

Visualització Gràfica Interactiva

Pràctica 3

Paisatge Fractal i Il·luminació

1. OBJECTIUS

a) Modelatge de superfícies fractals.

- Estructura de dades matricial i funcions.
- Lectura dels vèrtexs des d'un fitxer.
- Triangulació de la superfície.

b) Generació i visualització del paisatge fractal.

- Fractals Brownians. Mètode de desplaçament de punt mig aleatori.
- Soroll dependent de la distància als pics.
- Visualització de diferents resolucions del fractal. Gestió del Toolbar.

c) Il·luminació del fractal.

- Il·luminació per filferros, cares planes i Gouraud. Càlcul del vector normal a un triangle.
- Assignació de colors segons l'alçada (opció il·luminació suau)
- Efectes d'il·luminació. Transparències.
- Fonts de llum direccionals i puntuals.

2. MODELATGE DE SUPERFÍCIES

2.1. Estructura de dades matricial

Farem servir les següents matrius de tipus *double* per a emmagatzemar el nostre terreny fractal:

- Una matriu `zz[513][513]` (512+la posició 0 x 512+la posició 0) de tipus *double* per les alçades.
- Una matriu `normalsC[513][513][3]` de tipus *double*, de les mateixes dimensions que `zz` i 3 components (x,y,z), per a guardar el vector normal de cada triangle o cara.
- Una matriu `normalsV[513][513][3]` de tipus *double* de les mateixes dimensions que `zz` i 3 components (x,y,z), per a guardar el vector normal de cada vèrtex.

Aquestes variables ja estan declarades en el fitxer *fractals.cpp*.

2.2. Funcions

Per a generar i visualitzar el paisatge fractal hem d'afegir a l'aplicació els següents fitxers:

- *fractals.cpp*, *fractals.h*. Ha de contenir les següents funcions:

- `llegir_pts()`. Funció per a la lectura de les alçades inicials i dels pics des d'un fitxer.
- `escriure_pts()`. Funció per a l'escriptura de les alçades del fractal i els pics en un fitxer.
- `itera_fractal()`. Funció per a la generació de les alçades intermitges.
- `soroll()`. Funció per al càlcul d'un soroll aleatori depenent de la distància a diversos pics.
- `fract()`. Funció de dibuix del fractal per triangulació.
- ***normals.cpp, normals.h***. Conté les funcions per al càlcul de normals a triangles:
 - `normal()`. Calcula el vector normal normalitzat ($\|v\|=1$) de dos vectors donats. **Ja està implementada.**
 - `normalvertex()`. Donada una superfície triangulada, calcula el vector normal normalitzat de cada vèrtex. **Ja està implementada.**
 - `normalcara()`. Donada una superfície triangulada, calcula el vector normal de cada triangle o cara. **Heu d'implementar-la.**

El fitxers *fractals.** i *normals.** els trobareu a Caronte. En Visual Studio podeu afegir/crear fitxers al projecte amb el recurs de menú *Projecto* → *Agregar Elementos Existentes* i apareix la finestra MFC per a llegir fitxers o bé col·locant el cursor en “Archivos de código Fuente” de la pestanya “Explorador de archivos”, i us apareixerà una finestra on heu de seleccionar “Agregar → Elemento Existente” i de nou us apareix la finestra MFC per a llegir el fitxer font a inserir. Per a fitxers de capçalera (.h), col·loqueu el cursor en “Archivos de encabezado” de la pestanya “Explorador de archivos”. En Qt seria amb l'opció *Edit* → *Headers* i col·locant el cursor sobre el missatge *Headers*, i pulsar el botó dret del mouse en Qt.

IMPORTANT: Recordeu de copiar els fitxers font que vulgueu inserir (.cpp i .h) en la carpeta *EntornVGI* del vostre entorn, on es troben la resta de fitxers font.

Si el fitxer no existeix, us demanarà si voleu afegir-lo igualment. Si li dieu que sí, us l'afegirà a la llista de fitxers de la pestanya *File View en Visual* (i en la finestra *Edit* en Qt). Quan cliqueu a sobre, l'entorn Visual o Qt us l'obrirà i editarà.

La persiana de menú *Objecte* → *Fractals* ha de ser de tipus *Pop Up* i controlarà la visualització del terreny fractal, tant el soroll com la paleta de colors a utilitzar.

2.2. Lectura dels vèrtexs des d'un fitxer

Les alçades inicials i els pics de la muntanya es troben al fitxers *.mnt (*escena8p.mnt, riu.mnt, cat64.mnt*). Un exemple concret del format d'aquests fitxers és el següent:

```

9 9                // Tamany de la quadrícula en punts (9=0+8)
9.726536           // Alçada núm. 1
16.666667          // Alçada núm. 2
...
-70.098124         // Alçada núm. 80
-120.000000        // Alçada núm. 81
6                 // Nombre de pics significatius de l'escena
128.000000 64.000000 192.000000 1.000000 // cx[0] cy[0] radi[0] hmax[0]
448.000000 128.000000 192.000000 1.000000 // cx[1] cy[1] radi[1] hmax[1]
256.000000 192.000000 128.000000 1.000000 // cx[2] cy[2] radi[2] hmax[2]
0.000000 512.000000 271.529004 -1.000000 // cx[3] cy[3] radi[3] hmax[3]
448.000000 384.000000 181.019336 -1.000000 // cx[4] cy[4] radi[4] hmax[4]
```

```
256.000000 512.000000 128.000000 -1.000000 // cx[5] cy[5] radi[5] hmax[5]
```

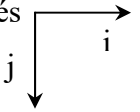

El tamany de la quadrícula (n) pot ser variable en cada fitxer, però acostuma a ser una potència de 2 més 1 punt (la posició 0). El nombre de pics pot ser variable o pot no haver-n'hi cap. En cas de no haver-n'hi, el darrer caràcter del fitxer serà '0' per indicar que no hi ha pics.

Els fitxers contenen $n \times n$ valors que indiquen les alçades (coordenades Z) de la muntanya. Cal que les alçades estiguin equiespaiades. Per exemple, tenint en compte el tamany de la quadrícula (512) i el nombre de punts equiespaiat (suposem 8 per $n=9$) cada parell d'alçades està separat 64 unitats en la matriu **zz**. Això ens dona l'increment inicial o *pas* per a agafar els vèrtexs i visualitzar els triangles de la muntanya fractal (64). Utilitzeu la variable global **pas**, ja declarada a **EntornVGIView.h** (o **CGenvWindow i GLWidget** en Qt), per guardar la resolució del fractal en la classe **CEntornVGIView** (o **CGenvWindow i GLWidget** en Qt).

La lectura de les alçades inicials la implementarem a la funció `llegir_pts()` que ens retornarà el pas en funció del tamany inicial de la quadrícula obtinguda del fitxer. Cal incloure la llibreria `<stdio.h>` per tal de poder obrir i llegir fitxers en el fitxer **fractals.cpp**.

2.3. Triangulació de la superfície

Utilitzarem els índexs (i, j) per accedir a la matriu d'alçades **zz**[i][j] com a coordenades (x, y) al pla. Això vol dir que pintarem punts de coordenades $(i, j, zz[i][j])$. Però, **compte**, no confongueu l'accés a la matriu amb el punt de l'espai que representa:

L'orientació d'una matriu és  mentre que l'orientació del pla és la inversa .

És aquesta darrera orientació la que heu de tenir en compte quan calculem vectors normals.

La forma més simple de representar una superfície és a partir de facetes o polígons plans. Com que tres punts sempre formen un triangle, la manera més general de visualitzar una superfície arbitrària és a partir d'una triangulació. La nostra estructura de dades basada en una quadrícula motiva la triangulació de la *figura 1*. Les fletxes indiquen l'orientació de cadascun dels polígons.

El codi de la triangulació d'una superfície donada com a una matriu d'alçades ha d'estar escrit en la funció *fract* dins el fitxer **fractals.cpp**. La funció *fract* la cridarem des de les funcions `dibuixaEscenaGL()` del mòdul **escena.cpp** tant en **MFC** com en **Qt** quan l'objecte sigui `O_FRACTAL`.

Tots els vèrtexs dels triangles han de ser recorreguts en el sentit invers de les agulles del rellotge, perquè així quan calculem els seus vectors normals tots apuntaran cap amunt (eix Z positiu). Això vol dir que a l'hora de donar les coordenades dels vèrtexs a pintar ho farem en l'ordre establert en la *figura 1*. S'ha de generalitzar per a les quadrícules de diferent tamany, doncs aquest ens dona una resolució fractal. El tamany de la quadrícula ve controlat per la variable **pas**. **Centreu el pla x-y de la muntanya en el (0,0) i visualitzeu les alçades inicials en filferros.**

Per a dibuixar els dos triangles de la *figura 1* cal fer:

```
glBegin(GL_TRIANGLES);
    glVertex3f(i,j,zz[i][j]);           // V1
    glVertex3f(i+pas,j,zz[i+pas][j]);   // V2
    glVertex3f(i+pas,j+pas,zz[i+pas][j+pas]); // V3
glEnd();
```

```

glBegin(GL_TRIANGLES);
    glVertex3f(i,j,zz[i][j]);           // V1
    glVertex3f(i+pas,j+pas,zz[i+pas][j+pas]); // V3
    glVertex3f(i,j+pas,zz[i][j+pas]);   // V4
glEnd();

```

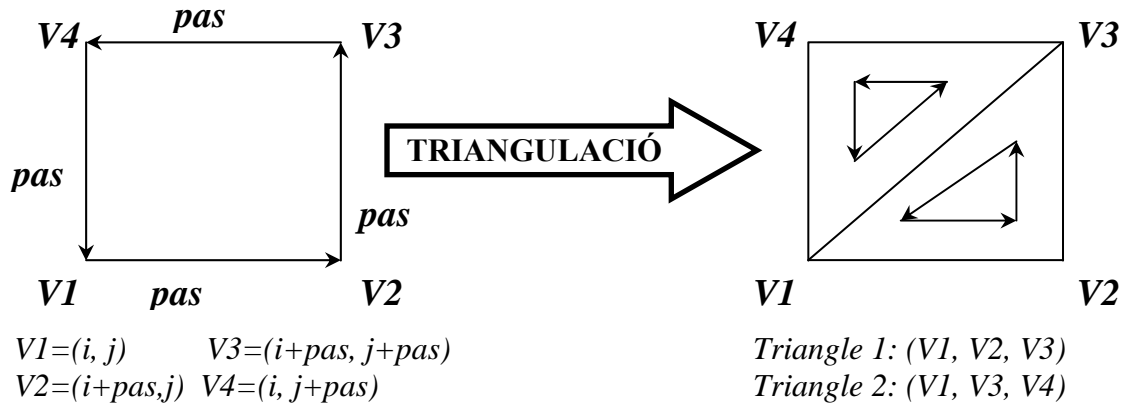


Figura 1. Triangulació d'una quadrícula.

3. GENERACIÓ I VISUALITZACIÓ DEL PAISATGE FRACTAL

3.1. Fractals Brownians

Una superfície fractal de tipus brownià consisteix a donar unes alçades aleatòries sobre una quadrícula del pla x-y. Seguirem el mètode *del punt mig aleatori* (tema modelatge 3D dels apunts de teoria) per a la generació de les alçades del fractal.

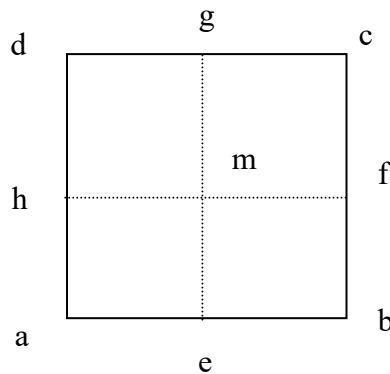


Figura 2. Subdivisió d'una faceta segons l'algorisme del punt mig aleatori.

Donat un rectangle de vèrtexs a, b, c, d , (figura 2) i alçades inicials (z_a, z_b, z_c, z_d), les alçades (z_e, z_f, z_g, z_h, z_m) dels punts mitjos de cada aresta e, f, g, h, m s'assignen de la següent manera:

- Les alçades dels punts e, f, g, h es calculen fent els promitjos de les alçades dels vèrtexs més propers i sumant-hi un soroll. El model és:

$$z_e = (z_a + z_b) / 2 + \text{soroll}(ie, je)$$

- Pel que fa a l'alçada z_m , la podem calcular com:

$$z_m = (z_a + z_b + z_c + z_d) / 4 + \text{soroll}(im, jm)$$

La variable *soroll* és una funció d'un valor aleatori proporcional a la longitud de l'aresta. El fractal es genera partint de les $n \times n$ alçades inicials llegides en el fitxer que s'assignen de forma equiespaiada en la matriu **zz**. Després, apliquem de forma recursiva el procés de subdivisió i calculem les alçades intermitges tenint en compte el soroll fins a cobrir tota la matriu d'alçades (resolució màxima, **pas=1**).

Nosaltres usarem una distribució uniforme (funció **rand()**) per a generar el soroll. Així doncs, fixada una certa resolució (o *pas*), el soroll d'un punt genèric (*i, j*) serà:

$$soroll(i,j) = f(i,j) * (valor_aleatori) * pas$$

on $f(i,j)$ és una funció que depèn de la posició del punt (*i,j*) i *valor_aleatori* és un valor generat de forma aleatòria entre 0 i 1.

Els nombres aleatoris els calcularem amb la funció **rand()**. En primer lloc necessitem generar una llavor utilitzant el rellotge intern:

```
// Llavor dels nombres a generar. Funció cridada un sol cop
srand( (unsigned) time(NULL));
```

La llavor s'ha de cridar una sola vegada en el programa. Cada cop que necessitem un valor aleatori, cridarem a la funció **rand()** amb la sintaxi:

```
// Cridar cada cop que necessitem generar un valor aleatori
valor_aleatori= (double) rand()/RAND_MAX;
```

Per utilitzar les funcions de generació de nombres aleatoris, cal incloure les llibreries `<stdlib.h>` i `<time.h>` en el fitxer on es criden. La variable **RAND_MAX** és una constant interna i no cal tocar-la.

3.2. Soroll dependent de la distància a uns pics.

Per a generar un paisatge fractal com el de la figura 6 farem que la funció $f(i,j)$ depengui de les distàncies, $d[k]$, a diversos **pics** amb centres $cx[k], cy[k]$, $k=0,...,Kmax$. Concretament farem que $f(i,j)$ sigui:

$$f(i,j) = g(d[0]) + \dots + g(d[Kmax])$$

La funció de soroll $g(d[k])$ depèn de la distància $d[k]$ del pic k -èssim al punt (*i,j*) que estem avaluant, de forma que la funció g prendrà el seu valor **màxim** ($hmax[k]=1$, si el pic es d'alçada, és a dir, muntanya) o **mínim** ($hmax[k]=-1$, si el pic és de profunditat, és a dir, fons del mar) en el cas que $d[k]=0$ (és a dir, el punt (*i,j*) és en el centre del pic $cx[k], cy[k]$), i prendrà el valor **zero** quan el punt (*i,j*) estigui fora del radi d'influència $r[k]$ respecte del centre del pic.

En la pràctica implementarem un màxim de 6 pics, és a dir $Kmax=5$.

Definim les següents funcions de soroll g donat un pic de radi d'influència r , alçada màxima h , distància d entre el punt (*i,j*) i el centre del pic (cx,cy):

FUNCIÓ LINIAL :

$$g(d)= \begin{cases} h*[1-d/r], & \text{si } d < r \\ 0 & , \text{ si } d > r \end{cases}$$

FUNCIÓ QUADRÀTICA:

$$g(d)= \begin{cases} h*[1-d*d/(r*r)], & \text{si } d < r \\ 0 & , \text{ si } d > r \end{cases}$$

FUNCIÓ SQRT:

$$g(d)= \begin{cases} h*[1-\text{sqrt}(d)/\text{sqrt}(r)], & \text{si } d < r \\ 0 & , \text{ si } d > r \end{cases}$$

FUNCIÓ DIFERENCIABLE:

$$g(d)= \begin{cases} h*(1-d/r) * (1-d/r)], & \text{si } d < r \\ 0 & , \text{ si } d > r \end{cases}$$

Incorporeu quatre recursos de menú (Linial, Quadràtica, Sqrt, Diferenciable) de tipus *Checked* dins la persiana *Objecte → Fractal* per a generar fractals amb cadascuna de les funcions anteriors. Per a cada un dels quatre recursos, el valor de la variable de control *objecte* serà el mateix (O_FRACTAL) però el valor de la variable de control *soroll* serà diferent: S_LINIAL, S_QUADRATIC, S_SQRT, S_DIFERENCIABLE. Aquests valors ja estan predefinits al fitxer *fractals.h*.

Cada fitxer d'alçades inicials *.*mnt* conté els centres, radis i alçades màximes dels pics de soroll. Per exemple, els centres, radis i hmax de cada pic definits en el fitxer *escena8p.mnt* són els següents:

```
(cx[0], cy[0])=(128,64);      r[0]=128+64;      hmax[0]=1;
(cx[1], cy[1])=(512-64,128);  r[1]=128+64;      hmax[1]=1;
(cx[2], cy[2])=(256,256-64);  r[2]=128;      hmax[2]=1;
(cx[3], cy[3])=(0,512);      r[3]=sqrt(2)*(128+64); hmax[3]=-1;
(cx[4], cy[4])=(512-64,256+128); r[4]=sqrt(2)*128; hmax[4]=-1;
(cx[5], cy[5])=(256,512);      r[5]=128;      hmax[5]=-1;
```

Per a llegir el fitxer fractal *escena8p.mnt* farem servir el recurs de menú *Arxius → Obrir Fractal...* que tenim definit a *CEntornVGIView* en MFC (o *GLWidget* en Qt) (funció *OnFileOpen()*). En aquesta funció es crida a un diàleg per seleccionar el fitxer i a la sortida d'aquest diàleg teniu el nom del fitxer a la variable de tipus *CString* *nom*. En aquest punt cal cridar la funció del fitxer *fractals.cpp* que llegeixi el fitxer d'alçades i pics del fractal (*llegir_pts()*) i la que inicialitzi la resta d'alçades del fractal (*itera_fractal()*).

OPCIONAL: Escriure el fractal en un fitxer. També teniu implementat el recurs *Arxius → Guardar Fractal com...* que tenim definit a *CEntornVGIView* en MFC (o *GLWidget* en Qt) (funció *OnFileSaveAs()*) que ha de guardar les alçades del fractal segons la resolució de visualització (variable *pas*) que tingui, així com els pics, en un fitxer *.*mnt*. Igual que en l'anterior, està fet el diàleg que permet definir el nom de fitxer d'escriptura. Després del diàleg, el nom del fitxer es troba a la variable *nom*.

3.3. Visualització a diferents resolucions fractals. Gestió del Toolbar (MFC)

Per a fixar la resolució fractal que volem visualitzar, crearem dos botons al toolbar: *It+*, per augmentar-la i *It-*, per disminuir-la (figura 3). Com de costum, associarem aquests recursos a la classe *CEntornVGIView*.

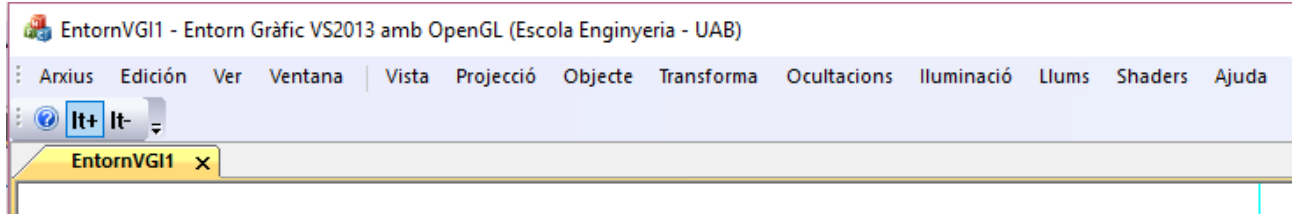


Figura 3. Toolbar pel control de la resolució fractal.

La variable pas de la classe *CEntornVGIView* serà l'encarregada de controlar la resolució fractal.

El muntatge i gestió d'un botó del Toolbar, és semblant a la d'una opció de menú. Per a dissenyar un botó heu d'anar a l'opció *Toolbar* (que es troba dins la pestanya *Vista de Recursos*) i fer un doble click a **IDR_MAINFRAME_256** (botonera per a Windows 7. **IDR_MAINFRAME** és la antiga botonera per a Windows XP). Apareixerà l'editor del *ToolBar* de l'aplicació (figura 4).

Feu un click sobre l'eina per escriure text (A) i escriviu el text que voleu que contingui el botó. Per associar-li un identificador (ID_XXX) al botó, cal fer un doble click sobre el botó o seleccionant el botó polseu tecla dreta del mouse i apareixerà un desplegable on trieu *Propiedades*.

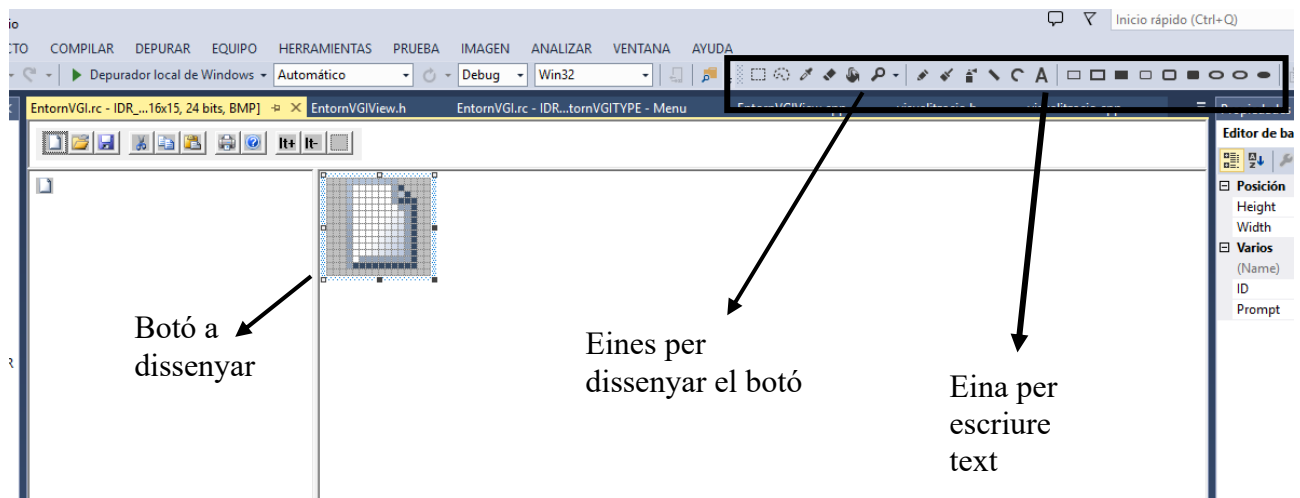


Figura 4. Editor del Toolbar.

Tot botó amb identificador (ID_XXX) del *Toolbar* sol portar associat el missatge **COMMAND**, al que li podeu associar una funció de tractament del botó **OnBotó()**. Per a definir aquesta funció, seleccioneu **Proyecto** → **Asistente para clases** on us apareixerà la finestra de la figura 5.

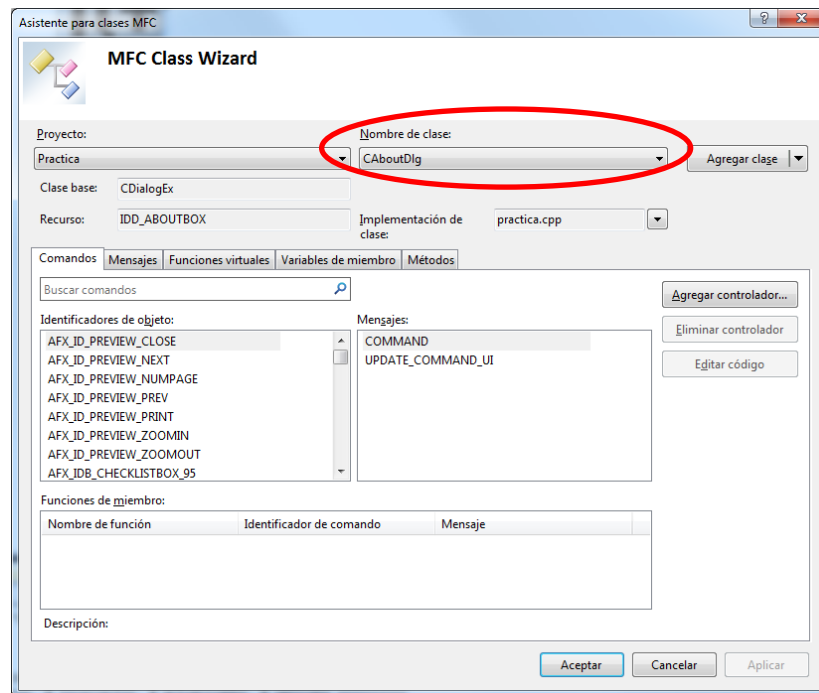


Figura 5. Assistent per a classes.

Per a inserir la funció de tractament a un identificador (**ID_XXX**), sigui un botó o un menú, a la classe **CEntornVGIView**, en primer lloc trieu com a nom de classe **CEntornVGIView**. Després en la finestra de l'esquerra trieu l'identificador d'objecte (botó, en la figura 6 **ID_ITERAMES**) i el missatge que voleu activar (**COMMAND**, que definirà la funció **Oniterames()**). Seleccioneu el botó "Agregar controlador", us demanarà confirmació de la funció que esteu definint i la classe on la poseu i si accepteu posa l'esquelet de la funció definida en **EntornVGIView.cpp**.

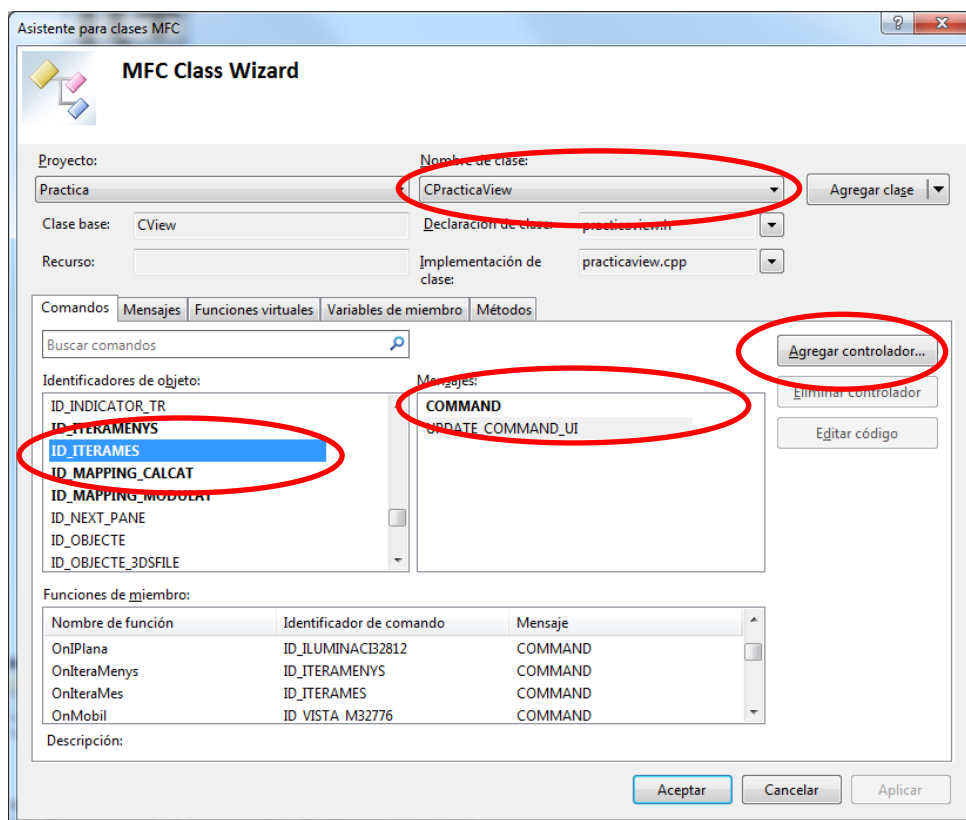


Figura 6. Assistent per a classes per a definir la funció de tractament del botó It+ (**ID_ITERAMES**).

3.4. Visualització de diferents resolucions fractals. Gestió del Toolbar (Qt)

Per a fixar la resolució fractal que volem visualitzar, crearem dos botons al toolbar: *It+*, per augmentar-la i *It-*, per disminuir-la (figura 7). Com de costum, associarem aquests recursos a la classe *GLWidget*.

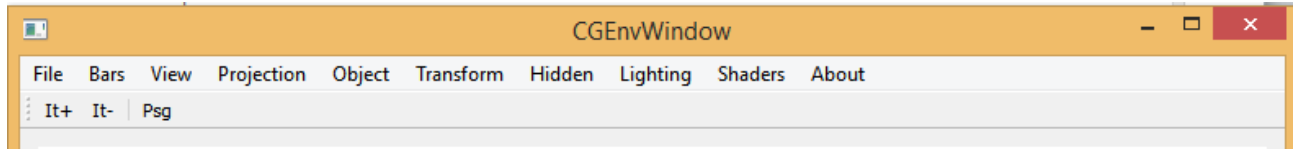


Figura 7. Toolbar pel control de la resolució fractal.

La variable **pas** de la classe *GLWidget* serà l'encarregada de controlar la resolució fractal.

El muntatge i gestió d'un botó del Toolbar, és semblant a la d'una opció de menú. En primer lloc, definim les funcions *OnIteraMes()* i *OnIteraMenys()* com a slots en la classe *CGEnvWindow* (figura 8) (cliclar botó dret del mouse sobre la finestra controlada per *CGEnvWindow* i de la finestra que apareix triar *Change signals/slots*).

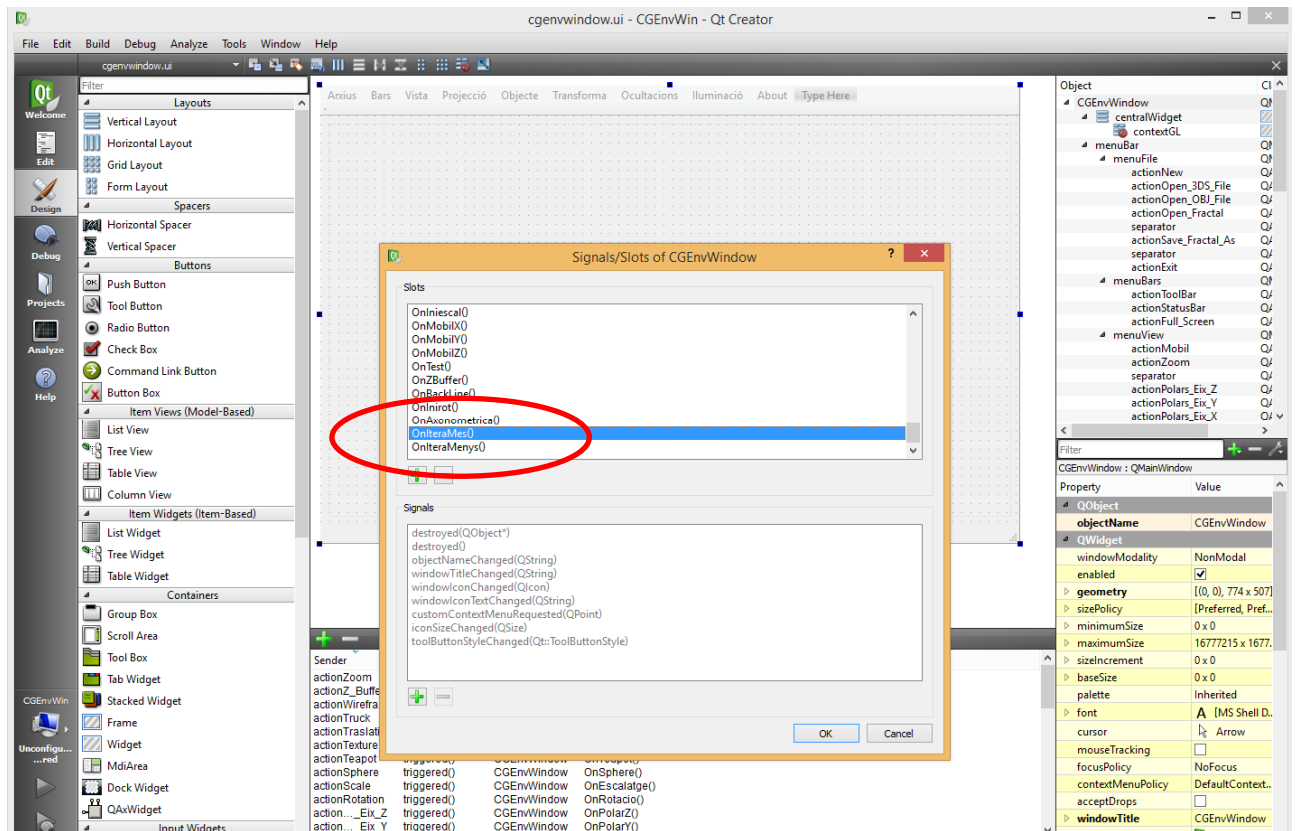


Figura 8. Definició de les funcions *OnIteraMes*, *OnIteraMenys* en la classe *CGEnvWindow*.

Posteriorment, afegir dues opcions de menú (*It+*, *It-*) a qualsevol desplegable. En la figura 9 s'utilitza el desplegable *Projecció*.

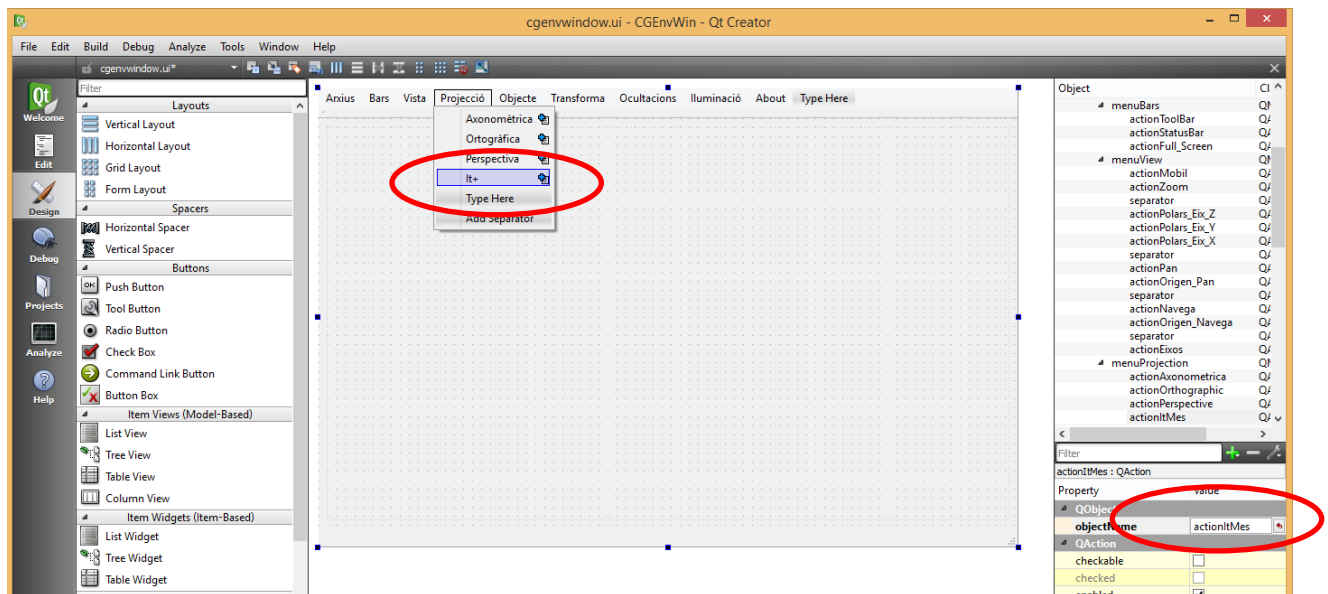


Figura 9. Definició de l'opció It+ amb el nom actionItMes.

Posteriorment s'assigna l'acció *actionItMes* amb la funció *OnIteraMes()* definida a la classe *CGEnvWindow* (figura 10).

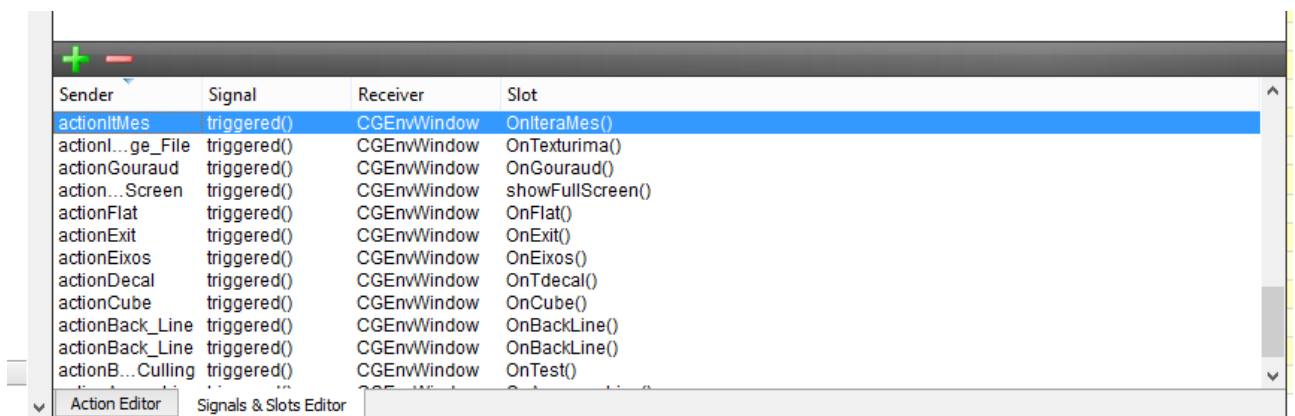


Figura 10. Assignació de l'acció *actionItMes* al slot *OnIteraMes()*.

Fet això, desplacem l'opció demenú a la zona del *ToolBar*, entre la barra de menús i l'àrea de dibuix (figura 11). Seleccionant l'opció It+ amb el botó esquerra del mouse el desplacem a la zona del Toobar.

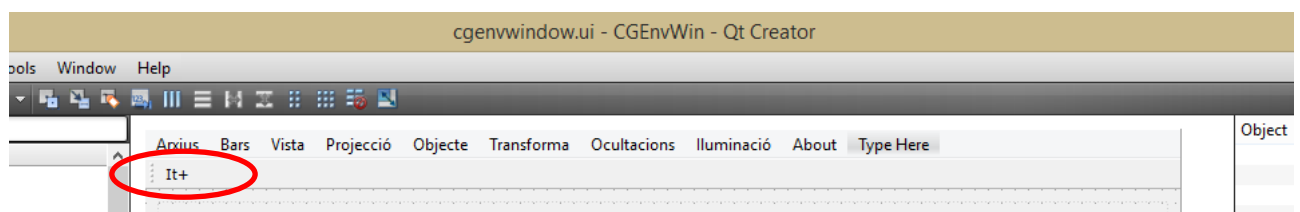


Figura 11. Inserció opció It+ en Toolbar.

Fet això, ja només cal definir les funcions *OnIteraMes()* i *OnIteraMenys()* de la classe *CGEnvWindow* per a que actualitzin la variable pas de *CGEnvWindow* que mostra el valor de la

variable pas al *StatusBar*, actualitzar la variable pas de la classe **GLWidget** i cridar a les funcions `OnIteraMes()` i `OnIteraMenys()` de **GLWidget** per a que es revisualitzi l'escena.

La funció `OnIteraMenys()` en la classe **CGLWindow** seria:

```
// Itera menys (fractal)
void CGLWindow::OnIteraMenys()
{

// Crida a la funcio 'On*' dins contextGL per a gestionar l'opcio
    ui->contextGL->OnIteramenys();

// Refrescar Status Bar
    Barra_Estat();
}
```

La funció `OnIteraMenys()` en la classe **GLWidget** seria:

```
// Itera mes (fractal)
void GLWidget::OnIteramenys()
{
// Modificar el valor de pas si l'objecte és O_FRACTAL.

// Crida a PaintGL() per redibuixar l'escena
    updateGL();
}
```

4. IL·LUMINACIÓ

En l'aplicació estan definits tres tipus d'il·luminació:

1. Filferros (`ilumina = 'f'`)
2. Il·luminació per cares planes (`ilumina = 'p'`)
3. Il·luminació suau de Gouraud (`ilumina = 's'`)

Per cada tipus d'il·luminació, hem de calcular de forma diferent els vectors normals del fractal que s'estigui visualitzant i guardar-los en les matrius **normalsC** i/o **normalsV**. El tamany dels triangles sobre els que es calcula el vector normal depèn de la resolució del fractal, definida a la variable **pas**.

4.1. Filferros

Per dibuixar el fractal (funció `fract()` en **fractals.cpp**), en aquest tipus d'il·luminació no cal definir vectors normals, pel que per visualitzar els triangles cal fer:

```
glBegin(GL_TRIANGLES);
    glVertex3f(i,j,zz[i][j]);           // V1
    glVertex3f(i+pas,j,zz[i+pas][j]);   // V2
    glVertex3f(i+pas,j+pas,zz[i+pas][j+pas]); // V3
glEnd();

glBegin(GL_TRIANGLES);
    glVertex3f(i,j,zz[i][j]);           // V1
    glVertex3f(i+pas,j+pas,zz[i+pas][j+pas]); // V3
    glVertex3f(i,j+pas,zz[i][j+pas]);   // V4
glEnd();
```

4.2. Cares Planes

Aquest tipus d'il·luminació (que és el més senzill possible) consisteix a assignar la mateixa intensitat a tots els píxels d'una cara. Es calcula, doncs, un únic vector normal per cara i OpenGL calcula la intensitat de tots els punts de la cara a partir d'aquest vector normal. Cal que el vector estigui normalitzat. Per normalitzar un vector $v=(v_x, v_y, v_z)$ hem de dividir totes les seves components pel seu mòdul m .

$$m = \text{sqrt}(v_x^2 + v_y^2 + v_z^2).$$

Per a calcular el vector normal a una cara cal fer el producte vectorial de dos vectors generats pels tres vèrtexs de la cara tal i com mostra la figura 12.

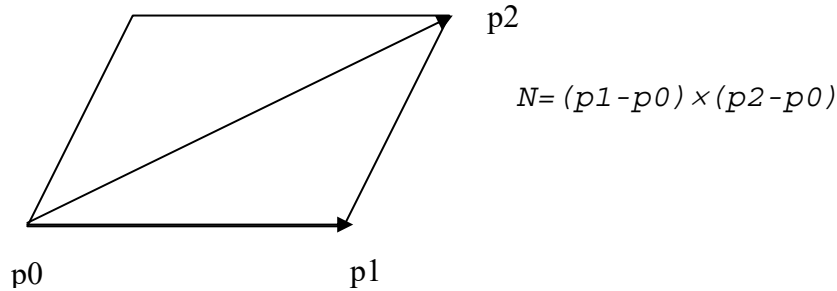


Figura 12. Càlcul del vector normal N .

Recordeu que donats dos vectors $v_1=(x_1,y_1,z_1)$ i $v_2=(x_2,y_2,z_2)$ el seu producte vectorial és el vector que té com a components:

$$v_1 \times v_2 = (y_1 z_2 - y_2 z_1, z_1 x_2 - z_2 x_1, x_1 y_2 - x_2 y_1)$$

El codi del càlcul del producte vectorial normalitzat ja està implementat a la funció *normal* del fitxer **normals.cpp**.

Utilitzarem les matrius **normalsC**, **normalsV**, per emmagatzemar les normals a cadascun dels triangles de la superfície. La posició del vector normal en les matrius **normals*** ha de ser de la següent forma: A **normalsC[i][j]** s'ha de guardar la normal del *Triangle1* (V_1, V_2, V_3) i a **normalsV[i][j]** la normal del *Triangle2* (V_1, V_3, V_4). La funció *normalcara()* dins el fitxer **normals.cpp** és l'encarregada de calcular les normals a les cares de la triangulació que s'hagi de visualitzar.

Per dibuixar el fractal (funció *fract()* en **fractals.cpp**) en aquest tipus d'il·luminació cal definir el vector normal per la cara (**normalsC[i][j][0]**, **normalsC[i][j][1]**, **normalsC[i][j][2]**), abans d'especificar les tres coordenades dels vèrtexs. Així, per dibuixar els triangles del fractal cal fer:

```
glBegin(GL_TRIANGLES);
    glNormal3f(normalsC[i][j][0], normalsC[i][j][1], normalsC[i][j][2]); // VNorm.
    glVertex3f(i,j,zz[i][j]); // V1
    glVertex3f(i+pas,j,zz[i+pas][j]); // V2
    glVertex3f(i+pas,j+pas,zz[i+pas][j+pas]); // V3
glEnd();

glBegin(GL_TRIANGLES);
    glNormal3f(normalsV[i][j][0], normalsV[i][j][1], normalsV[i][j][2]); // VNorm.
    glVertex3f(i,j,zz[i][j]); // V1
    glVertex3f(i+pas,j+pas,zz[i+pas][j+pas]); // V3
    glVertex3f(i,j+pas,zz[i][j+pas]); // V4
glEnd();
```

4.3. Il·luminació suau de Gouraud

L'algorisme d'il·luminació de Gouraud, consisteix a calcular una intensitat a cada vèrtex de la malla i interpolar aquestes intensitats linealment a la resta de punts de la malla. La intensitat de cada vèrtex es calcula a partir del vector promig de les normals unitàries a les cares a les quals pertany l'esmentat vèrtex. El càlcul del vector normal a cada vèrtex ja està implementat a la funció `normalvertex()`. Rep com a paràmetres la matriu d'alçades `zz[i][j]`, la variable que controla la resolució del fractal que visualitzem (`pas`) i la matriu `normals[i][j]` on es guarden els vectors normals a cada vèrtex.

Per a dibuixar el fractal (funció `fract()` en *fractals.cpp*) amb aquest tipus d'il·luminació, per a cada vèrtex de coordenades $(i, j, z[i][j])$ haureu de calcular el seu corresponent vector normal (`normalsV[i][j][0], normalsV[i][j][1], normalsV[i][j][2]`) i definir-la amb la comanda `glNormal3f()` abans de definir el vèrtex:

```
glBegin(GL_TRIANGLES);
    glNormal3f (normalsV[i][j][0], normalsV[i][j][1], normalsV[i][j][2]); // Normal a V1
    glVertex3f(i, j, zz[i][j]); // V1
    glNormal3f(normalsV[i+pas][j][0], normalsV[i+pas][j][1],
               normalsV[i+pas][j][2]); // Normal a V2
    glVertex3f(i+pas, j, zz[i+pas][j]); // V2
    glNormal3f(normalsV[i+pas][j+pas][0], normalsV[i+pas][j+pas][1],
               normalsV[i+pas][j+pas][2]); // Normal a V3
    glVertex3f(i+pas, j+pas, zz[i+pas][j+pas]); // V3
glEnd();

glBegin(GL_TRIANGLES);
    glNormal3f (normalsV[i][j][0], normalsV[i][j][1], normalsV[i][j][2]); // Normal a V1
    glVertex3f(i, j, zz[i][j]); // V1
    glNormal3f((normalsV[i+pas][j+pas][0], normalsV[i+pas][j+pas][1],
               normalsV[i+pas][j+pas][2]); // Normal a V3
    glVertex3f(i+pas, j+pas, zz[i+pas][j+pas]); // V3
    glNormal3f((normalsV[i][j+pas][0], normalsV[i][j+pas][1],
               normalsV[i][j+pas][2]); // Normal a V4
    glVertex3f(i, j+pas, zz[i][j+pas]); // V4
glEnd();
```

4.3.1. Assignació de colors segons l'alçada (opció il·luminació Gouraud)

Per tal de donar un aspecte més realista al fractal, en l'opció d'il·luminació Gouraud assignarem a cada vèrtex del fractal un color diferent segons l'alçada `zz[i][j]`. A l'opció d'il·luminació suau de la funció `Ilumina()` s'ha activat la il·luminació de Gouraud amb la comanda `glShadeModel(GL_SMOOTH)` per tal que OpenGL interpoli les intensitats a tots els punts de la malla.

La manera més simple d'assignar colors és a partir d'una paleta RGB (en el nostre cas de 24 colors) en forma de tres vectors, `med_colorR`, `med_colorG`, `med_colorB`. A cada alçada `zz[i][j]` se li assignarà una índex `k` per accedir a cadascun d'aquests vectors i assignar-li el color al vèrtex $(i, j, z[i][j])$ mitjançant la funció `glColor3f()`.

Heu de definir una funció d'assignació de colors per a calcular l'índex `k` de tal manera que el vèrtex d'alçada mínima del fractal se li assigni el color de la posició més baixa (0) de la paleta i al vèrtex d'alçada màxima el color de la posició més alta (23) de la paleta. Els valors de aquesta paleta (`med_colorR`, `med_colorG`, `med_colorB`) estan definits a *fractals.h*.

En *fractals.h* hem definit tres paletes de color (mediterrani, tundra i glacià). Utilitzeu una de les tres paletes. En l'opció *Objecte* → *Fractals* obriu una opció *Paleta?* per activar o desactivar la paleta (si la paleta està desactivada s'utilitza el color de l'objecte per defecte, usualment). Opcionalment, podeu definir una persiana per poder triar cada una de les tres paletes (veure figura 13).

4.4. Efectes d'il·luminació: Transparències. Dibuix d'un pla de mar

La manera més estàndard de donar un aspecte transparent als objectes és usar l'anomenat *blending*. El *blending* o combinació consisteix a barrejar el color del píxel de l'objecte que anem a dibuixar (*destination pixel*) amb el color del píxel ja existent en aquell moment en el buffer (*source pixel*). El factor que controla les proporcions entre els dos colors és el factor *alfa*, valor entre 0 i 1 que és el quart paràmetre de la comanda OpenGL *glColor4f*. La manera de combinar els dos colors és una funció depenent d'*alfa* anomenada *blending function*. Nosaltres usarem la *GL_ONE_MINUS_SRC_ALPHA*. En aquesta modalitat, el color del píxel destí ve determinat per $\text{alfa} * (\text{color_destí}) + (1 - \text{alfa}) * (\text{color_origen})$. Per tant, quan més petit sigui el paràmetre *alfa*, més gran serà el color destí i, per tant, més transparent semblarà l'objecte que estem dibuixant. Les comandes OpenGL per activar el *blending* són:

```
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

Observeu que si volem que un objecte sigui transparent a tots els existents a l'escena cal que el pintem al final. Useu la tècnica del *blending* per a afegir a l'escena un mar semitransparent que abarqui tota la quadrícula i de color $(r,g,b,a)=(0.2,0.75,0.9,0.5)$ al pla $z=0$. Utilitzeu la comanda *glRectf(x1,y1,x2,y1)*.

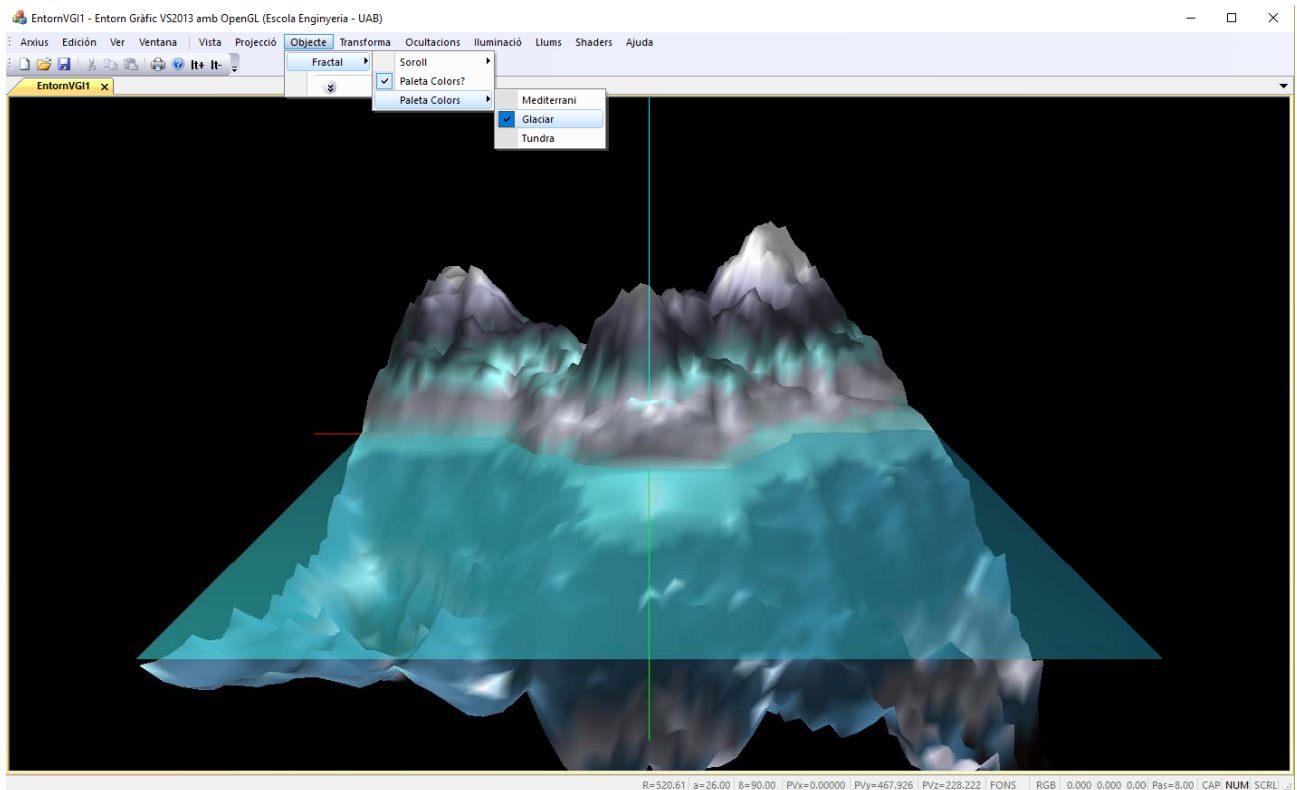


Figura 13. Paisatge fractal amb paleta glacià.

4.5. Fonts de llum direccionals i puntuals (OPCIONAL)

La posició d'una llum ve definida amb `glLightfv(GL_LIGHTi, GL_POSITION, position)`, on *position* és un vector de posició (x,y,z,w) . El valor w s'utilitza per definir fonts direccionals o puntuals:

- $w=0$: La font de llum és una **font direccional** de direcció (x,y,z) .
- $w=1$: Es tracta d'una **font puntual** situada a (x,y,z) .

En l'entorn bàsic, al principi de la rutina *Ilumina* dins el fitxer *visualitzacio.cpp* està definida la font `GL_LIGHT0` com una font direccional a $(0,0,1)$. El recurs de menú *Iluminacio* → *Fixe?* controla si la font de llum és fixa en coordenades món o mòbil respecte del punt de vista.

Anem a canviar la il·luminació per poder visualitzar els fractals.

Definirem un recurs de menú *Iluminació* → *Sol?* (controlat per la variable de control booleana *isol* definida a *CEntornVGIView*) (figura 6) que definirà la font de llum `GL_LIGHT0` com una font de llum direccional o puntual amb els següents valors:

- ***isol=true***: Font de llum direccional amb direcció $(0,0,200)$ (figura 14).
- ***isol=false***: Font de llum puntual situada a $(0,0,200)$ (figura 15).

Així, l'opció *Iluminació* → *Fixe?* sols controlarà la posició fixe o mòbil de la font de llum.

Per fer coherent la il·luminació en el fractal, quan s'activi l'opció *Iluminació* → *Sol?* (*isol=true*) s'ha d'activar també l'opció *Iluminació* → *Fixe?* (*ifixe=true*).

Fixeu-vos que en el cas de font de llum direccional les cares apareixen ben il·luminades, fruit de la incidència paral·lela dels raigs de llum (figura 13). En el cas de font de llum puntual han d'aparèixer zones fosques degut a la diferent incidència de la llum en les cares (figura 14).

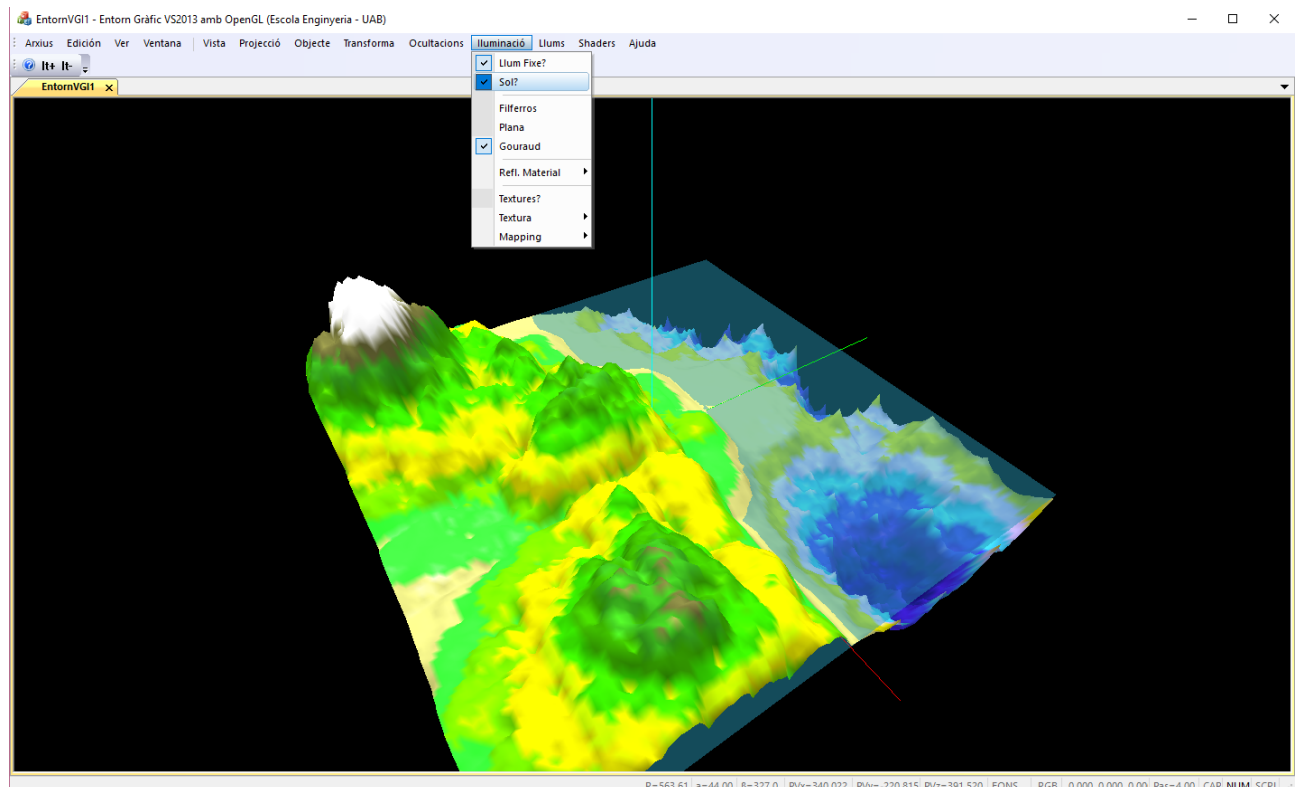


Figura 14. Paisatge fractal amb soroll linial, font fixe i sol activat.

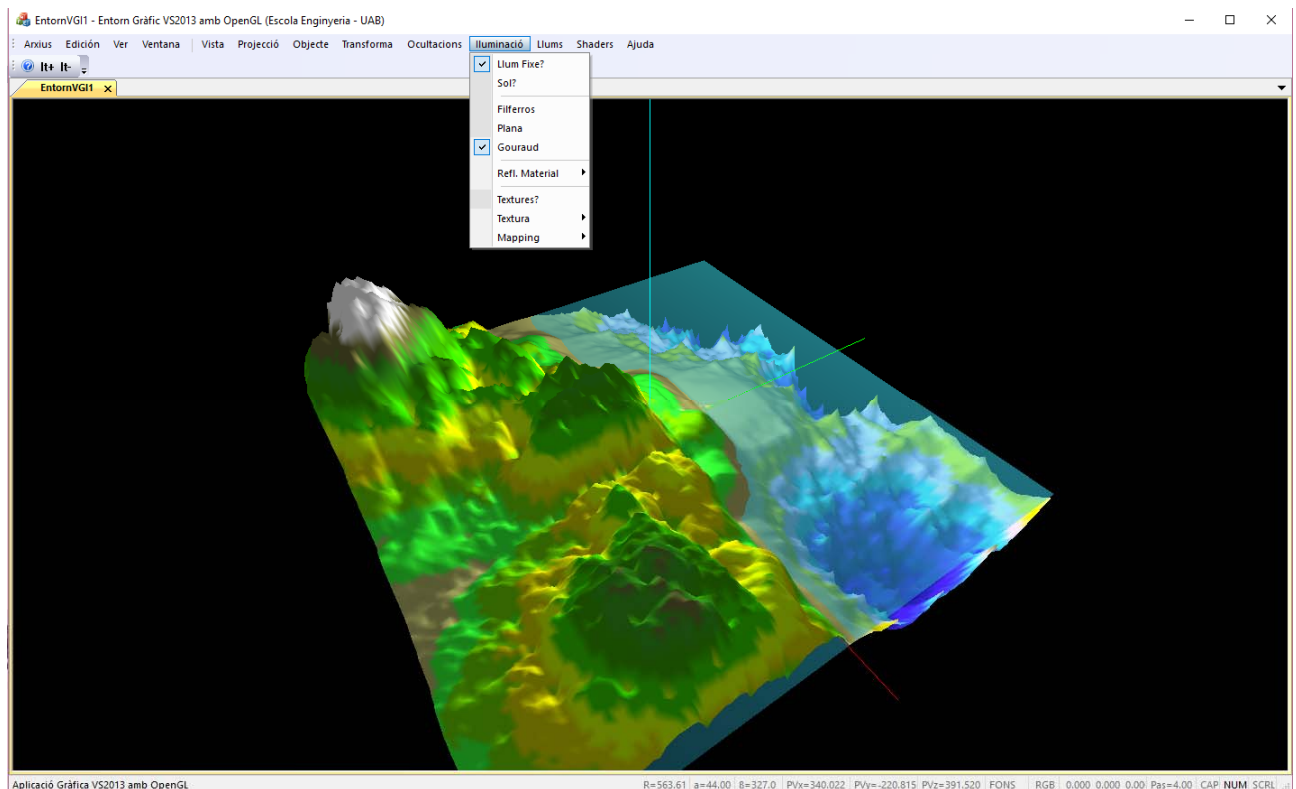
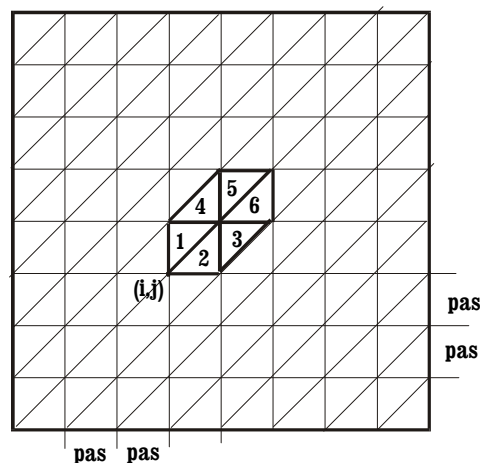


Figura 15. Paisatge fractal amb soroll linial i font fixe i sol desactivat.

5. PREGUNTES

Pregunta 1. Defineix el codi OpenGL de la iteració i la triangulació dels triangles 1,2,3,4,5,6 de la malla hexagonal que es mostra en la figura segons els índex i,j de la malla.



Pregunta 2. Explica amb quantes i quines iteracions (índexs i límits dels índex) has fet per a calcular els vectors normals de totes les cares de la malla. Ajuda't d'un dibuix com el de la pregunta 1 per a descriure quins vèrtexs calcules en cada iteració. Pots calcular tots els vèrtexs en una única iteració?.

Pregunta 3. Com has determinat la resolució inicial del fractal (variable `pas`) a partir del tamany inicial de la quadrícula llegida en el fitxer (n)?.

Pregunta 4. Com has determinat l'índex **k** de la paleta de colors segons l'alçada **zz[i][j]**?

Pregunta 5. Comenta quines diferències trobes en l'aspecte de la muntanya pels diferents sorolls que hem implementat (quin soroll és més abrupte, quin el més suau, etc.).

6. ENTREGA, AVALUACIÓ I DATES LÍMIT

6.1 Avaluació

- 1) En la següent sessió de pràctiques l'alumne haurà de fer un examen escrit de forma individual de preguntes i exercicis relacionats amb la pràctica (**avaluació individual**). Per tant, **cal que tots els grups vinguin a primera hora de la sessió d'examen**. No fer l'examen implicarà un 0 de la part d'avaluació individual.
- 2) En una de les sessions de pràctiques (al final de la que toca o la següent) els alumnes hauran de fer una demo del treball fet al professor (**avaluació grupal**). No fer-la implicarà un 0 en la part d'avaluació grupal.
- 3) Les dates límit d'entrega són úniques i improrrogables.
- 4) Les primeres avaluacions (demo i examen) es valoraran sobre 10. Les avaluacions de recuperació (demo i examen) es valorarà sobre 8.

6.2 Normes d'estil

Us donem un entorn bàsic a principi de curs. A partir de l'entorn, podeu afegir les funcions i paràmetres a les funcions que creieu necessàries, podent modificar el codi de la manera que creieu convenient, sempre que documenteu de forma adient el codi.

Les pràctiques les anireu acumulant sobre l'entorn de pràctiques triat, és a dir que al final de curs heu de tenir un únic projecte amb les 4 pràctiques. **Es recomana, però, que quan acabeu cada pràctica guardeu una còpia de seguretat abans de començar la següent.**

El professor pot demanar-vos el codi font de la pràctica quan ho cregui necessari.

6.3 Dates Límit d'Entrega

DEMO:	23 de Novembre 2017 o 14 Desembre 2017
EXAMEN:	14 de Desembre 2016
RECUPERACIÓ (DEMO+EXAMEN):	25 de Gener 2018

I RECORDEU:

"Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live." Martin Golding

Enric Martí
Octubre 2017