Laboratori VIG. Pràctica 2
Primavera 2009-2010

Professors de VIG

13 d'abril de 2010

L'objectiu d'aquesta pràctica és ampliar l'aplicació del lliurament anterior. Les noves funcionalitats que haureu d'implementar inclouen la incorporació d'un entorn d'il·luminació i l'animació del vehicle al llarg del circuit. L'aplicació resultant constitueix la tercera pràctica de laboratori que serà objecte d'avaluació.

1 Introducció

Per al desenvolupament d'aquesta pràctica disposareu de tres sessions de laboratori. L'aplicació resultant ens l'haureu de lliurar seguint les mateixes instruccions que per a la pràctica anterior. La seva avaluació es farà en base a l'assoliment de les funcionalitats requerides (veure l'apartat 2) i a la usabilitat de la interfície de l'aplicació.

La data límit de lliurament és el **dijous 6 de maig a les 16:00**. A més a més, els professors us poden demanar una demostració de l'aplicació en hores de classe de laboratori.

A efectes de facilitar-vos la seva implementació us proporcionem codi preprogramat. Una part d'aquest codi afecta a la implementació global de la pràctica i comporta substituir/modificar codi que us vàrem subministrar en la Pràctica 1. Us recomanem que feu aquests canvis abans de començar aquesta pràctica. Concretament, al directori `/assig/vig/sessions/S2.2/` trobareu:

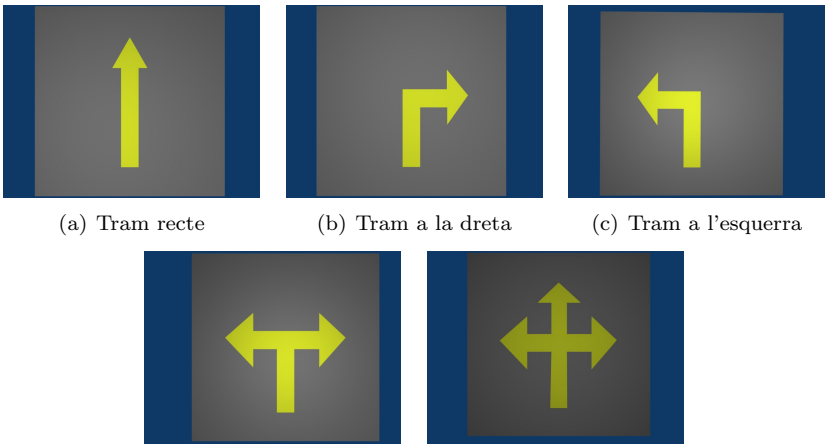
- Un fitxer `CalculaNormals.codi` amb la implementació del mètode `CalculaNormals()` que heu d'afegir a la classe `Object` (en el `object.cpp`, i, òbviament, la seva capçalera en el `object.h`).
- Un fitxer `ComputeNormal.codi` amb la implementació del mètode `computeNormal()` que calcula la normal d'una cara, i que cal afegir a la classe `Face`.
- Un directori `uiLlums_qt4` que conté la definició de la classe `llum` que necessitareu per a gestionar les llums, una interfície que podeu fer servir per a modificar-les, i un petit exemple d'ús per a facilitar-vos la seva incorporació al vostre codi.

2 Funcionalitats de l'aplicació

A continuació us indiquem les funcionalitats requerides en aquesta pràctica. A l'apartat 4 teniu el guió detallat de cadascuna de les tres sessions de laboratori.

L'aplicació incorporarà aquestes noves funcionalitats:

- Moviment del vehicle. El vehicle s'haurà de moure per damunt dels trams de carretera i seguint la direcció que aquests trams marquen, és a dir en un tram recte la direcció del vehicle es manté i aquest es mou travessant el tram de carretera, mentre que en un tram de carretera que marca canvi de direcció, el vehicle haurà de canviar de forma suau la direcció de moviment per a poder continuar pel següent tram de carretera en la direcció indicada. Hi haurà trams de carretera on s'ha de mantenir la direcció, trams que obliguen a un determinat gir (dreta o esquerra) i trams que donen opció a escollir (si girar o no, dreta o esquerra, etc.). Els trams possibles són els mostrats a la figura 1.



(a) Tram recte (b) Tram a la dreta (c) Tram a l'esquerra

(d) Tram amb doble sortida (e) Tram amb triple sortida

Figura 1: Imatges dels possibles trams

En tots els casos, els trams de carretera estàn situats sobre el pla XZ (pla Y=0) i els possibles moviments del vehicle són únicament X+, X-, Z+ i Z- (és a dir en les direccions paral·leles als eixos X o Z). En el cas en què es pot escollir la direcció per al següent tram (en els trams que donen opció) cal escollir la direcció de sortida de forma aleatòria i equiprobable. Quan hagi de girar a un cert tram, i mentre recorre aquest tram, el vehicle descriurà un quart de circumferència, i romandrà en tot moment tangent a la seva trajectòria.

Quan el vehicle es troba en un tram de carretera que desemboca en un altre tram que va en una direcció diferent, també haurà d'efectuar un gir per a seguir la direcció del nou tram, com si aquest hagués estat un tram de gir cap a aquella direcció (veure figura 2).

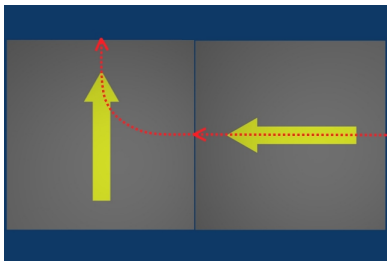


Figura 2: En aquest cas el vehicle també ha de fer el gir per a sortir en la direcció correcta.

- Aturada i posada en marxa del vehicle. S'ha de poder aturar el vehicle i tornar-lo a posar en marxa en qualsevol moment prement la barra espaiadora.
- Velocitat de moviment. La interfície gràfica ha de permetre incrementar i decrementar la velocitat del moviment que realitza el vehicle.
- Presentació de l'escena amb il·luminació i eliminació de parts amagades. Heu de mostrar l'escena amb realisme. L'usuari ha de poder modificar —a través de la interfície— els colors ambient, difús i especular d'un llum (`GL_LIGHT0`) que es troba en una posició donada respecte de la càmera. També a través de la interfície s'haurà de poder modificar la seva posició relativa a la càmera. Finalment, s'ha de poder encendre i apagar el llum. L'escena s'ha de visualitzar tenint en consideració el llum (si és actiu), el material de les cares dels objectes i l'eliminació de parts amagades.
- La interfície haurà de tenir un *check-box* que activi i desactivi el suavitzat d'arestes, que s'aplicarà a tots els objectes de l'escena llevat del cotxe i els trams.

3 Omplert de polígons i il·luminació

Recordeu que OpenGL s'ha de veure com una màquina d'estats. En l'entrega anterior, heu pintat els models en filferros enviant al pipeline gràfic les cares, però indicant que volíeu sols el contorn dels polígons (col·locant aquesta màquina d'estats a l'estat corresponent mitjançant una crida a `glPolygonMode(GL_FRONT_AND_BACK, GL_LINE)`).

També disposeu de codi per a modificar aquest estat, per fer que OpenGL rasteritzi l'interior dels polígons. Recordeu que el primer paràmetre de `glPolygonMode()` permet associar un comportament diferent amb les cares que miren cap endavant i a les que miren cap enrera (cosa que OpenGL determinarà segons l'ordre en què giren els vèrtexs i no basant-se en cap normal que hagueu enviat al pipeline). El segon paràmetre, indica a OpenGL que volem que pinti no sols el contorn, sinó també l'interior dels polígons (`GL_FILL`).

Si només fem això, veurem els polígons omplerts, però els models (potser) donaran una sensació estranya en girar-los. La raó és que OpenGL no està eliminant les parts amagades, sinó que cada instrucció de dibuixar sobreescriu el que ja hi havia al frame buffer. El resultat és que el darrer que es pinta sembla ser "davant" independentment de des d'on

ens ho mirem. Per tal que OpenGL elimini les parts amagades usant l'algorisme de Z-buffer cal activar aquest mecanisme. Per això es fa servir altre cop la crida `glEnable()`. L'opció que ens interessa ara és `GL_DEPTH_TEST`. Aquesta crida sols us cal fer-la un cop a la inicialització, de moment. També caldrà modificar la crida a `glClear()` per tal que esborri el z-buffer a més a més del buffer de color abans de cada nova visualització. Totes aquestes inicialitzacions ja les heu fet servir en la pràctica anterior en què podíeu visualitzar en filferros o amb eliminació de cares ocultes.

Nota: Incidentalment, el funcionament del z-buffer es veu afectat per la col·locació dels plans de retallat anterior i posterior. El motiu pel qual cal ajustar dins del possible la posició d'aquests plans al model, és que es disposa d'una precisió finita per emmagatzemar les profunditats. OpenGL usarà els bits de què disposa per a codificar les profunditats compreses entre aquests dos plans, de manera que hi ha més valors representables prop de l'observador, que prop del pla posterior de retallat. De fet, si `near` és la distància al pla de retall anterior, aproximadament la meitat dels valors possibles de z es troben entre `near` i `2*near`.

Per tal d'accelerar el càlcul de l'eliminació de parts ocultes per z-buffer, es pot, generalment, efectuar un procès de *back-face culling*. OpenGL permet activar aquest procès amb `glEnable(GL_CULL_FACE)`; per defecte es considera que es volen eliminar la part de darrera de les cares (`GL_BACK`) i que les cares estan orientades en sentit anti-horari(`GL_CCW`). Per modificar aquestes inicialitzacions es poden utilitzar les comandes `glCullFace()` i `glFrontFace()`.

Tanmateix, en la Pràctica 1 vàreu observar que, malgrat només pintar-se les cares visibles, els models tenien una aparença "plana". La causa és que no teniem definit cap entorn d'il·luminació i les cares no tenien assignat un material sinó un color constant.

Per a veure els polígons amb la simulació d'il·luminació, heu de donar a OpenGL informació suficient sobre els materials dels objectes i la il·luminació de l'entorn en què es troben. Haureu de substituir la/les crida/es a `glColor()` que tingueu al bucle que envia a dibuixar les cares de cada objecte, per un seguit de crides a `glMaterial()` definint totes les propietats del material declarat per cada cara (color difús, ambient, especular,...). [En realitat en activar la simulació de la il·luminació OpenGL deixarà de fer cas de les crides a `glColor()`, pel que no és estrictament necessari treure-les].

A més a més, caldrà que envieu a OpenGL una normal per cada cara, necessària per a aplicar els models empírics d'il·luminació. Aquesta normal, en aquesta pràctica, es calcula per a cada objecte si crideu al mètode `CalculaNormals()` que heu incorporat al vostre codi segons us hem indicat en la introducció d'aquest document; aquesta crida pot fer-se tan bon punt s'hagin carregat els models. La comanda necessària per a assignar una normal a una cara en OpenGL és `glNormal()` i s'ha de cridar abans d'enviar les coordenades dels vèrtexs de la cara.

Quan dibuixeu els models podeu fer servir la crida `glEnable(GL_NORMALIZE)` per tal d'indicar a OpenGL que normalitzi les normals. Encara que les normals que tingueu calculades siguin unitàries en coordenades del model, no ho seran en coordenades de l'aplicació (ni de l'observador) si la vostra matriu `MODELVIEW` inclou escalats (com és el cas de la vostra pràctica).

Quan tenim definits els colors dels materials, els llums actius (i correctament inicialitzats) i la il·luminació activada, OpenGL pot fer l'omplert de les cares de dues maneres diferents: donant-los-hi un color constant (`GL_FLAT`) o simulant el color de cada píxel a partir del color dels vèrtexs del polígon. El que ha de fer s'indica mitjançant una crida a `glShadeModel()`, que accepta els paràmetres `GL_FLAT` i `GL_SMOOTH`. El segon és el valor per defecte quan s'inicialitza OpenGL. El càlcul del color dels vèrtexs el calcula OpenGL aplicant models empírics d'il·luminació.

La informació de l'entorn d'il·luminació que us cal afegir per a poder tenir realisme es discuteix més avall. OpenGL té definit per defecte una llum (la `GL_LIGHT0`) i, per tant, si l'enceneu hauríeu de poder veure "quelcom" una mica més realista sense necessitat de completar la resta de declaració de llums, només realitzant les crides a `glMaterial()` i a `glNormal()` per a cada cara.

Nota: En aquesta pràctica, s'utilitzarà el llum `GL_LIGHT0`, modificant els seus valors per defecte, per a codificar el que anomenem llum de càmera (veure sessió 1 de l'apartat 5).

3.1 Llums

OpenGL permet definir una sèrie de llums de diferents característiques. Els llums poden ser posicionals (és a dir tenir una posició definida en el model) o direccionals (és a dir que tots els raigs que emeten tenen la mateixa direcció, com la llum del sol en un exterior); poden ser spots (com els usats en teatres, per exemple, per a fer-nos una idea), o poden ser omnidireccionals (com una bombeta penjada, sense pantalla (bé, aproximadament, és clar)); poden presentar atenuació (a mida que ens en allunyem) o no. Aquí no ens estendrem en totes les possibilitats. Tingueu en compte, si decidiu experimentar amb elles, que són moltes i us consumiran molt temps.

Per controlar els llums, doncs, hi ha un conjunt de crides de l'API amb les que us haureu de familiaritzar. El nombre total de llums disponibles depèn de cada instal·lació (es pot esbrinar consultant `GL_MAX_LIGHTS`), però l'estàndard garanteix que n'hi haurà almenys vuit. Per a distingir-los, es fan servir les constants `GL_LIGHT0`, `GL_LIGHT1`, `GL_LIGHT2`, ... Un programa pot aprofitar el fet que aquestes constants són consecutives per a referir-se als diferents llums en bucles, per exemple (és a dir que està garantit que, per exemple, `GL_LIGHT4==GL_LIGHT0+4`). Les crides de l'API més importants que necessitareu en relació als llums són:

- `glLight()`: Aquesta crida us permet modificar deu paràmetres que defineixen cadascun dels llums, incloent la posició/direcció (aquest paràmetre està "sobrecarregat") i el color (distingint el color ambient, difús i especular). Per a obtenir bons resultats, es recomana fer servir intensitats de llum ambient petites o nul·les. En cas contrari, la il·luminació resultarà semblant a l'obtinguda amb `GL_FLAT`, perquè el càlcul d'il·luminació ambient no depèn de la normal. En el moment de cridar a `glLight()` per a fixar la posició d'un llum, la posició indicada serà afectada per la transformació a coordenades d'observador (`MODELVIEW`) vigent en aquell instant. Si fem la crida tenint la identitat a la pila de matrius de `MODELVIEW`, per tant, les coordenades s'interpretaran com coordenades d'observador, mentre que si hi ha una matriu diferent,

no. Si la matriu actual és la que transforma de coordenades de model a coordenades d'observador, per exemple, les coordenades del llum s'interpretaran en coordenades de model.

- `glEnable()`, `glDisable()`: Aquestes crides, que ja hem trobat abans, permeten activar i desactivar el càlcul d'il·luminació (`GL_LIGHTING`), i cadascun dels llums (`GL_LIGHT0`, `GL_LIGHT1`, ...).
- `glLightModel()`: Aquesta crida us permet controlar aspectes més subtils del càlcul d'il·luminació. Us pot ser d'utilitat el triar el `GL_LIGHT_MODEL_LOCAL_VIEWER`.

En el manual d'OpenGL, o en el seu defecte en el llibre de l'assignatura, trobareu tota la informació (paràmetres, tipus dels paràmetres, opcions,...) de les instruccions anteriors.

3.2 Interfície en Qt de la il·luminació

Per a ajudar-vos en la part de la interfície per a la il·luminació (modificació dels paràmetres dels diferents llums, activació/desactivació dels llums, etc.) us passem un diàleg de Qt que inclou tota la part d'interfície que es requereix a la pràctica respecte a inicialització de paràmetres dels llums, al directori `/assig/vig/sessions/S2.2/uiLlums_qt4`. Juntament amb la implementació d'aquest diàleg, teniu un fitxer "main.cpp" que en fa ús, per a què pugueu tenir un exemple de com incloure'l en la vostra pràctica. Per descomptat, podeu decidir usar-lo o no, o modificar quelcom que no us agradi, però tingueu en compte que cal que la vostra pràctica ofereixi aquesta funcionalitat en la interfície.

4 Guió detallat.

Per a la realització d'aquesta pràctica us proposem una distribució de les tasques a realitzar en les diferents sessions de laboratori. Aquestes indicacions les donem amb la intenció que us serveixin d'orientació per a distribuir-vos la càrrega de la pràctica. També us donem informacions i "pistes" que us poden ser d'utilitat en el disseny i programació del codi. En les classes de laboratori s'introduiran els conceptes requerits en cada sessió.

No cal que us ajusteu exactament a aquesta distribució, però si veieu que hi ha una desviació important respecte d'aquest guió, cal que us esforceu en avançar més ràpid, o difícilment acabareu la pràctica per a la data d'entrega.

Sessió 1 En aquesta primera sessió haureu d'implementar l'animació del vehicle.

- Incloure un *timer* que refresqui l'escena cada .04 segons quan l'animació està activada. Cada vegada que es repinta, es farà avançar, en aquest cas, el temps (és a dir s'actualitzarà la posició del vehicle).

- Implementar el moviment del vehicle fent que aquest canviï de posició en la direcció de moviment que ve marcada pel tram de carretera sobre el que es troba, i en funció de la velocitat que porti. En l'estructura de dades de cada tram de carretera disposem d'un vector de 4 enters (`seguents`) on en cada posició es guarda l'índex, en cas que existeixi, del tram següent en aquella direcció (en cas que no existeixi següent tram en aquesta direcció a la posició del vector hi haurà un -1). Les direccions són X+, X-, Z+ i Z-, i estan en aquest ordre en el vector. Aquestes direccions són absolutes respecte del sistema de coordenades de l'aplicació.
- Implementar el gir en el moviment del vehicle. Quan el tram de carretera indica que cal fer un gir aquest haurà de ser suau, per tant, s'iniciarà el gir quan el vehicle entri al tram i s'acaba quan surt del tram i ja ha girat 90^o en el sentit indicat.
- Afegir el necessari per a permetre a l'usuari aturar o reiniciar el moviment del vehicle prement la barra espaiadora.
- Afegir a la interfície un `slider` per a poder controlar la velocitat del vehicle.

Sessió 2 En aquesta sessió prepararem l'aplicació per a fer servir models d'il·luminació.

- Feu els canvis necessaris per a incloure les normals de les cares a les estructures de dades.
- Activeu el *face-culling*. Observeu l'efecte sobre la renderització de l'escena, fent girar la càmera.
- Implementeu les crides a `glMaterial()` per a definir el color de les cares i la corresponent crida a `glNormal()`. Recordeu fer la crida a `glEnable(GL_NORMALIZE)` per a que OpenGL faci la normalització de las normals. En aquest punt, si habiliteu el càlcul d'il·luminació amb `glEnable(GL_LIGHTING)` i enceneu el llum `GL_LIGHT0`, s'hauria de veure l'escena amb certa il·luminació. Recordeu que el llum `GL_LIGHT0` té els seus paràmetres definits per defecte; consulteu el manual per entendre les imatges que obteniu tenint en consideració aquests paràmetres. En la següent sessió afegireu una interfície per a modificar-los.

Sessió 3 Durant aquesta sessió, inserireu l'editor de llums (o una altra interfície equivalent de la vostra preferència), i fareu que la llum de càmera sigui editable mitjançant aquesta interfície. A més, implementareu l'opció de suavitzar les arestes dels models referenciats a `lreferencies`.

- Copieu el directori `uiLlums_qt4` de `/assig/vig/sessions/S2.2` en el vostre directori local de la pràctica. En aquest directori està definida la classe `llum` que necessiteu per a definir els focus de llum i també hi ha els fitxers necessaris per al widget Qt que us oferim per a facilitar-vos la modificació dels paràmetres dels llums. Naturalment, modifiqueu els vostres .pro per a que el que feu servir d'aquest codi es compili i munti al vostre programa.

- Habiliteu el mode d'il·luminació amb viewpoint local a efectes de millorar el resultat de la visualització (`glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER,GL_TRUE)`). Observeu l'efecte de deshabilitar el `GL_LIGHT_MODEL_LOCAL_VIEWER`. Compareu el que succeeix si sou lluny de l'escena, o molt a prop d'alguns dels objectes (trieu objectes que siguin reflectants).
- Feu que la llum de la càmera sigui modificable des de la interfície.
- Afegiu codi que calculi normals per vèrtex a tots els objectes d'`lreferencies`, i el codi necessari (a la interfície i als mètodes de renderització) per a poder dibuixar l'escena amb suavitzat d'arestes. Vigileu que el codi que feu servir tingui en compte l'eficiència.
- Afegiu a la interfície un mecanisme per a encendre i apagar el llum.
- Aprofiteu el temps restant per a polir la interfície, i netejar el codi si cal. Recordeu que la qualificació de la pràctica té en compte la correctesa i elegància de la implementació, i la usabilitat de la interfície.

Finalment, podeu aprofitar per a afegir un fitxer `comentaris.txt` al directori arrel de la vostra pràctica si voleu comentar qualsevol incidència o diferència a la vostra implementació. No cal que aquest fitxer existeixi, però si hi és ha de contenir informació rellevant. Un fitxer amb un contingut de l'estil "He fet el que demanava l'enunciat" resultarà en una penalització a la vostra nota.