que hi hagi llums a l'escena. Si us ha funcionat sense configurar-les és perquè al fer `glEnable(GL_LIGHT0)` heu activat la llum (`GL_LIGHT0`) que, per defecte, OpenGL col·loca a la càmera i inicialitza amb uns valors per defecte. Però podem modificar aquests valors, posicionar-la en altres llocs, o fins i tot encendre i apagar independentment fins a 8 llums [2], anomenades `GL_LIGHT0` fins a `GL_LIGHT7`. El mecanisme és semblant al dels materials, però aquest cop fem servir `glLight*()` (que també existeix en les variants `f`, `i`, `fv` i `fi`). Aquestes crides tenen tres paràmetres, que designen la llum a modificar, el paràmetre específic que volem assignar, i el nou valor. Els paràmetres específics poden ser, entre d'altres, `GL_AMBIENT`, `GL_DIFFUSE` i `GL_SPECULAR` (que com en els materials designen els corresponents colors, però aquest cop de la llum), o `GL_POSITION` (per a donar una nova posició de la llum). Hi ha d'altres paràmetres per a definir atenuacions i *spotlights* que no veurem en aquest laboratori (però podeu explorar si voleu).

Aquestes crides funcionen, pel que fa als colors de la llum, de manera semblant als materials. **Noteu** que si la llum no ha de canviar de color o intensitat, podeu assignar-los durant la inicialització i oblidar-los (recordeu fer servir vectors de quatre *floats*, i la variant `glLightfv()`).

Observació: Vigileu de no posar intensitats molt altes de colors de llum, per què podeu saturar els colors percebuts de les cares. En particular, aconsellem NO posar color `GL_AMBIENT` i posar els mateixos colors per `GL_DIFFUSE` i `GL_SPECULAR`

La inicialització de la posició és una mica diferent. Quan executeu la crida `glLightfv(GL_LIGHTx, GL_POSITION, pos)`, **cal** també que `pos` contingui una posició en coordenades homogènies (amb quatre components). Si la quarta és 1, les tres primeres components s'interpretaran com coordenades de la posició de la llum. Si és zero, s'interpretaran com components d'un vector que indica la direcció des de la qual prové la llum (per a modelar llums molt distants, com la del sol). Altra diferència important és que **en el moment de fer la crida** OpenGL multiplicarà aquesta posició per la matriu de `ModelView` actual (la que estigui activa en aquell moment) per a convertir les coordenades a coordenades d'ull/observador. **Si volem posicionar una llum en coordenades de món** (per exemple, un fanal en una escena), haurem de repetir aquesta crida cada vegada que la `ModelView` canviï. **Si en canvi volem que una llum tingui una posició donada en coordenades d'ull/observador** (per exemple per a simular llums solidàries amb la càmera o a una ubicació fixa respecte d'ella) haurem d'assegurar-nos que la matriu de `ModelView` conté la identitat en el moment de cridar a `glLight*()` (per exemple fent un `glPushMatrix()` i un `glLoadIdentity()` just abans de cridar `glLight*`, i un `glPopMatrix()` després per a restablir la transformació que hi havia).

►La llum 0, definida per defecte per OpenGL, és una llum de càmera, per què?. Tanmateix res ens priva de re-definir-la com dessitgem.

►Desactiveu, de moment, la llum `GL_LIGHT0` i inicialitzeu una nova llum `GL_LIGHT1` com llum de càmera. Programeu la tecla numèrica `'1'` per a què permeti activar-la i desactivar-la. Per entendre-ho tot, és convenient que inicialitzeu els seus colors amb valors coneguts per vosaltres. Proveu diferents valors. Moveu la càmera. proveu diferents posicions de la llum en relació a la càmera. Aquest llum, en acabar ha de ser blanc.

►Col·loqueu ara la llum `GL_LIGHT0` a una alçada fixa i petita per sobre **d'una de les cantonades** del terra de la vostra aplicació (en coordenades de món, pel que s'ha explicat més amunt, haureu de repetir la crida a `glLightfv(..., GL_POSITION,...)` cada vegada que es modifiqui la matriu de ModelView). La llum hauria de ser groguenca. Analitzeu el que veieu; sabeu explicar-ho?

►La tecla numèrica `'0'` us hauria de permetre encendre i apagar la llum 0.

►Programeu la tecla `'m'` per a què us permeti modificar la posició de la llum anterior, per a què estigui -successivament- sobre el següent vèrtex (en el sentit cíclic dels vèrtexs del terra) a la mateixa alçada; a més, abans de tornar al vèrtex inicial, és situarà -a la mateixa alçada- sobre el patricio que no es mou. El que veieu és consistent amb la vostra explicació? ►Se us acut com millorar la qualitat d'aquestes imatges, quant a la il·luminació?. ►Proveu a pintar i no pintar les parets, enteneu la il·luminació resultant?.

►Programeu una nova llum blanca `GL_LIGHT2` que es mou solidàriament amb el patricio i que es pot activar/desactivar amb la tecla `'2'`. Aquesta llum estarà ubicada en la mateixa posició respecte el patricio en què ubiqueu l'observador en mode primera persona.

►**Opcional però interessant**.

- Proveu activar i desactivar el *Back-Face Culling* i veure el seu efecte. Cal utilitzar `glEnable(GL_CULL_FACE)`. Mireu el manual. Observeu si l'aplicació és més eficient. Observeu també l'escena des de sota del terra. Enteneu el que passa?

[2] El standard d'OpenGL indica que tota implementació d'OpenGL ha de suportar almenys vuit llums, però com cada llum addicional representa un esforç computacional no menystenible, tots els fabricants opten per suportar exactament vuit; així compleixen el standard sense degradar el rendiment.

- Mireu en el manual el funcionament de `glLightModel()`. Proveu modificar la llum ambient i que es pugui encendre i apagar.

4 Lliurament

Cal que, com a molt tard, el dia abans de l'examen de laboratori lliureu l'aplicació resultant del bloc 3 ampliada amb les següents funcionalitats:

- En l'engegar l'aplicació mostri l'escena amb els materials que els correspond segons s'ha indicat en la Secció 2 i il·luminats per la llum –blanca– de càmera (`GL_LIGHT1`).
- Prement la tecla `'n'` s'ha de poder communar entre normal per cara i normal per vèrtex.
- Prement la tecla `'1'` es pugui activar/desactivar el llum de càmera.
- Prement la tecla `'0'` es pugui encendre i apagar un llum d'escena –groc– (`GL_LIGHT0`) -ubicat sobre una cantonada del terra.
- Prement la tecla `'m'` la posició del focus d'escena passi a estar sobre el vèrtex següent del terra –seqüencialment– com s'indica en la Secció 3.
- Prement la tecla `'2'` es pugui activar/desactivar una llum blanca que es mou solidàriament amb el patricio.
- La tecla `'i'` ha de permetre activar/desactivar la il·luminació.
- Modifiqueu el *reset* per a què, a més a més de modificar la càmera, torni a les condicions inicials d'il·luminació.
- Amplieu el vostre *help* amb informació de les noves funcionalitats.