

Pràctica de Gràfics (Gràfics per Computador i Tècniques Gràfiques) Curs 11-12 Visualització i manipulació d'objectes

Professors:

Ferran Diego

Felipe Lumbreras

Introducció

Aquesta pràctica vol servir com a introducció al món dels gràfics per computador. Farem una petita aplicació que permeti visualitzar i manipular objectes 3D.

Per fer-la ens aprofitarem del potencial que dona la llibreria gràfica **OpenGL**, i altres llibreries relacionades amb ella con la GLUT. El llenguatge de programació recomanat és el VisualC++ que és el que disposarem a classe de pràctiques.

Grups

No més de dues persones per grup. La feina està pensada per fer-se entre dos. Si alguna persona vol fer-la de manera individual assumeix el sobre esforç de forma voluntària.

Tots dos han de treballar en la pràctica, per tant a les sessions d'avaluació de pràctiques (lliuraments) tots els membres del grup han de ser-hi presents i a tots se'ls hi faran preguntes de seguiment.

Avaluació

Continuada. Quatre lliuraments en total, tres dels quals són obligatoris. L'últim es reserva per fer el lliurament4, els retards dels tres primers o per pujar la nota dels lliuraments anteriors.

La nota de pràctiques constitueix un 40% de la nota final de l'assignatura que es reparteix en lliurament1 = 10%, lliurament2 = 15%, lliurament3 = 15%. S'ha d'aprovar la part de pràctiques amb 5 o més per poder optar a superar l'assignatura i cadascun d'aquests tres lliuraments obligatoris s'han de superar individualment. Hi ha un darrer lliurament opcional que serveix per pujar nota (lliurament4 = 15%)

Se seguirà el següent criteri a l'hora de puntuar cada lliurament:

- S'entrega la part bàsica de cada lliurament a temps i funcionat correctament: 7
- Es fan les millores o algunes de les millores: de +0 a +3 (a criteri del professor).
- Retards en els lliuraments: -2
- Errors lleus de funcionament: de -0 a -2 (a criteri del professor)
- Errors greus impliquen no superar el lliurament i haver-ho de tornar a presentar al següent lliurament.

S'entén que aquesta guia és només per un desenvolupament normal de la pràctica. Els casos de còpia i poca implicació d'algun membre de l'equip tindran una avaluació a criteri del professor.

Lliuraments

1.- Lectura i visualització bàsica d'un objecte

- Escena, creació, càmera
- Lectura d'objectes simples
- Visualització mitjançant punts, arestes i cares
- Projectió ortogràfica i perspectiva

2.- Múltiples objectes. Visualització avançada

- Lectura de més d'un objecte
- Lectura completa d'objectes
- Caixa mínima i centre de l'objecte
- Càlcul de normals
- Dibuix de normals, eixos, caixa mínima
- Visualització amb il·luminació

3.- Transformacions. Manipulació.

- Translació, rotació i escalat.
- Manipulació amb el moviment del ratolí
- Manipulació d'objectes per separat

4.- Millores

- Millores lliures a partir del resultat anterior

Lliurament 1.- Lectura i visualització bàsica d'un objecte

Escena, creació, càmera

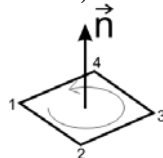
Per tal de crear l'aplicació farem ús de la llibreria GLUT. OpenGL és només una llibreria gràfica i per tant no té funcionalitats associades a la manipulació de finestres ni d'events. GLUT ens permet de forma ràpida construir una aplicació independentment de plataforma, amb la possibilitat de gestionar events de teclat i del ratolí. La llibreria GLUT no ve instal·lada per defecte en el Visual ni al sistema operatiu, així que el primer que s'ha de fer és fer aquesta instal·lació: <http://www.xmission.com/~nate/glut.html>. El que podem fer és mantenir una còpia de glut32.dll, glut32lib, glut.h al nostre projecte per no haver de tornar a instal·lar i configurar cada cop que es netegi el sistema.

Agafem com a punt de partida l'esquelet de programa que trobem al Red Book (Capítol 1). <http://www.gprogramming.com/red/chapter01.html> adaptant-lo a les nostres necessitats. Veiem que l'estructura del programa té una part d'inicialització i després un conjunt de funcions que estableixen com s'ha de gestionar els events de reescalat, de teclat o de ratolí i quina és la funció de dibuix principal mitjançant unes funcions de callback, per finalitzar amb un bucle que serà l'encarregat de dibuixar i gestionar tots els events.

Per adaptar aquest esquelet de partida a la llibreria glut hem preparat un codi base per a començar a l'apèndix Inici amb OpenGL.

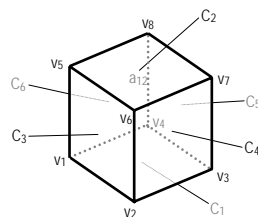
Lectura d'objectes (simple)

El model d'objectes que utilitzarem a la pràctica és un model explícit de cares. La descripció de l'objecte consta d'informació geomètrica per un costat amb vèrtexs definits per coordenades x,y,z i per altre d'informació de cares (índex dels vèrtexs que componen cada cara, en l'ordre que defineix la seva orientació).



Construïrem un objecte des del començament per veure quina informació conté i com s'estructura. Per fer això agafarem un editor de text i escriurem els valors que defineixen un cub. Per definir un objecte hem de tenir la informació geomètrica (que en el nostre cas seran els vèrtexs). Utilitzarem decimals (floats) ja que per un objecte qualsevol la posició d'un vèrtex a l'espai no ha de coincidir necessàriament amb coordenades enters.

```
0.0 0.0 0.0
0.0 0.0 1.0
0.0 1.0 0.0
0.0 1.0 1.0
...
```



Però amb aquesta informació no en tenim prou, només podríem dibuixar els punts, no sabem com s'han d'unir. Hem d'incloure nova informació, les arestes i les cares.

El model explícit de cares el que fa és "amagar" les arestes, ficant les dues informacions (cara i arestes) a la vegada. Cada cara serà la seqüència dels índexs dels vèrtexs que la

formen, de manera que el seu recorregut defineixi totes les arestes i per tant tota la cara. En aquest cas utilitzarem enters (naturals) ja fem referència als índexs dels vèrtexs. El primer índex serà el zero (notació C).

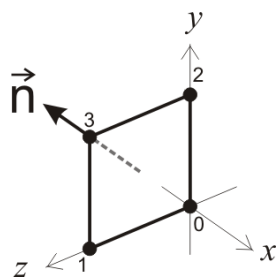
Exemple de cara: 0 1 3 2

El sentit de recorregut d'aquests vèrtexs dóna altra informació que s'aprofitarà més endavant per definir el sentit de la normal a la cara. Convé que totes les cares d'un objecte "apunten" en la mateixa direcció. Per fer això utilitzem la regla de la mà dreta: agafant els vèrtexs amb la mà dreta en l'ordre assenyalat, el polze apunta en la mateixa direcció (o tots cap a fora o tots cap a dins del objecte).

Tornant a la definició del fitxer de dades, per simplificar la lectura d'aquesta informació numèrica, tant dels vèrtexs, de les cares, com del índexs de vèrtexs que formen cada cara, afegirem informació extra. Indicarem quants elements tenim davant de cada bloc. Quants vèrtexs té l'objecte i el conjunt de coordenades que defineixen el vèrtexs, després quantes cares tenim i darrere la informació que defineix cadascuna de les cares indicant al principi per cada cara quants vèrtexs la componen.

seguint amb l'exemple del cub, tindrem:

```
8
0.0 0.0 0.0
0.0 0.0 1.0
0.0 1.0 0.0
0.0 1.0 1.0
...
6
4 0 1 3 2
...
```



Dibuixeu en paper un cub i numereu els seus vèrtex i cares. Després feu un fitxer de text que el representi.

Afegiu al vostre codi les funcionalitats necessàries per llegir qualsevol objecte en un fitxer de text que tingui aquesta estructura. Recordeu-vos de fer la reserva de memòria, i la destrucció. En aquest lliurament comencem amb la informació més bàsica (vèrtexs i cares) que acabarem de completar en el següent lliurament. Si us sobra temps podeu completar la lectura de la resta de camps per tal d'avançar la feina segons s'explica més endavant en la "Lectura completa d'objectes".

Visualització mitjançant punts, arestes i cares

Per visualitzar la informació dels vèrtexs de forma simple, farem un bucle que recorri tots ells dibuixant un punt. D'aquesta manera tindrem l'objecte 3D a pantalla representat amb punts. En aquest cas la primitiva de dibuix és el punt (GL_POINTS).

Per tal de fer la representació mitjançant arestes el que farem és fer un recorregut per totes les cares i per cadascuna d'aquestes, connectarem en ordre tots els vèrtexs mitjançant la primitiva de dibuix línia tancada (GL_LINE_LOOP).

Per acabar, i amb la mateixa estructura de recorregut que acabem de fer per les arestes podem tancar les cares mitjançant polígons (GL_POLYGON).

Projecció ortogràfica i perspectiva

Anem a veure l'escena amb dos tipus diferents de projeccions. La primera d'elles consisteix a veure una projecció paral·lela on les línies paral·leles a la realitat es mantenen paral·leles al dibuix. Aquesta projecció és típica de programes de dibuix assistit per ordinador. La segona és una vista en perspectiva que és el que estem acostumats a veure (per exemple l'ull humà, una càmera, ...) on les línies que són paral·leles en la realitat, en aquest cas, es troben en un punt del dibuix (punt de fuga) i on la mida dels objectes representats al dibuix depèn de la distància des d'on hem fet aquesta representació (més petits quant més allunyats estiguin). A l'esquerra de la figura una projecció ortogonal de tres cubs i a la dreta la projecció perspectiva.



Per la primera projecció utilitzarem una funció de la llibreria bàsica:

```
void glOrtho(GLdouble left, GLdouble right, GLdouble bottom,  
            GLdouble top, GLdouble nearVal, GLdouble farVal);
```

Per la segona farem ús d'una funció de la llibreria d'utilitats:

```
void gluPerspective(GLdouble fovy, GLdouble aspect, GLdouble zNear,  
                  GLdouble zFar);
```

Control

En aquest lliurament assignarem a les següents entrades de teclat les funcionalitats corresponents

“1”: lectura d'un fitxer amb nom prefixat **“1.txt”**. Comencem amb **“.txt”** per diferenciar la resta de fitxers d'objectes **“.3d”** on es canviarà el format una mica.

“p”: projecció ortogràfica/ projecció perspectiva

“q”: primitiva de dibuix punt (activada/desactivada)

“w”: primitiva de dibuix línia (activada/desactivada)

“e”: primitiva de dibuix polígon (activada/desactivada)

MILLORES:

Llistes de visualització

Per tal de fer més eficient la visualització d'objectes i la seva manipulació farem servir Llistes de visualització (*Display List*). Podem trobar documentació sobre aquest punt al Capítol 7 del Red Book (<http://www.glprogramming.com/red/chapter07.html>).

Amb aquesta tècnica els elements a visualitzar es calcularan un únic cop i OpenGL s'encarregarà de guardar aquesta informació per la seva posterior visualització. Per tant, en la funció que fa la visualització farem crides a aquestes llistes prèviament generades amb una crida a la funció `glCallList(Llista_X)`.

Per generar una llista de visualització direm a OpenGL que activi una d'aquestes llistes, generarem el gràfic (de la mateixa manera que ho fariem si volguéssim visualitzar-lo) i tancarem aquesta llista de visualització. Això ho farem per totes les entitats gràfiques que anem a dibuixar i el millor lloc per construir-les és un cop s'ha fet la lectura i la resta de càlculs relacionats amb informació que no hem llegit (càlcul de normals, generació de coordenades de textures, ...).

```
// generació de llista de visualització per X
if (glIsList(Llista_X))
    glDeleteLists(Llista_X, 1);
Llista_X=glGenLists(1);
glNewList(Llista_X, GL_COMPILE);
//codi per dibuixar l'element X
glEndList();
```

A les llistes de visualització s'accedeix mitjançant un índex (enter) que hem anomenat en aquest cas `Llista_X` (on `X` pot ser Punts, Arestes, Cares, Eixos, Caixa, ...), aquestes variables han de ser del tipus `GLuint`.

Lectura d'objectes 3D estàndards

Aquesta millora consisteix en llegir objectes amb un format més elaborat que el format bàsic de la pràctica. Com a exemples de formats podem tenir:

Geomview Object File Format (OFF): <http://people.sc.fsu.edu/~burkardt/data/off/off.html>

Stereolithography (STL): [http://en.wikipedia.org/wiki/STL_\(file_format\)](http://en.wikipedia.org/wiki/STL_(file_format))

Lliurament 2.- Múltiples objectes. Visualització avançada

Lectura de més d'un objecte

Afegim més d'un objecte a l'aplicació. Establim un nombre màxim de 9 objectes en forma d'array que associarem després a les tecles de l'1 al 9.

Lectura completa d'objectes

Al lliurament anterior els objectes estaven representats mitjançant punts i cares, però aquesta informació resulta insuficient si volem il·luminar i texturitzar els objectes. Anem a complicar una mica el model d'objecte afegim una mica més d'informació: normal a la cara, normals als vèrtexs, coordenada de textura.

Per tal de poder tenir objectes amb diferent informació el fitxer d'objecte 3d començarà per una capçalera que indicarà què hi ha a dins i per tant ens permetrà llegir còmodament només el que hi ha.

Nota: els fitxers seran correctes i no s'haurà de fer la comprovació de la seva coherència.

** veure l'apèndix Format de fitxers .3d per una descripció més detallada de les possibilitats.*

Caixa mínima i centre de l'objecte

Aprofitarem la lectura i que hem de recórrer tota la informació per fer el càlcul de la caixa mínima (*Bounding Box*) de l'objecte. Aquesta informació formarà part de la representació de l'objecte en memòria i consisteix en dos punts: $(x_{\min}, y_{\min}, z_{\min})$, $(x_{\max}, y_{\max}, z_{\max})$ que defineixen els extrems de la caixa que conté l'objecte. Calcular la caixa mínima No és res més que cercar aquestes coordenades mínimes i màximes. Dibuixar-la consistirà en mitjançant aquests dos punts reconstruir el paral·lelepípede (caixa) que representen. El centre de l'objecte el podem definir com el centre del *bounding box*, és a dir, el punt mig d'aquests dos punts que hem calculat abans. Hi ha més maneres per definir el centre de l'objecte, per exemple el centre de masses, però per la nostra aplicació tenim prou amb el centre de la caixa.

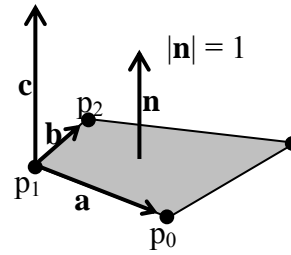
Càlcul de normals

Els arxius .3d contenen informació diversa sobre els objectes. Entre aquesta informació podem trobar les normals. Aquestes normals són molt importants per poder calcular com es reflexa la llum sobre la seva superfície.

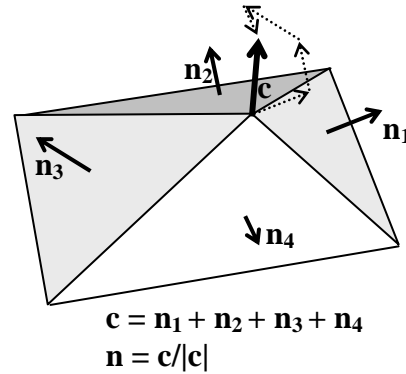
Si el fitxer no inclou la informació de les normals, les hi calcularem nosaltres. L'objectiu final és que cada objecte tingui tant normal a les cares com normals als vèrtex. S'ha de dir que això només és possible si el fitxer conté informació de les cares.

El càlcul de les normals s'ha de fer en ordre: primer les cares i després els vèrtexs ja que necessitem les primeres per a calcular les segones.

- **Normals a les cares:** agafem els tres primers vèrtexs que defineixen la cara (p_0, p_1, p_2) i calculem dos vectors mitjançant aquests punts $\mathbf{a} = p_0 - p_1$ i $\mathbf{b} = p_2 - p_1$. Amb l'ajuda del producte vectorial calculem el vector perpendicular a aquests dos $\mathbf{c} = \mathbf{a} \times \mathbf{b}$. I només ens queda normalitzar aquest vector per obtenir la normal (dividir cada component pel seu mòdul $\mathbf{n} = \mathbf{c}/|\mathbf{c}|$).



- **Normals als vèrtexs:** les normals als vèrtexs seran la mitjana de les normals a les cares que conflueixen en aquest vèrtex. Pel seu càlcul sumarem les normals de totes la cares que tenen aquest vèrtex i normalitzarem al final per obtenir una normal unitària.



En ambdós casos hem de fer un bucle per totes les cares. En el primer per cada cara calcularem la seva normal i en el segon cas per cada cara sumarem el vector normal de la cara a tots els vectors normals dels vèrtexs que la defineixen. Un cop hem fet això podem recórrer totes les normals als vèrtexs normalitzant-les. Pel cas de les normals als vèrtexs, si apliqueu la definició directament arribareu a un esquema amb tres bucles i una condició, en aquest cas la lectura és molt lenta, és millor fer servir el primer bucle que recorri les cares.

Recordatori:

Sigui	$\mathbf{a} = (a_x, a_y, a_z)$ i $\mathbf{b} = (b_x, b_y, b_z)$
Suma:	$\mathbf{a} + \mathbf{b} = (a_x + b_x, a_y + b_y, a_z + b_z)$
Resta:	$\mathbf{a} - \mathbf{b} = (a_x - b_x, a_y - b_y, a_z - b_z)$
Mòdul:	$ \mathbf{a} = \text{sqrt}(a_x \cdot a_x + a_y \cdot a_y + a_z \cdot a_z)$
Normalitzar:	$\mathbf{a}/ \mathbf{a} = (a_x/ \mathbf{a} , a_y/ \mathbf{a} , a_z/ \mathbf{a})$
Prod. escalar:	$\mathbf{a} \cdot \mathbf{b} = a_x \cdot b_x + a_y \cdot b_y + a_z \cdot b_z$
Prod. vectorial:	$\mathbf{a} \times \mathbf{b} = (a_y \cdot b_z - a_z \cdot b_y, a_z \cdot b_x - a_x \cdot b_z, a_x \cdot b_y - a_y \cdot b_x)$

Dibuix de normals, eixos, caixa mínima

Aquesta nova informació que hem llegit o calculat anem a visualitzar-la també.

- **Normals:** El primer que farem serà representar les normals (si les tenim) tant de les cares com dels vèrtex. Una normal a un vèrtex no deixa de ser una línia que va des del vèrtex a un punt situat a una unitat de distància d'aquest punt en la direcció especificada per la normal. En el cas de la normal a la cara, aquesta va des del centre de la cara al punt especificat per la direcció de la normal a una distància de 1 unitat. Podem establir un factor per controlar la llargada de la normal i que sigui proporcional a la mida de l'objecte.

- **Caixa Mínima:** Partint dels dos punts que defineixen la caixa mínima hem de dibuixar el paral·lelepípede que la representa. Amb dotze línies la tenim enllestida.
- **Eixos:** La visualització dels eixos correspon tant als eixos de coordenades de la escena (normalment el 0,0,0) com als eixos de coordenades de cada objecte. És a dir, 3 vectors de longitud 1 que representen els eixos x,y,z o de l'origen de coordenades o del centre del objecte. Per dibuixar-los, podeu definir els eixos unitaris (1,0,0), (0,1,0) i (0,0,1) i dibuixar línies de l'origen (0,0,0) a aquests punts. Podeu afegir més línies per donar l'aspecte de fletxes.

Visualització amb il·luminació

L'objectiu d'aquesta part és familiaritzar-se amb els models d'ombrejat, control d'il·luminació i els diferents models d'il·luminació que té OpenGL. El model d'il·luminació que fem servir considera el material de l'objecte i la normal a la superfície. Hem de pensar que per aconseguir un bon efecte hem de tenir definides correctament les normals dels objectes.

El primer que farem serà mirar com es defineix una llum, Capítol 5 del Red Book:

<http://www.glprogramming.com/red/chapter05.html>

Per fer-ho, entre d'altres, hem de especificar les seves característiques, si és puntual, direccional, focal, la seva posició, la direcció de la llum, ...

El que demanem és que posicioneu al menys una font de llum i activeu la llum ambiental, que es pugui modificar el model de il·luminació (`glShadeModel`). Opcionalment podeu modificar el tipus de material de l'objecte. Veureu que l'efecte de la llum varia considerablement.

Un cop definida la llum, o les llums, hem d'activar-les a OpenGL: 1. Si cal desactivar-les farem servir `glDisable(GL_LIGHTING)`. Això activa i desactiva de forma global les funcions relacionades amb llums.

Control:

En aquest lliurament assignarem a les següents entrades de teclat les funcionalitats corresponents

“1”...“9”: lectura d'un fitxer **“.3d”** que associarem als objectes **1...9**

“p”: projecció ortogràfica/ projecció perspectiva

“q”: primitiva de dibuix punt (activada/desactivada)

“w”: primitiva de dibuix línia (activada/desactivada)

“e”: primitiva de dibuix polígon (activada/desactivada)

“r”: representació de normals (activada/desactivada)

“t”: representació de la caixa mínima (activada/desactivada)

“y”: representació dels eixos (activada/desactivada)

“u”: il·luminació en l'escena (activada FLAT, activada SMOOTH, desactivada)

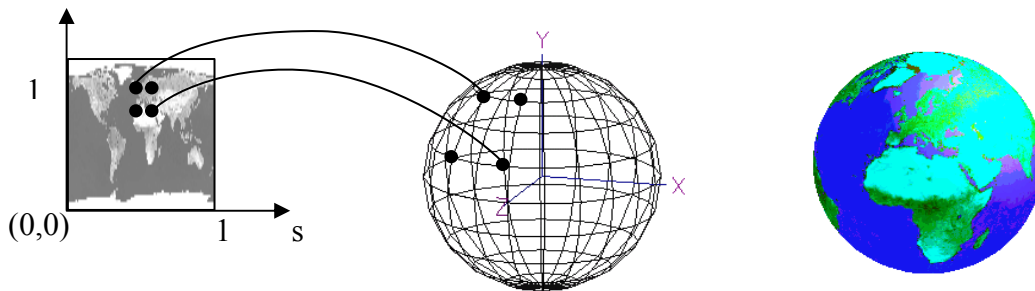
MILLORES:

Visualització amb textura

Aquesta funcionalitat ens permet fer una visualització més realista. Les textures es generen a partir d'imatges, generalment fitxers bitmap (.bmp) de 24 bits i amb dimensions potència de 2. Per a carregar aquestes imatges en memòria buscarem les funcions adients (classes de tipus bitmap, bmp, ...) i per plasmar la textura a sobre dels objectes farem ús del Capítol 9 del Red Book: <http://www.glprogramming.com/red/chapter09.html>

- **Coordenades de textura**

La imatge que representa la textura té un sistema de coordenades propi (s,t) i omple una zona en el pla s,t de $[0,1] \times [0,1]$. El *mapping* de textura consistirà en assignar a cada vèrtex de l'objecte un valor (s,t) que indiqui quina zona de textura li correspon. OpenGL s'encarregarà d'interpol·lar els valors entre els diferents vèrtexs. Depenent de la informació que vingui en el fitxer de dades (primera línia de text que identifica les dades) actuarem d'una manera o una altra per assignar les coordenades de textura. En ambdós casos utilitzarem la funció `glTexCoord2d(s,t)` que és l'encarregada d'assignar les coordenades de textura en OpenGL.



- **Fitxers de dades amb valors de textura**

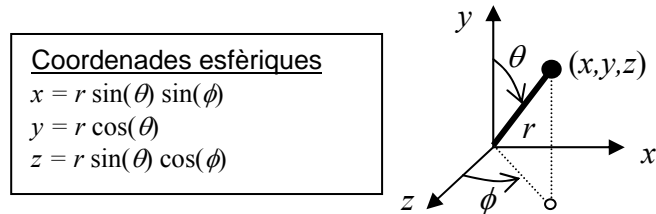
Aquest és el cas més senzill i el que farem serà assignar a cada vèrtex el valor de textura que tinguem en el fitxer de dades. De la mateixa manera que fèiem amb les normals, assignarem les coordenades de textura de cada vèrtex abans de la definició del propi vèrtex. Per a cada vèrtex tindrem per tant:

```
glNormal3d(nx, ny, nz);  
glTexCoord2d(s, t);  
glVertex3d(x, y, z);
```

- **Fitxers de dades sense coordenades de textura**

En aquest cas generarem aquestes coordenades (s,t) nosaltres mateixos. Una manera de fer-ho és pensar que l'objecte està envoltat per una esfera. Mirem per a cada vèrtex de l'objecte on es projecta el punt sobre aquesta esfera. Per fer això mirem la prolongació de la recta que passa pel centre de l'objecte i el punt, i calculem el punt de tall amb l'esfera. Sobre l'esfera tenim un sistema de coordenades esfèric (φ, θ) que ens indica la coordenada de textura que hem d'assignar al punt que estem tractant. Aquests angles els escalarem al rang $[0,1]$. Una manera de fer això és calcular el vector que va del centre de l'objecte al punt en qüestió, normalitzem aquest vector, i a partir dels valors (x,y,z) del vector calculem els valors (φ, θ) d'aquest vector expressat en coordenades esfèriques.

Podem arribar als valors (ϕ, θ) aïllant de les equacions que ens donen x, y, z . Es pot obtenir θ de la coordenada y . I es pot obtenir ϕ de la relació entre x, z . S'ha de tenir en compte que els valors de θ van de $[0, \pi]$ i que ϕ va de $[-\pi, \pi]$ i que s'ha de passar al rang $[0, 1] \times [0, 1]$. Ara disposem de les coordenades de textura a cada vèrtex i hem de dir-li a OpenGL quin mapa de bits ha d'utilitzar així com els diferents paràmetres que caracteritzen la textura. Podeu aprofitar i ficar aquesta textura dins d'una llista de visualització de la mateixa manera que es va fer amb els elements de l'objecte.



Nota: podeu utilitzar qualsevol imatge vostra sempre que sigui de 24 bits i amb mida potència de 2. També podeu editar petits objectes amb coordenades de textura per veure com es realitza la interpolació .

Desar objecte

Desar la nova informació generada en aquest lliurament en un nou fitxer de text.

Lliurament 3.- Transformacions. Manipulació.

Translació, rotació i escalat

Aplicarem transformacions geomètriques als objectes de la següent manera. L'objecte llegit apareix en les coordenades definides al fitxer, però un cop llegit les transformacions s'apliquen respecte del centre de la caixa mínima i s'acumulen. Així podríem fer la següent seqüència:

- carregar un cub a l'origen,
- traslladar el cub una unitat en la direcció x ,
- escalar el cub per un factor 2,
- traslladar el cub una unitat en la direcció x ,
- escalar el cub per un factor 1/2,

I veuríem com el cub avança mentre que s'expandeix i s'encongeix.

Aquesta part es pot realitzar de moltes maneres. Convé entendre bé els mecanismes, les funcions necessàries i l'ordre en el que s'apliquen les transformacions. Repassar el capítol 3 del Red Book (<http://www.glprogramming.com/red/chapter03.html>) per veure les funcions de que disposem, com funcionen i alguns exemples. És molt important que per aquesta part la utilització del `glPush()` i `glPop()` sigui correcta, així com guardar la matriu o matrius necessàries i recuperar-les per cada objecte que permetin concatenar transformacions, canviar d'objecte o canviar l'escena i que tot es faci de manera coherent.

Manipulació amb el moviment del ratolí

Moure l'escena, llums o càmera a partir del moviment del ratolí. És a dir, el moviment s'inicia quan l'usuari prem el botó dret de la rata i continua fins que no es deixa anar. Els moviments han de ser proporcionals i acordes al desplaçament de la rata. Per exemple, si movem la rata amb el botó dret polsat i ens desplaçem del límit esquerre al dret de la finestra de l'aplicació el que aconseguim és una rotació completa (360°) de la càmera.

Manipulació d'objectes per separat

En aquest cas, el que es demana és que els diferents objectes que tenim carregats a l'escena es manipulin per separat. És a dir, a partir d'una preselecció, el moviment del ratolí afecti només a un dels objectes sense alterar la posició dels altres ni de l'escena.

Control:

En aquest lliurament assignarem a les següents entrades de teclat les funcionalitats corresponents

“1”...“9”: lectura d'un fitxer “.3d” que associarem als objectes 1...9 i selecció d'objecte d'interès.

“0”: selecció de la llum com a objecte d'interès.

“x”: eliminació de l'objecte.

“p”: projecció ortogràfica/ projecció perspectiva

“q”: primitiva de dibuix punt (activada/desactivada)

“w”: primitiva de dibuix línia (activada/desactivada)
“e”: primitiva de dibuix polígon (activada/desactivada)
“r”: representació de normals (activada/desactivada)
“t”: representació de la caixa mínima (activada/desactivada)
“y”: representació dels eixos (activada/desactivada)

“u”: il·luminació en l'escena (activada FLAT, activada SMOOTH, desactivada)

Transformacions:

Ratolí:

Botó esquerre: rotació de l'objecte d'interès. En el cas de la llum la rotació es farà respecte de l'origen. En el cas dels objectes la rotació es farà respecte del centre de la caixa mínima.

Botó dret: rotació de la càmera. Sempre apuntant cap a l'origen.

Teclat:

Cursors: translació en el pla *XZ*.

Cursors+Alt: translació en altura.

“+”: escalat d'un factor 2

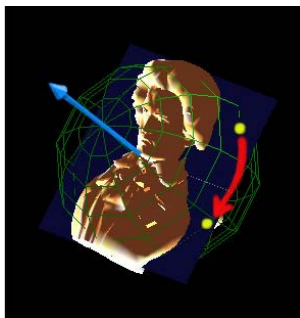
“-”: escalat d'un factor 1/2

MILLORES:

Trackball virtual

Les rotacions que fèiem abans estan bé com a primera aproximació, però si observeu detingudament els moviments veureu per exemple que l'eix *y* no pot sortir mai del pla definit per *YX*, també veureu que si gireu l'objecte en *y* 180° les rotacions en *x* canvien de sentit.

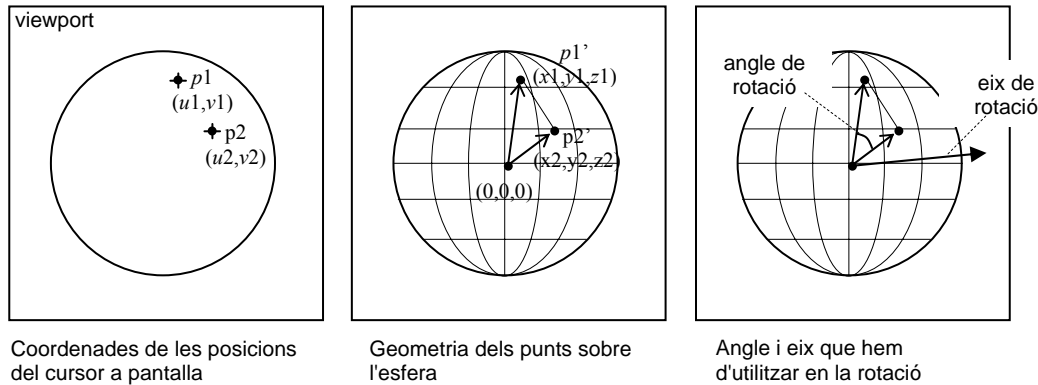
Per reparar aquestes limitacions anem a implementar un moviment més sofisticat que intenti emular el que faria un *trackball*. Per fer això anem a imaginar que el nostre objecte està englobat per una esfera i que les successives posicions del cursor estan sobre aquesta esfera. El moviment del cursor sobre l'esfera generarà una rotació en la direcció perpendicular al pla que conté els punts de la trajectòria i el centre de l'esfera.



Per exemple, a la figura si partim d'un punt i acabem en l'altre segons la trajectòria vermella, l'eix de rotació en aquest instant de temps serà l'eix blau.

Fins aquí hem descrit com funciona el *trackball*, Ara anem a veure com l'implementem. Per fixar les idees suposem una esfera de radi 1 que centrarem a l'origen (0,0,0) i dues posicions

del cursor: $p1 = (u1,v1)$, $p2 = (u2,v2)$. Els punts u,v estan en coordenades del viewport i hem d'escalar-los de manera adient per obtenir les coordenades x,y (recordeu que podem conèixer la mida del viewport com ja van fer en el cas anterior). Sabem que aquests punts x,y estan a sobre d'una esfera i coneixem la seva equació ($x^2+y^2+z^2=r^2$), amb això trobem la z . Hem passat de $(u1,v1)$, $(u2,v2)$ a $p1' = (x1,y1,z1)$, $p2' = (x2,y2,z2)$. Cadascun d'aquests punts defineixen amb l'origen un vector. D'aquests dos vectors traurem l'angle de rotació (producte escalar) i el vector perpendicular al pla que defineix l'eix de rotació (producte vectorial).



La funció de rotació que té OpenGL i que ja heu usat rep com a paràmetres l'angle i l'eix de rotació. Atès que en aquest cas els valors dels angles no s'incrementen com en el cas anterior haureu d'ajudar-vos d'altres funcions que treballin amb matrius (carregar identitat, posar i treure de la pila de transformacions).

S'ha de tenir cura de no passar un eix de rotació nul, que donaria lloc a perdre l'objecte de vista. També heu de vigilar les solucions de z dels punts que no estan dins l'esfera, en aquest cas el millor és projectar el punt escollit a sobre de l'esfera.

Selecció amb cursor

fer servir el ratolí per a la selecció de l'objecte que volem moure. Podeu consultar el Capítol 13 del Red book per a més informació: <http://www.glprogramming.com/red/chapter13.html>

Lliurament 4.- Millores

Millores lliures a partir del resultat anterior

Algunes idees per desenvolupar com a millora són:

- Detectar col·lisions.
- Deformacions de les normals.
- Deformacions de la posició dels vèrtexs.
- Objectes lligat, jeràrquics (les transformacions d'un afecten als altres)

Bibliografia:

- The OpenGL Programming Guide - The Redbook : http://www.opengl.org/documentation/red_book/
- NeHe Productions. <http://nehe.gamedev.net/>

Apèndix A: Inici amb OpenGL

A OpenGL el que hem de fer és especificar el sistema de coordenades que farem servir per representar els dibuixos. Tanmateix hem d'especificar el mecanisme mitjançant el qual les coordenades específiques dels vèrtex dels objectes (3 dimensions) es transformen en coordenades de pantalla (projecció geomètrica o ortogràfica).

Així doncs, per definir l'escena d'OpenGL farem les següents accions:

- Definir una finestra (“window”) que serà on representarem els dibuixos. Aquesta finestra la crearem mitjançant un sistema gestor de finestres com pot ser Windows o la llibreria glut. Les dimensions d'aquesta finestra s'especifiquen en píxels.
- Especifiquem la finestra de visualització o vista (“viewport”). Aquesta finestra és la regió dins de la finestra anterior que farem servir per dibuixar. Per defecte aquesta finestra ocupa tota l'àrea disponible. Tot i així podem especificar més d'una àrea de treball.
- Especificar el tipus de projecció: Tal i com hem dit, és necessari especificar el tipus de projecció que fem servir. La projecció especifica el volum de treball o espai de treball (espai 3D que volem representar a la finestra de visualització) i també com han de transformar-se les coordenades. Per exemple, si fem servir la projecció ortogràfica no tindrem la sensació de profunditat.

Podem traduir aquestes fases a codi. Per facilitar la tasca farem servir la biblioteca *glut*. Tot i així, començarem per crear un projecte buit de Visual C++.

Dins de les opcions de nou projecte, triarem la opció de Win32 *console application*. I dins de les possibles aplicacions, la de consola simple.

El que es demana al primer punt del lliurament és visualitzar una finestra i l'entorn de OpenGL tot tenint present el posicionament de la càmera. Una bona manera de començar és definint la posició de la finestra, la mida, i la inicialització bàsica (CodiBase penjat a la web).

```
#include "glut.h"

void inicialitzar(void); // Inclou aquelles operacions que només s'executen un cop.
void representarEscena(); // Aquesta funció és la que s'encarrega de visualitzar les
// dades per pantalla cada cop el que el Windows ho demana.
void processaTecles(unsigned char tecla, int x, int y); // Processa les tecles i activa
// les accions pertinents.
void finalitzar(); // Alliberament de memòria

int main(int argc, char **argv)
{
    glutInit(&argc,argv); // Inicialitza GLUT
    glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE); // Mode del color i n° de buffers
    glutInitWindowPosition(100,100); // Cantonada superior esquerra de finestra
    glutInitWindowSize(400,400); // Mida de finestra (píxels): amplada i alçada
    glutCreateWindow("Pràctiques de Gràfics en OpenGL"); //Crea la finestra
    inicialitzar(); // funció que inicialitza variables d'estat
    glutDisplayFunc(representarEscena); // Crida a funció que s'usa per dibuixar
    glutKeyboardFunc(processaTecles); // Crida a funció que s'usa per llegir tecles
    glutMainLoop(); // Bucle de processament d'events de l'aplicació
    return 0;
}

void inicialitzar(void)
{
    glClearColor(0.0, 0.0, 1.0, 1.0); // Establim el color de l'esborrat (blau)
    glMatrixMode(GL_PROJECTION); // Volem treballar amb la matriu de projecció
    glLoadIdentity(); // Reinicialitza el sistema de coordenades
    glOrtho(0.0,400.0,0.0,400.0,-1.0,1.0); // Especifica el volum de treball(que veiem)
    atexit(finalitzar); // Funció que es crida en acabar
}
```

```

void finalitzar()
{
    // Es crida just abans de sortir, després de l'exit(0), i és on alliberarem la memòria
}

void representarEscena()
{
    glClear(GL_COLOR_BUFFER_BIT); // Neteja el contingut de la finestra
    glColor3f(1.0f,0.0,0.0); // Estableix el color del dibuix
    glRectf(175.0f,225.0,225.0f,175.0); // Dibuixa un rectangle amb el color establert.
    glFlush(); // Provoca l'execució de les ordres d'OpenGL que
                // encara no s'han executat.

    glutSwapBuffers();
    glutPostRedisplay();
}

void processaTecles(unsigned char tecla, int x, int y)
{
    if (tecla == 27) // Si l'usuari pulsa la tecla ESC sortim
        exit(0);
}

```

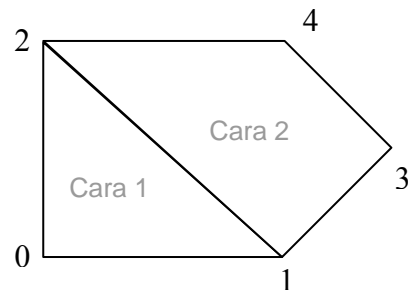
Apèndix B: Format de fitxers .3d

Els fitxers que contenen els objectes i que heu de llegir tenen l'extensió .3d. Tenen la següent estructura:

- Línia de text indicant el **tipus de dades** que ve a continuació. V[N][T][C][N], on els claudàtors indiquen que pot aparèixer aquest valor. Les lletres es corresponen amb la inicial del que trobarem al fitxer: **V**èrtexs, **N**ormals als vèrtexs, coordenades de **T**extura, **C**ares, **N**ormals a les cares. Les úniques combinacions lògiques possibles són: V, VC, VNTCN, VNC, VTC, VNTC, VNCN, VTCN. i d'aquestes les tres primeres seran les més comunes.
- **Nombre de vèrtexs** de l'objecte. Aquest valor l'hem de guardar ja que ens servirà per moltes coses, entre d'altres: ens servirà per reservar la memòria necessària pels vèrtexs (normals als vèrtexs i coordenades de textura), i per controlar els bucles de lectura.
- **Llista de vèrtexs** (pot tenir també informació de normals als vèrtexs i textura)
- **Nombre de cares** de l'objecte. Al igual que en el cas dels vèrtexs a més de guardar aquest valor en l'objecte ens servirà per reservar la memòria necessària per les cares.
- **Llista de cares** indicant a cada cara el nombre de vèrtexs que la componen i la llista d'índexs dels seus vèrtexs (pot contenir també informació de normal a la cara). Important: també hem de reservar memòria per guardar la informació de cada cara.

Exemple amb informació completa (2 polígons amb vèrtexs, normals als vèrtexs, coordenades de textura, cares i normals a les cares).

```
VNTCN
5
0.0 0.0 0.0 0.0 0.0 1.0 0.0 0.0
1.0 0.0 0.0 0.0 0.0 1.0 1.0 0.0
0.0 1.0 0.0 0.0 0.0 1.0 0.0 1.0
1.5 0.5 0.0 0.0 0.0 1.0 1.0 0.0
1.0 1.0 0.0 0.0 0.0 1.0 1.0 1.0
2
3
1 2 0 0.0 0.0 1.0
4
2 1 3 4 0.0 0.0 1.0
```



Explicació dels diferents elements:

