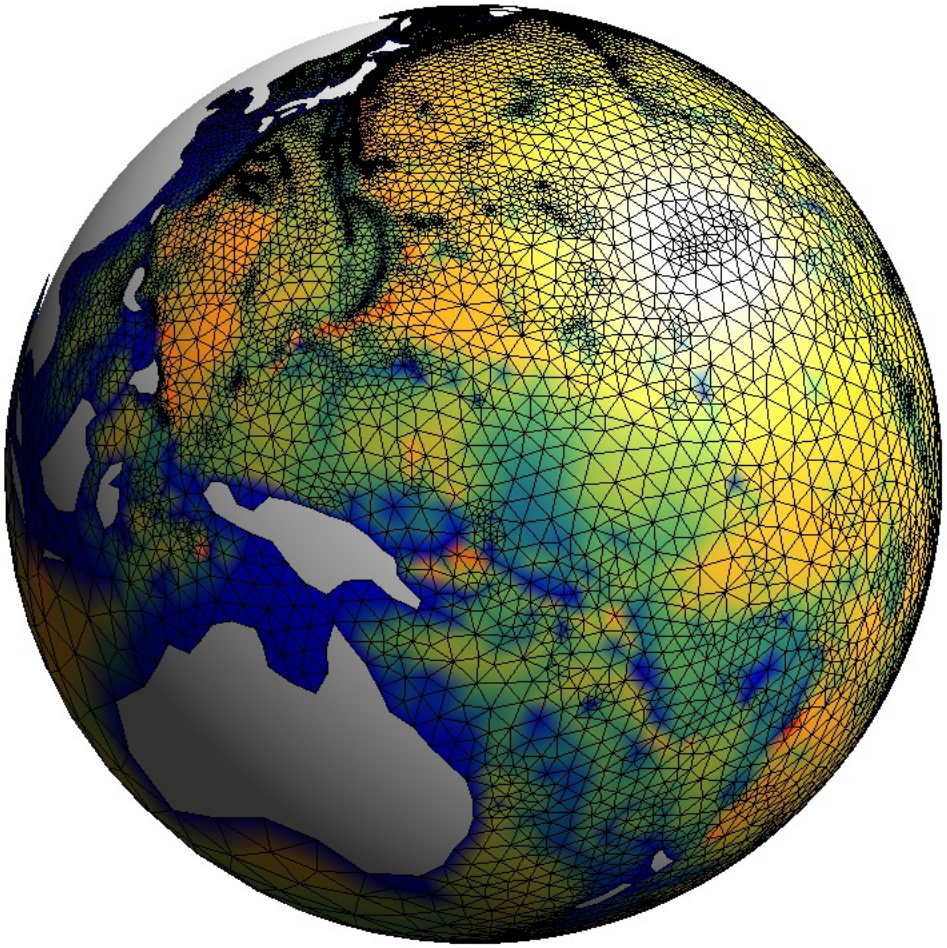
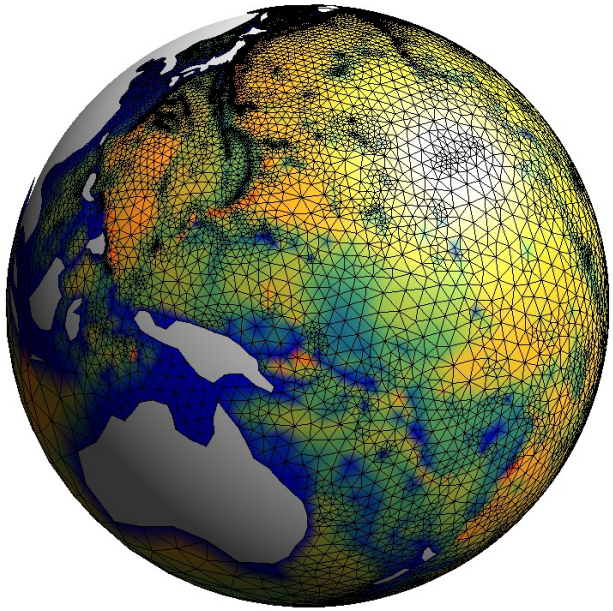
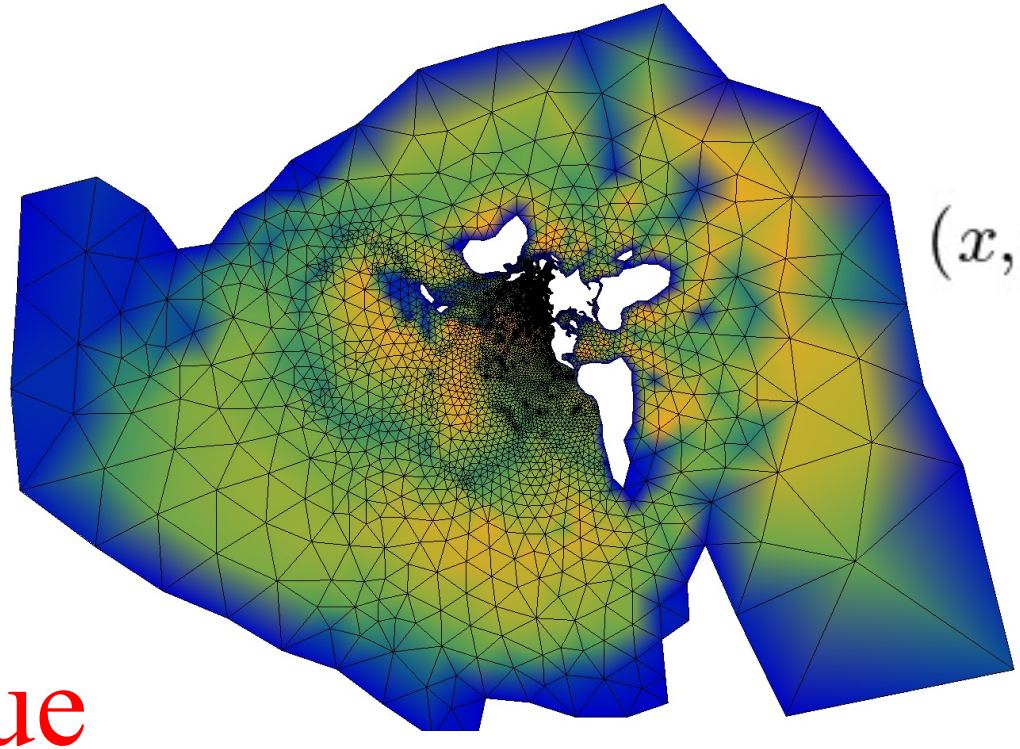
Un tout petit mot sur le projet



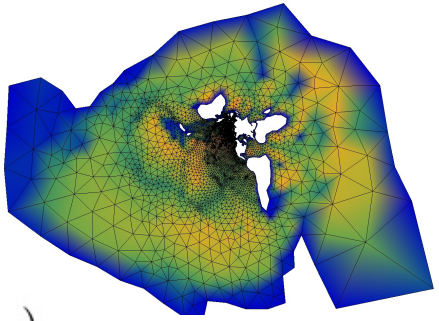


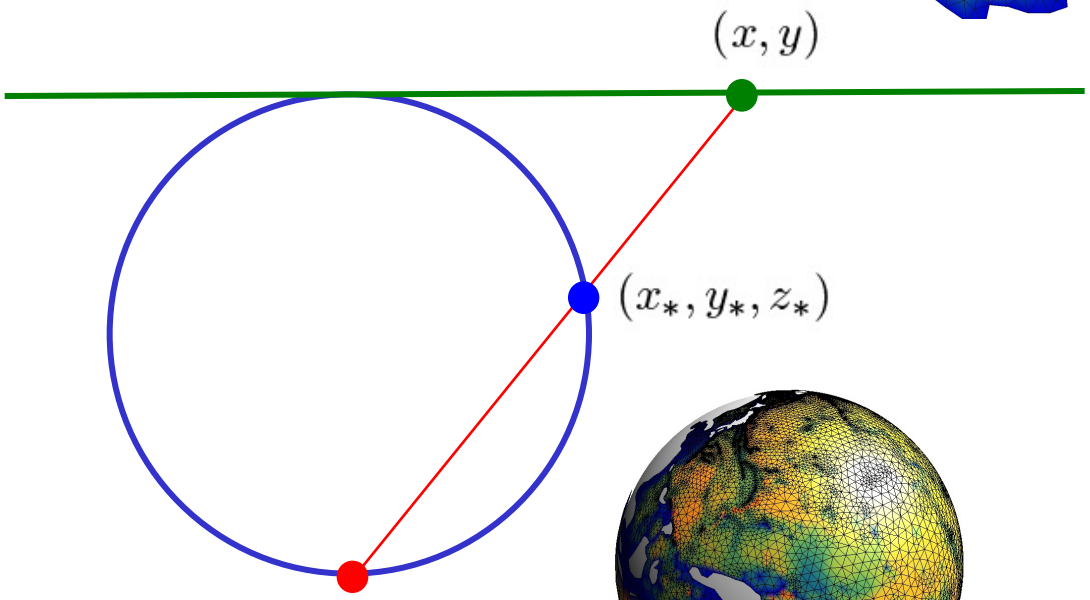
(x_*, y_*, z_*)



(x, y)

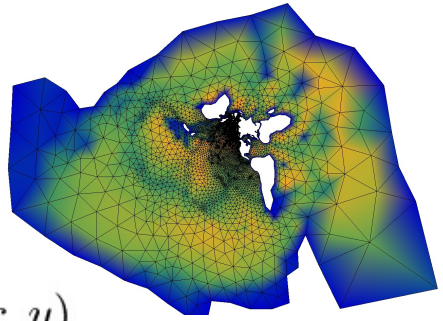
La projection stéréographique

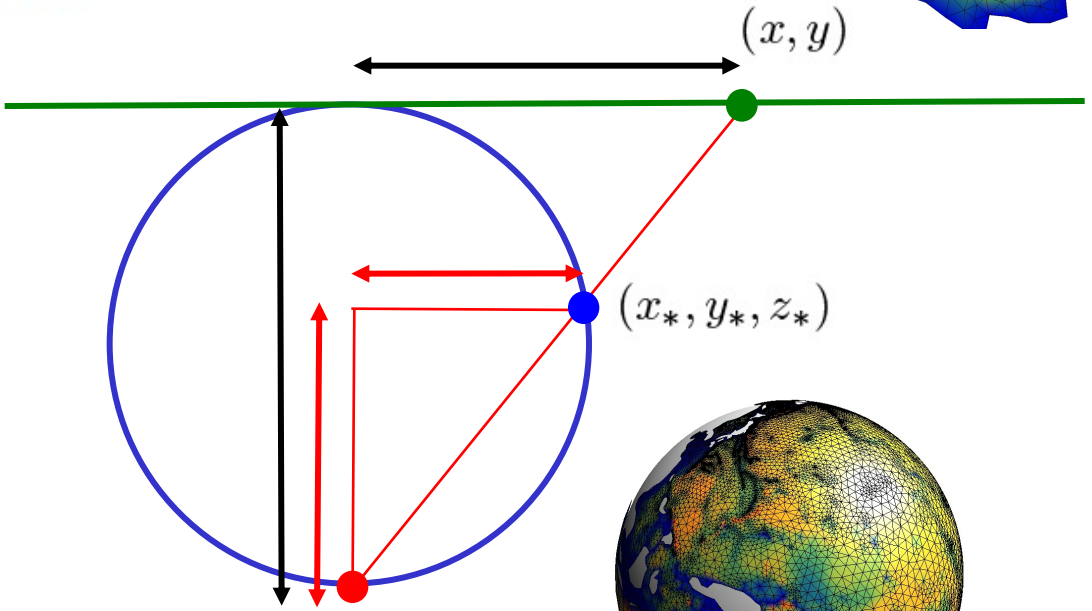




Le pôle sud est le
pôle de notre projection !

$$\frac{x}{x_*} = \frac{2R}{R + z_*}$$

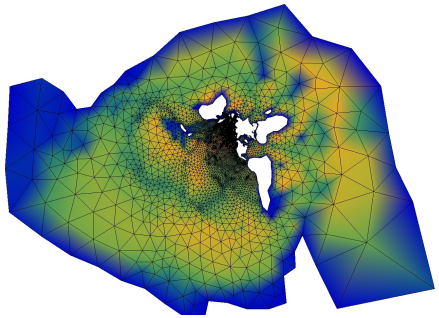


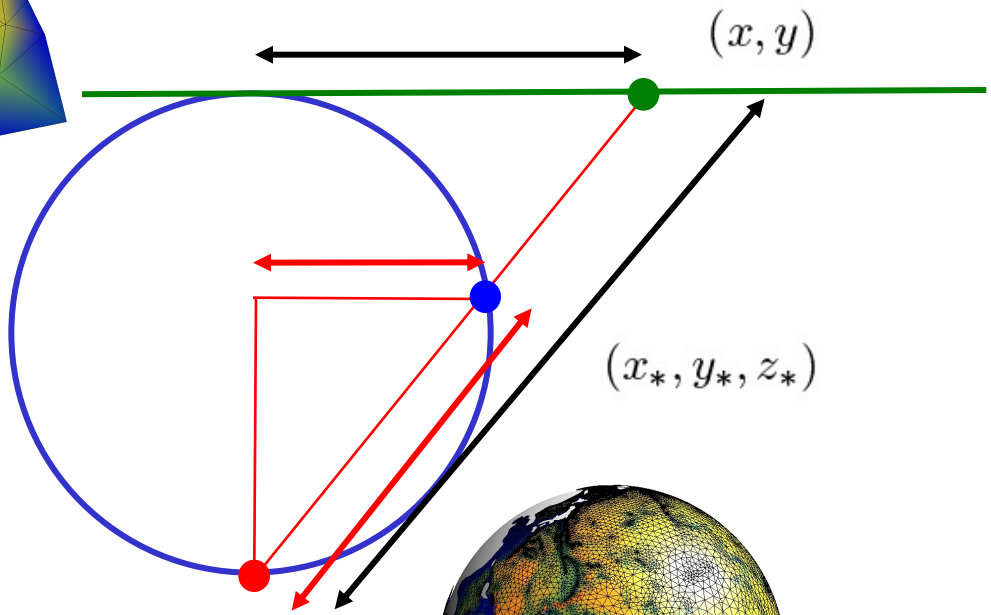


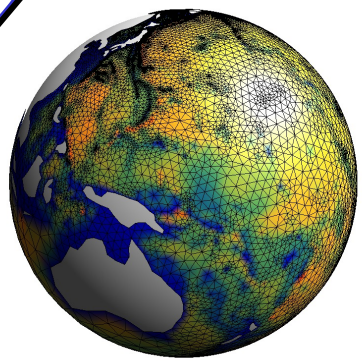
Un fifrelin de géométrie

$$\frac{x^2}{x_*^2} = \frac{4R^2+x^2}{(R+z_*)^2+x_*^2} = \frac{4R^2+x^2}{R^2+\underbrace{z_*^2+x_*^2}_{R^2}+2Rz_*} = \frac{1}{2R}\frac{2R}{(R+z_*)}\frac{4R^2+x^2}{2R} = \frac{x}{2Rx_*}\frac{4R^2+x^2}{2R}$$

$$x_* = \frac{4R^2x}{4R^2+x^2}$$





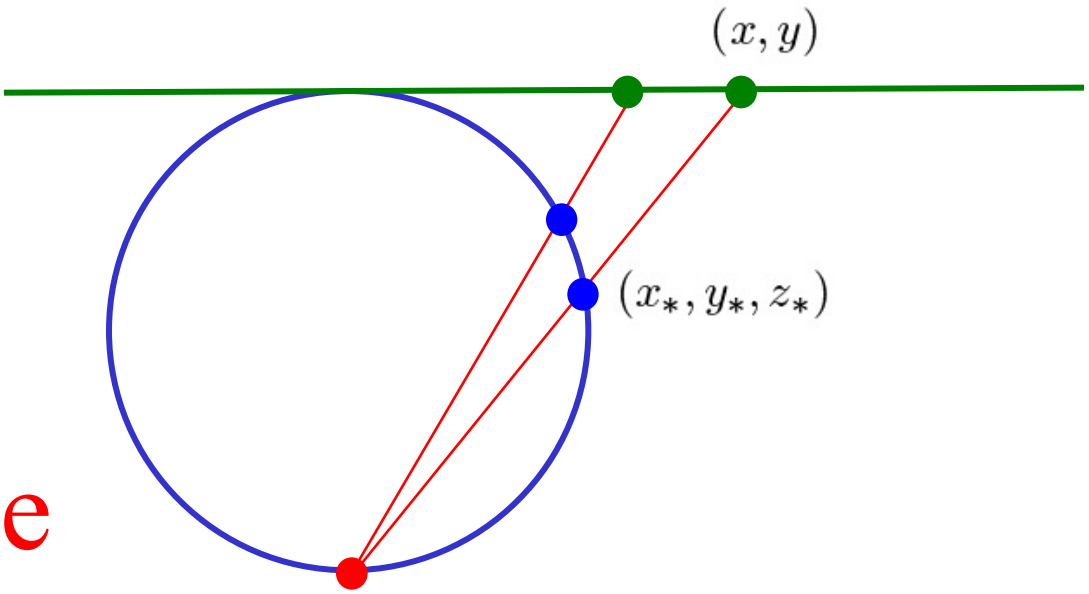


Un fifrelin d'algèbre

Et ce truc là ?

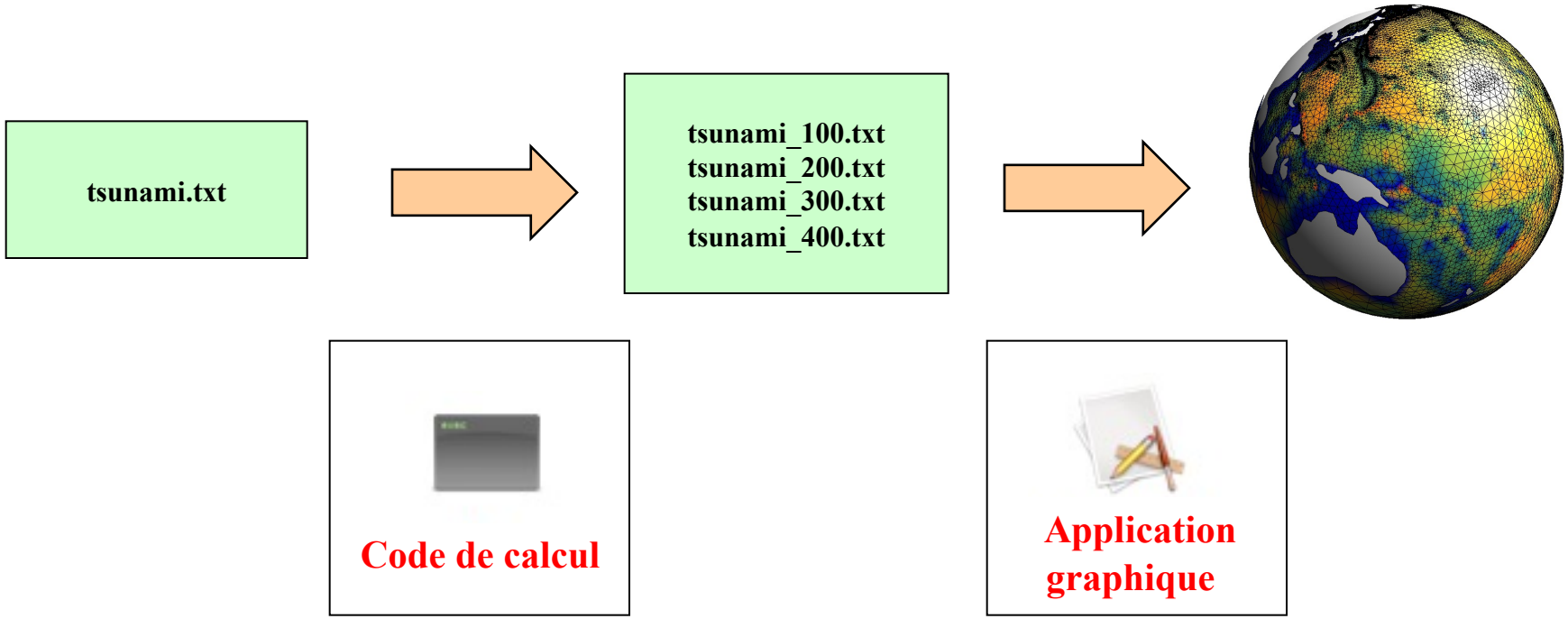
$$\begin{cases} \dfrac{\partial \eta}{\partial t} + \left(\dfrac{4R^2+x^2+y^2}{4R^2}\right)\dfrac{\partial}{\partial x}(hu) + \left(\dfrac{4R^2+x^2+y^2}{4R^2}\right)\dfrac{\partial}{\partial y}(hv) = \boxed{\dfrac{(xu+yv)h}{2R^2}} \\ \dfrac{\partial u}{\partial t} + \left(\dfrac{4R^2+x^2+y^2}{4R^2}\right)\dfrac{\partial}{\partial x}(g\eta) = -\gamma u + fv \\ \dfrac{\partial v}{\partial t} + \boxed{\left(\dfrac{4R^2+x^2+y^2}{4R^2}\right)}\dfrac{\partial}{\partial y}(g\eta) = -\gamma v - fu \end{cases}$$

Rapport des longueurs



Les équations dans le plan stéréographique

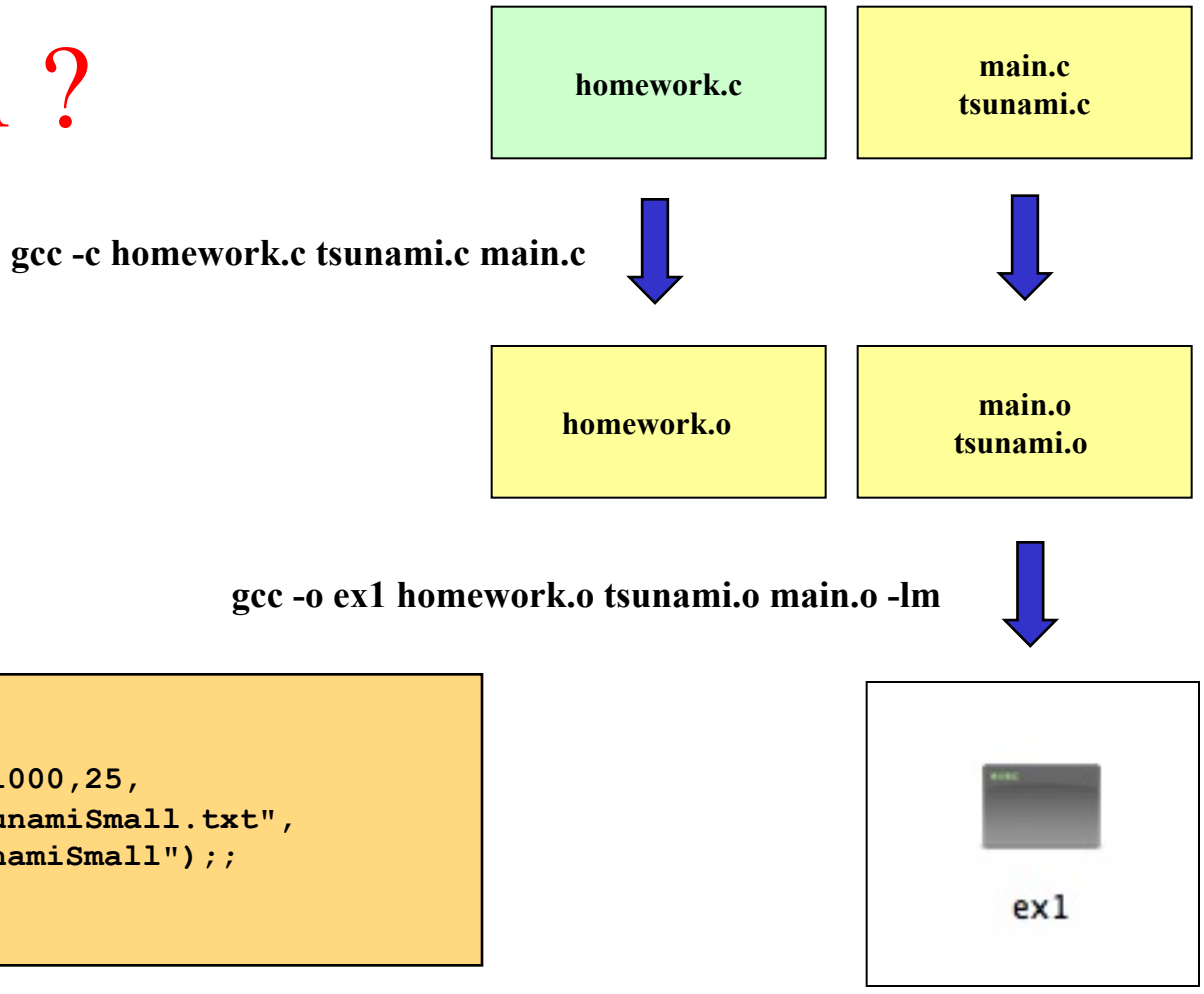
Deux applications distinctes :-)



Ceci est optionnel !
Introduction à OpenGL

C'est quoi le programme de calcul ?

l'unique fichier à soumettre




```
int main(void)
{
    tsunamiCompute(2.0,1000,25,
            "../data/tsunamiSmall.txt",
            "output/tsunamiSmall");;
    exit(0);
}
```

```
int main(void)
{
    tsunamiAnimate(2.0,1000,25,
            "../data/tsunamiSmall.txt",
            "output/tsunamiSmall");;
    exit(0);
}
```

Les fichiers de votre application !

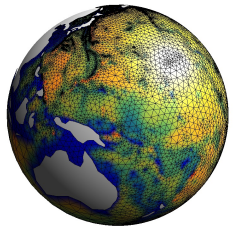
fun.c | **main.c tsunami.c**

gcc -c fun.c tsunami.c main.c

GLFW OpenGL

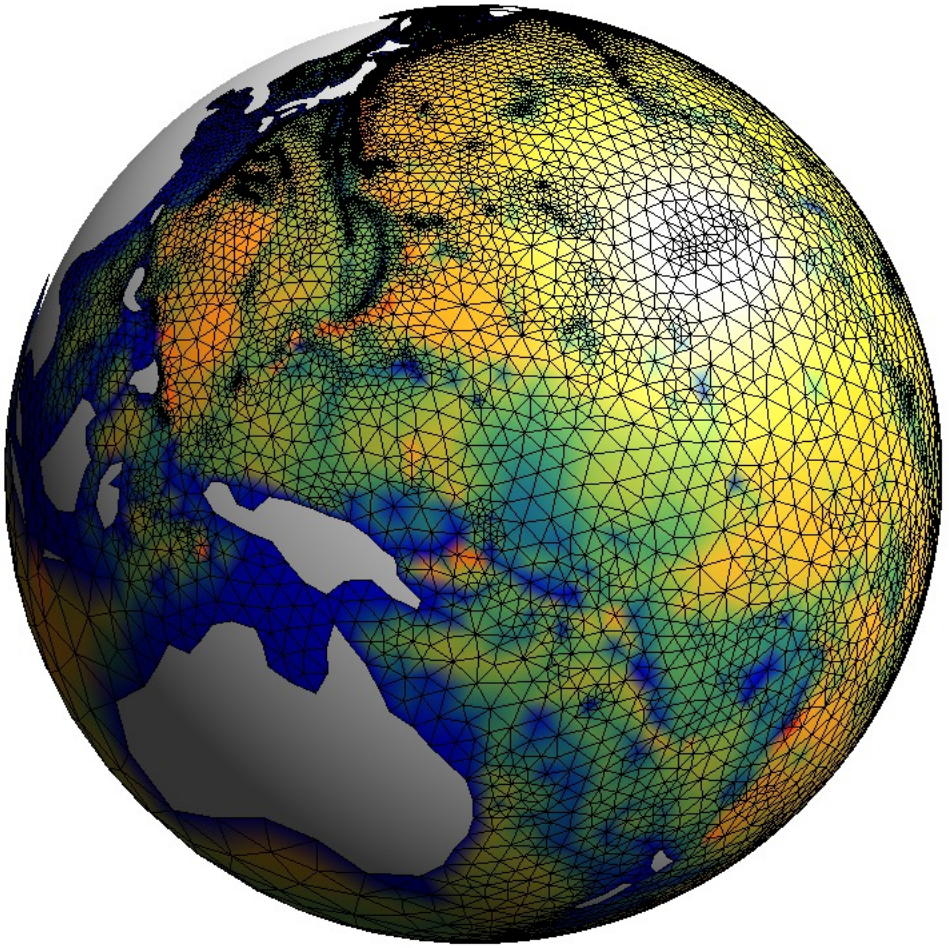
fun.o | **main.o tsunami.o**

gcc -o fun *.o GL/*.a –Dgraphics -framework cocoa -framework OpenGL

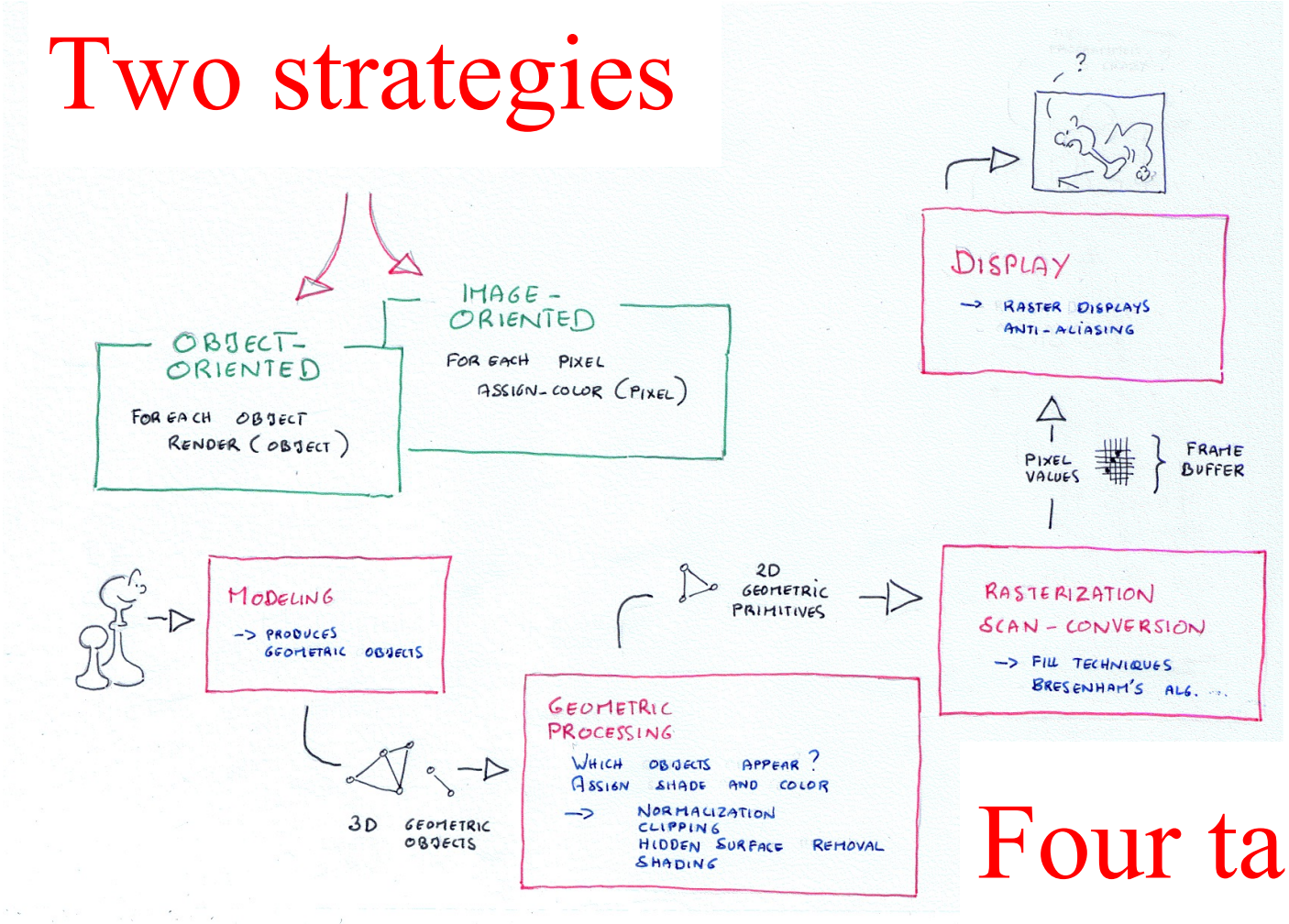


Et l’application graphique !

OpenGL for dummies

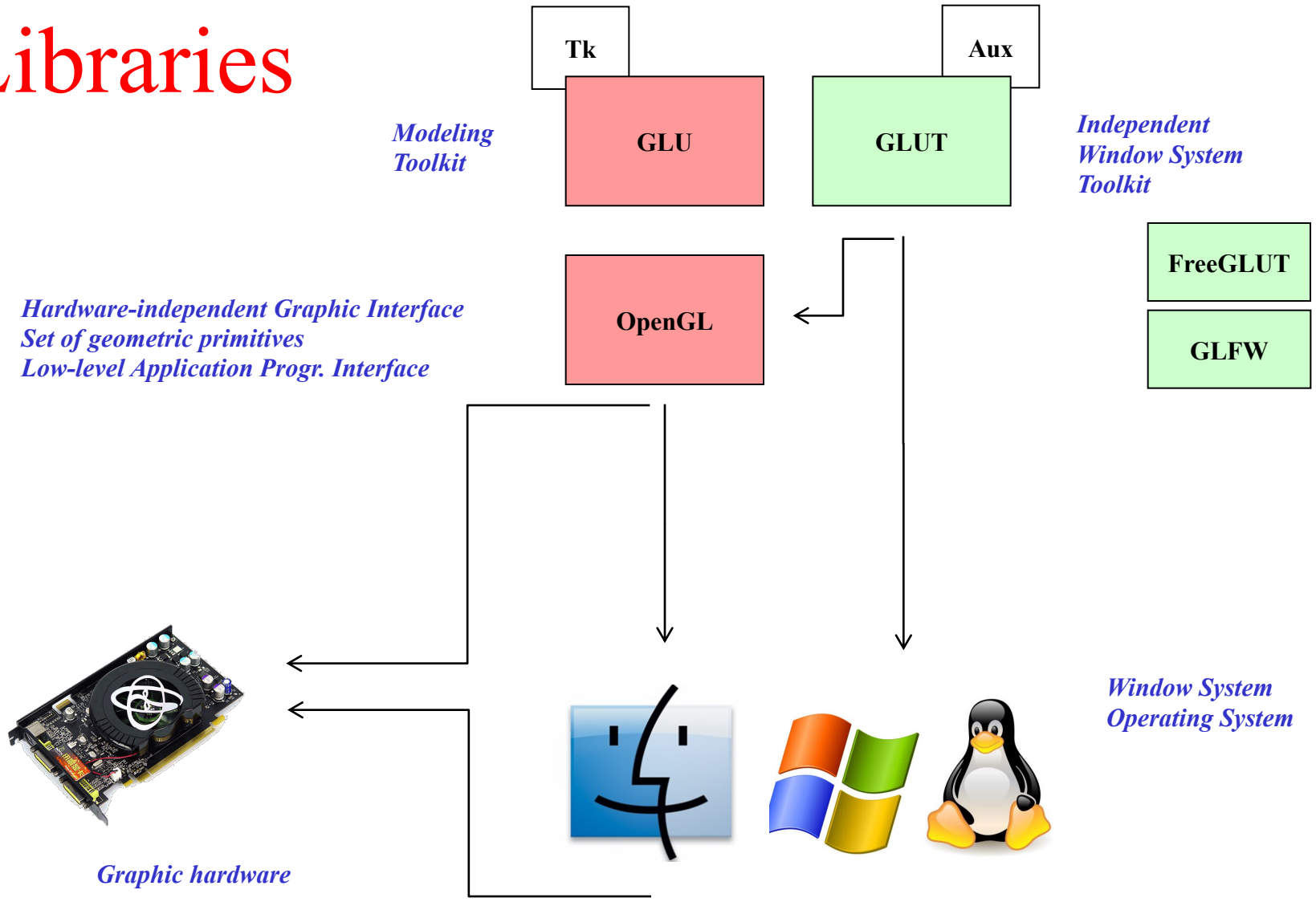


Two strategies



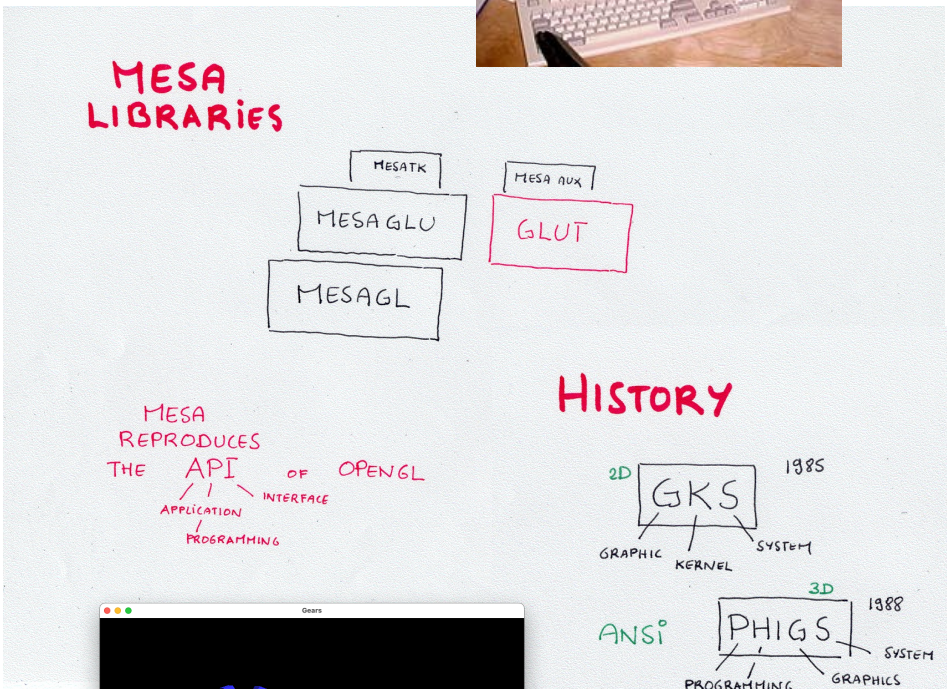
Four tasks

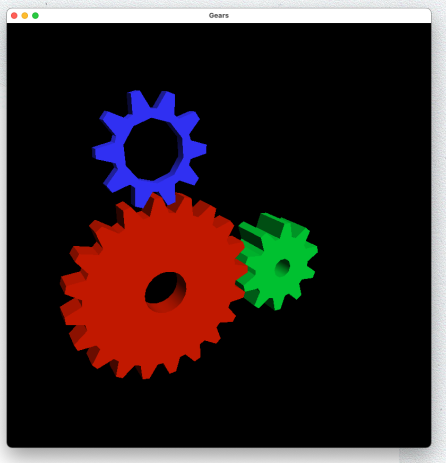
OpenGL Libraries



The purple tower design is a real classic, one of the best looking SGI machines ever made…

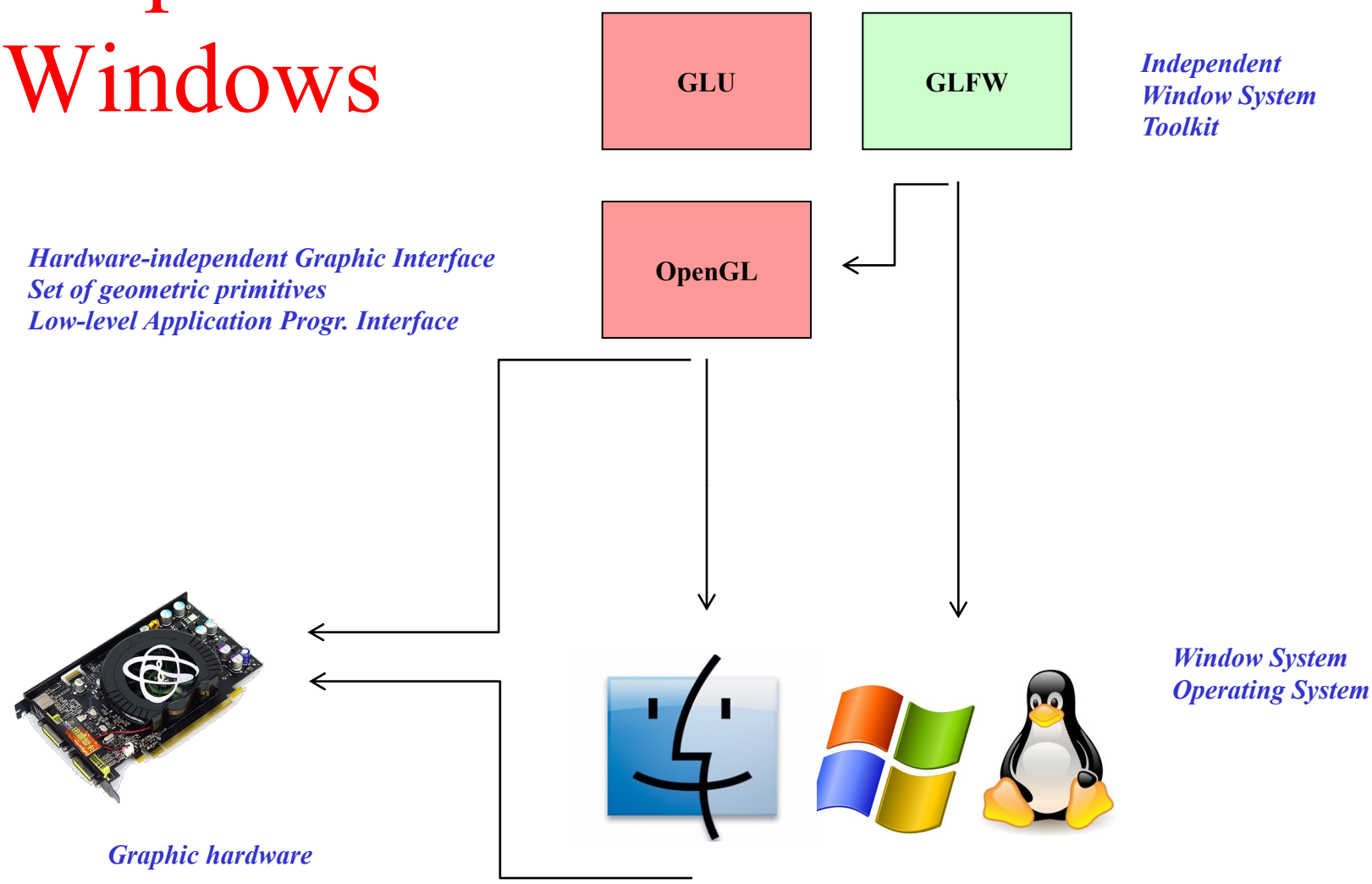
Un petit mot d'histoire





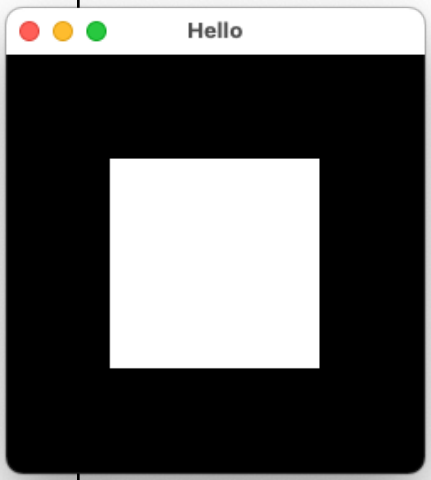
GLFW : le plus simple sous Windows

GLFW is a free, Open Source, multi-platform library for opening a window, creating an OpenGL context and managing input.



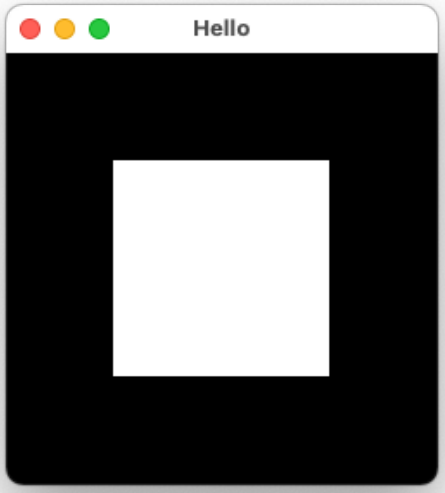
hello.c

```
int main()
{
    glfwInit();
    GLFWwindow* window = glfwCreateWindow(250,250,"Hello",NULL,NULL);
    glfwMakeContextCurrent(window);
    glfwSwapInterval(1);  $

    glClearColor(0.0,0.0,0.0,0.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0.0,1.0,0.0,1.0,-1.0,1.0);

    do { display();
         glfwSwapBuffers(window);
         glfwPollEvents();

    } while( glfwGetKey(window,GLFW_KEY_ESCAPE) != GLFW_PRESS &&
             glfwWindowShouldClose(window)

    glfwTerminate();
    exit(EXIT_SUCCESS);
}
```

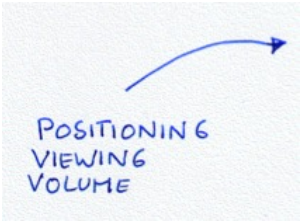


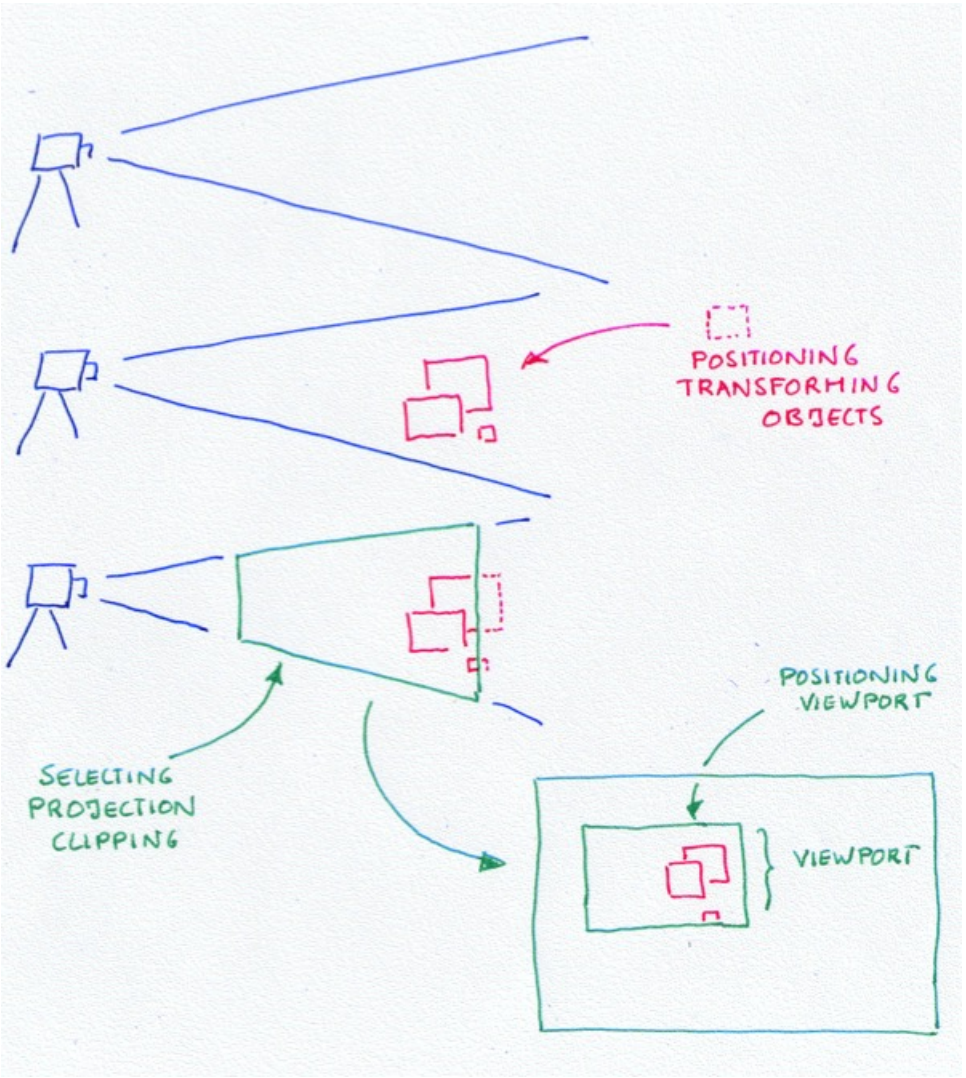

```
void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0,1.0,1.0);
    glBegin(GL_POLYGON);
          glVertex3f(0.25,0.25,0.0);
          glVertex3f(0.75,0.25,0.0);
          glVertex3f(0.75,0.75,0.0);
          glVertex3f(0.25,0.75,0.0);
    glEnd();
    glFlush();
}
```

Comment obtenir l'exécutable ?



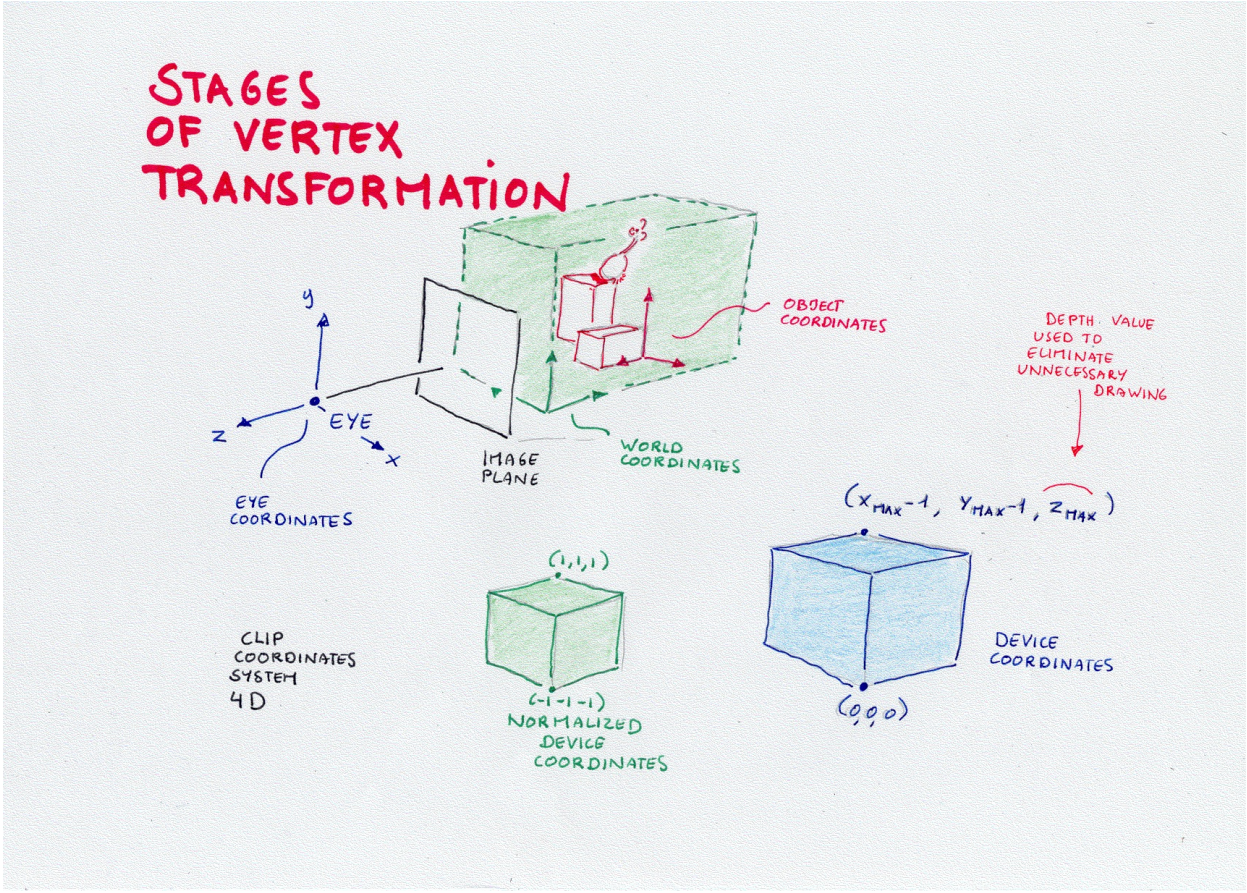

```
cc  -isysroot MacOSX12.3.sdk -mmacosx-version-min=12.0 -Wl,-search_paths_first
-Wl,-headerpad_max_install_names  hello.c -o hello glfw/src/libglfw3.a -
framework OpenGL -framework Cocoa -framework IOKit -framework CoreFoundation -
framework CoreVideo
```

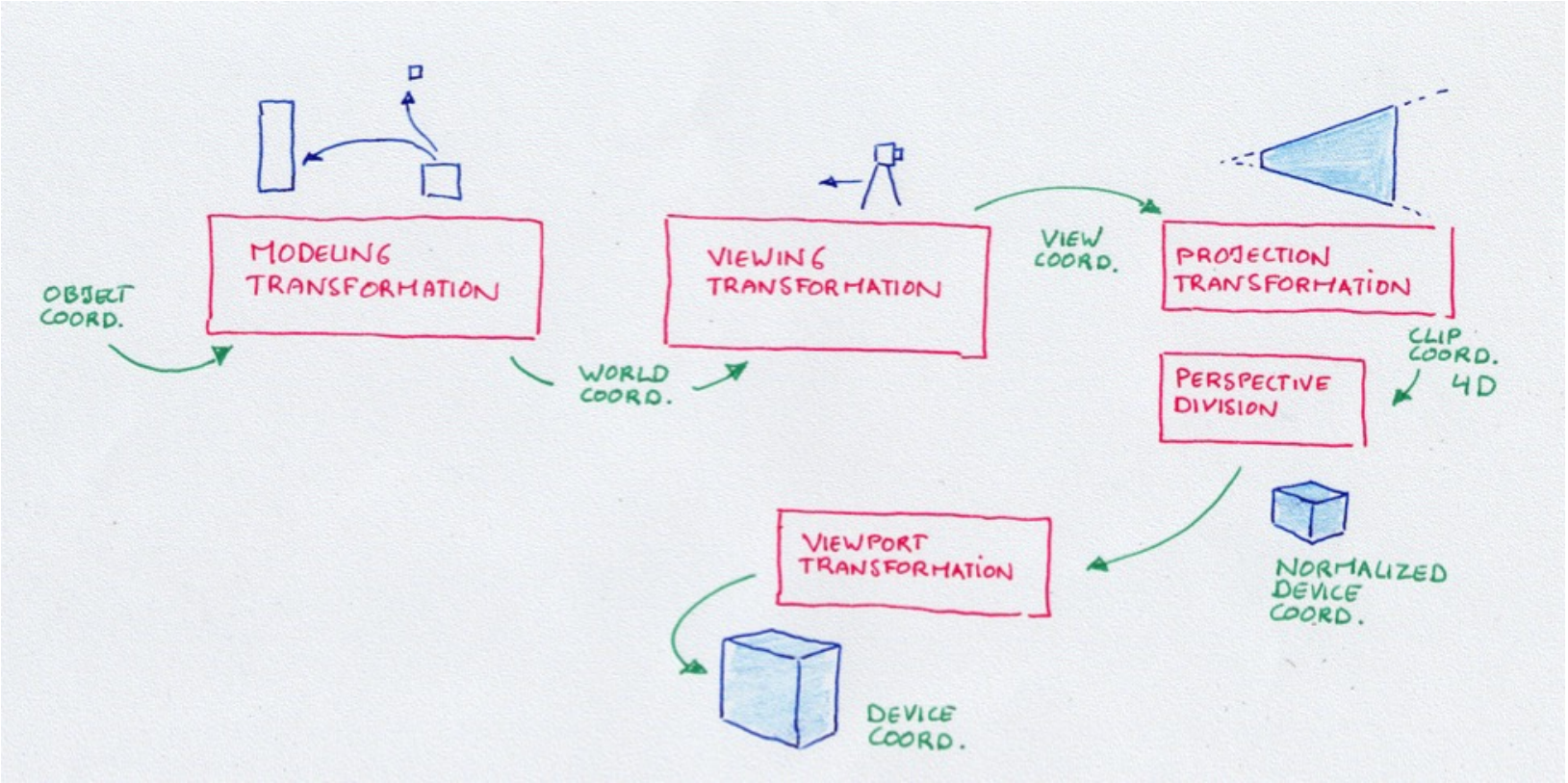




Analogie de la caméra

OpenGL : cela transforme des coordonnées…

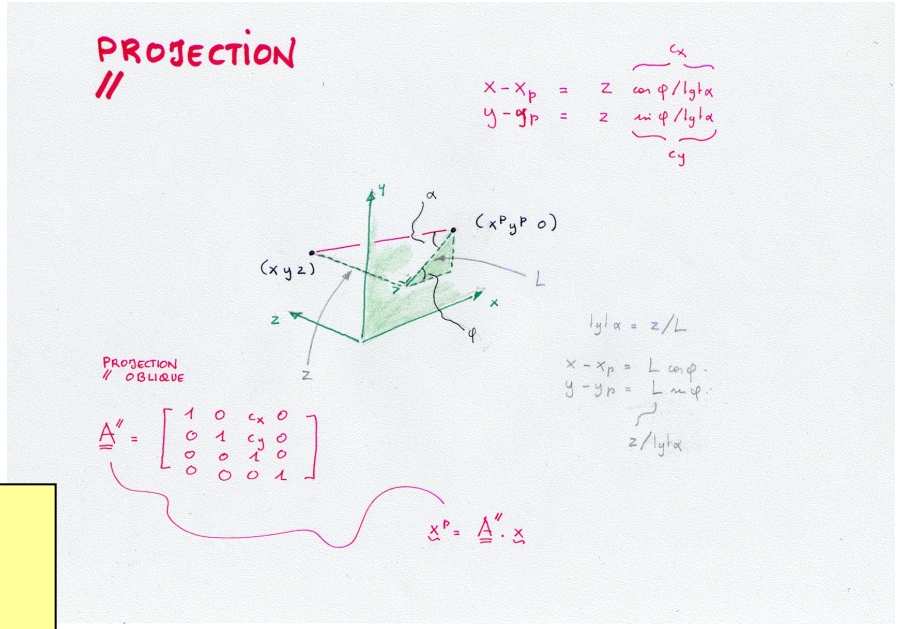


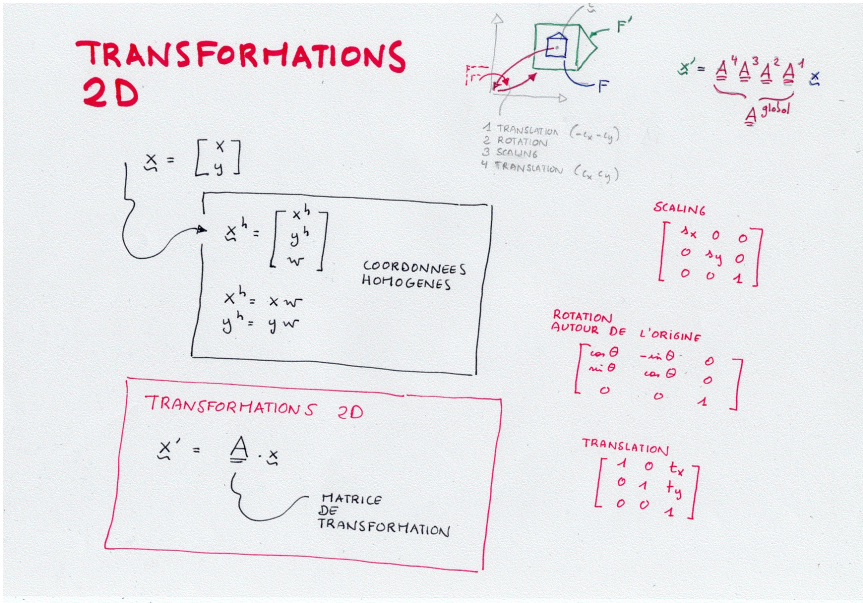


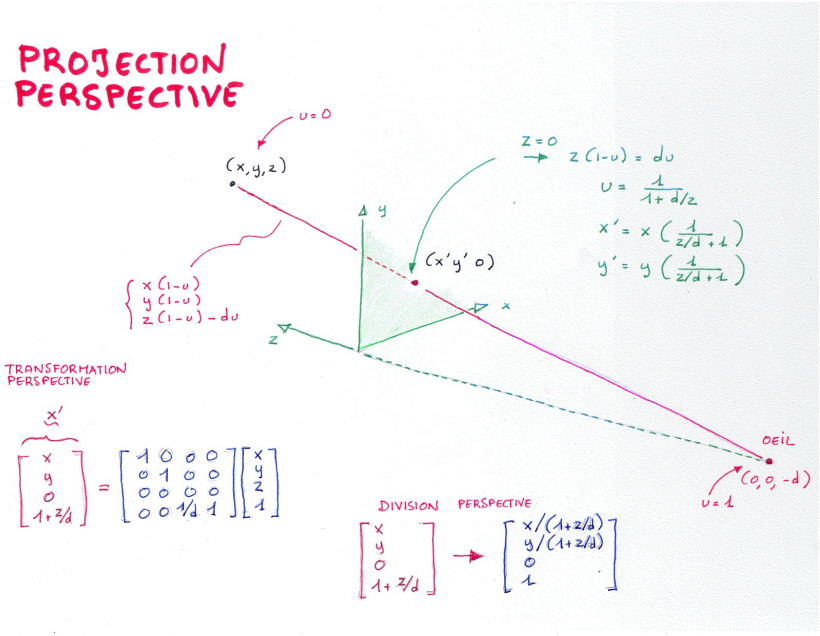
… via un pipe-line !

Principe simple : Accumuler les transformations

L'utilisation de 4 coordonnées homogènes permet de voir toutes les transformations comme le produit par une matrice de tansformation !







cube.c

```
int main()
{
    glfwInit();
    GLFWwindow* window = glfwCreateWindow(250,250,"Cube",NULL,NULL);
    glfwMakeContextCurrent(window);
    glfwSwapInterval(1);

    glClearColor(0.0,0.0,0.0,0.0);
    gllShadeModel(GL_FLAT);
    glfwSetWindowSizeCallback(window,reshape);
    glfwSetKeyCallback(window,keyboard);
    int width,height;
    glfwGetFramebufferSize(window,&width,&height);
    reshape(window,width,height);

    do {
          display();
          glfwSwapBuffers(window);
          glfwPollEvents();


    } while(glfwGetKey(window,GLFW
               glfwWindowShouldClos

    glfwTerminate();
    exit(EXIT_SUCCESS);}
```

```
void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0,1.0,1.0);
    glLoadIdentity();
    gluLookAt(0.0,0.0,5.0,0.0,0.0,0.0,0.0,1.0,0.0);
    glScalef(1.0,2.0,1.0);
    drawBox(2.0,GL_LINE_LOOP);
    glFlush();
}
```

Installer la caméra

Viewing transformation

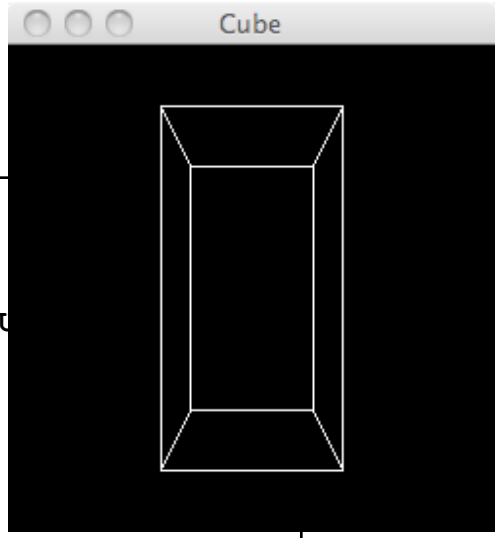
Modeling transformation

```
void display(void)
{
   glClear(GL_COLOR_BUFFER_BIT);
   glColor3f(1.0,1.0,1.0);
   glLoadIdentity();
   gluLookAt(0.0,0.0,5.0,
             0.0,0.0,0.0,
             0.0,1.0,0.0);
   glScalef(1.0,2.0,1.0);
   glutWireCube(2.0);
   glFlush();
}
```

Places the camera at (0,0,5)
Aims the camera lens towards (0,0,0)
Up-vector is (0,1,0)

This defines a unique position and orientation of the camera

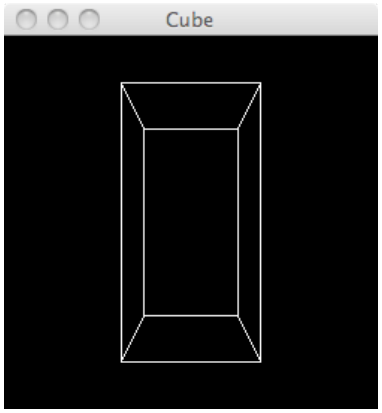
cube.c




```
int main()
{
   glfwInit();
   GLFWwindow* window = glfwCreateWindow(250,250,"Cube",NU
   glfwMakeContextCurrent(window);
   glfwSwapInterval(1);

   glClearColor(0.0,0.0,0.0,0.0);
   gllShadeModel(GL_FLAT);
   glfwSetWindowSizeCallback(window,reshape);
   glfwSetKeyCallback(window,keyboard);
   int width,height;
   glfwGetFramebufferSize(window,&width,&height);
   reshape(window,width,height);

   do {
        display();
        glfwSwapBuffers(window);
        glfwPollEvents();


   } while(glfwGetKey(window,GLFW_KEY_E
            glfwWindowShouldClose(wind

   glfwTerminate();
   exit(EXIT_SUCCESS);}
```

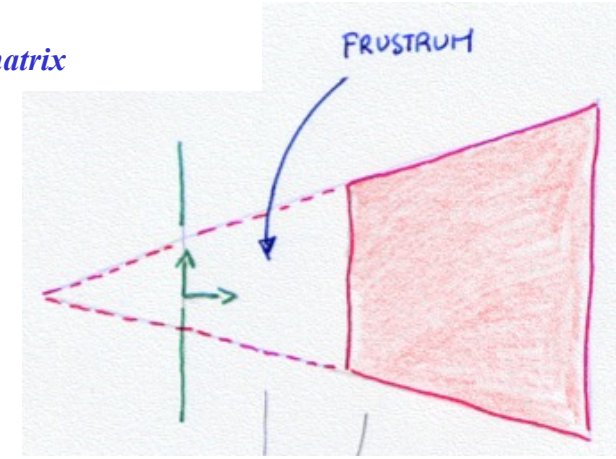
```
void reshape(GLFWwindow* window, int w, int h)
{
   glViewport(0,0,(GLsizei)w,(GLsizei) h);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   glFrustum(-1.0,1.0,-1.0,1.0,1.5,20.0);
   glMatrixMode(GL_MODELVIEW);
}
```

Volume de vue



Defines left and right clipping planes,
Defines bottom and top clipping planes,
Defines zNear and zFar

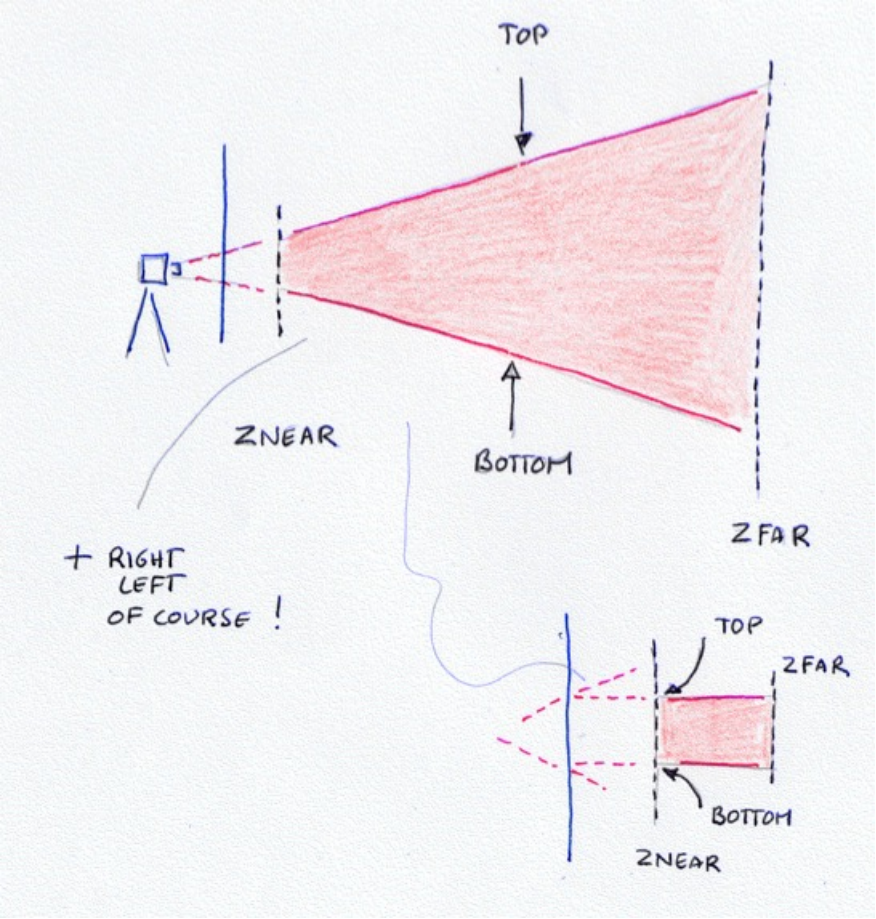
This defines a perpective matrix



Projection transformation →

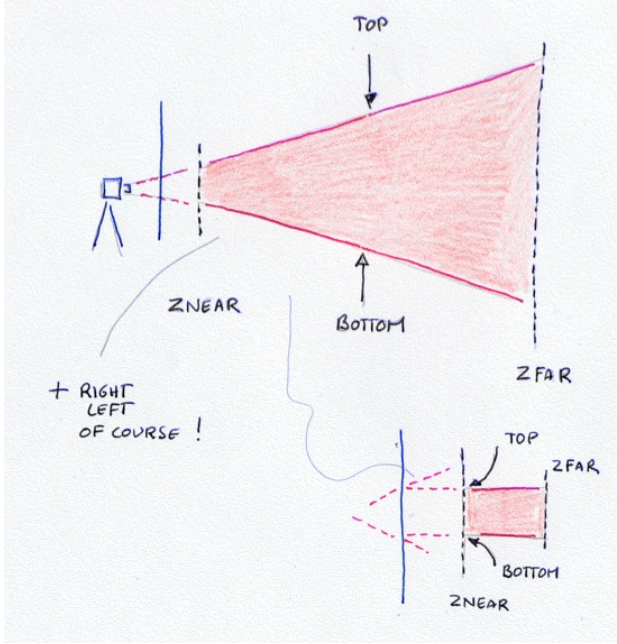
```
void reshape(GLFWwindow* window, int w, int h)
{
   glViewport(0,0,(GLsizei)w,(GLsizei) h);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   glFrustum(-1.0,1.0,-1.0,1.0,1.5,20.0);
   glMatrixMode(GL_MODELVIEW);
}
```

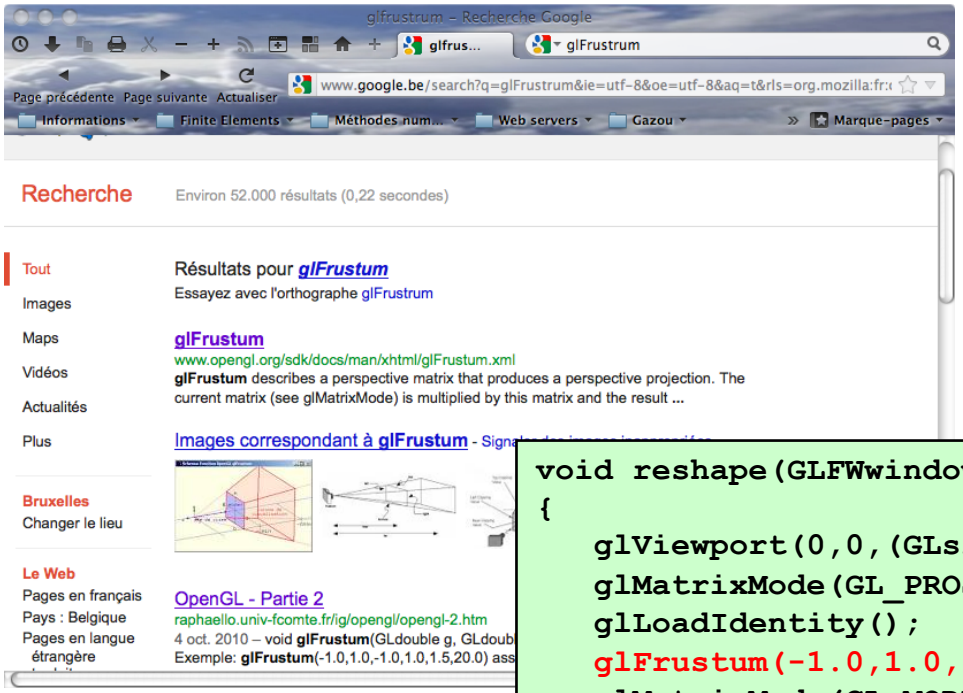
glFrustum




```
void reshape(GLFWwindow* window, int w, int h)
{
    glViewport(0,0,(GLsizei)w,(GLsizei) h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glFrustum(-1.0,1.0,-1.0,1.0,1.5,20.0);
    glMatrixMode(GL_MODELVIEW);
}
```

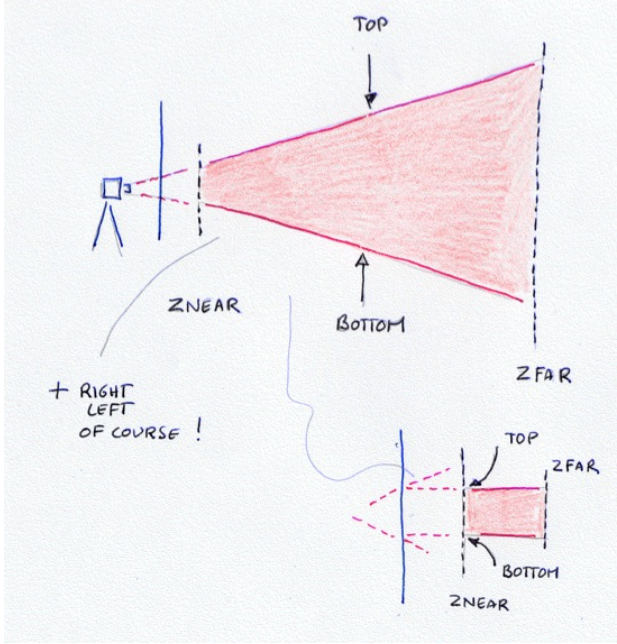
How to obtain the specification of the parameters ?






```
void reshape(GLFWwindow* window, int w, int h)
{
    glViewport(0,0,(GLsizei)w,(GLsizei) h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glFrustum(-1.0,1.0,-1.0,1.0,1.5,20.0);
    glMatrixMode(GL_MODELVIEW);
}
```

How to obtain the specification of the parameters ?



Mozilla Firefox

htt...ml — glFrustrum

Page précédente | Page suivante | Actualiser | perso.uclouvain.be/vincent.legat/teaching/meca2170,

Informations | Finite Elements | Méthodes num... | Marque-pages

NAME

glFrustum - multiply the current matrix by a perspective matrix

C SPECIFICATION

```
void glFrustum( GLdouble left,
                GLdouble right,
                GLdouble bottom,
                GLdouble top,
                GLdouble zNear,
                GLdouble zFar )
```

PARAMETERS

left, *right* Specify the coordinates for the left and right vertical clipping planes.

bottom, *top* Specify the coordinates for the bottom and top horizontal clipping planes.

zNear, *zFar* Specify the distances to the near and far depth clipping planes. Both distances must be positive.

DESCRIPTION

glFrustum describes a perspective matrix that produces a perspective projection. The current matrix (see **glMatrixMode**) is multiplied by this matrix and the result replaces the current matrix, as if **glMultMatrix** were called with the following matrix as its argument:

$$\begin{pmatrix} \frac{2}{right-left} & 0 & A & 0 \\ 0 & \frac{2}{top-bottom} & B & 0 \\ 0 & 0 & C & D \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

$$A = \frac{right+left}{right-left}$$

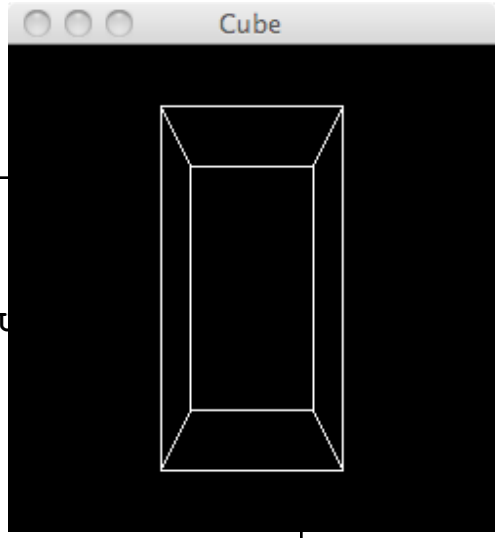
$$B = \frac{top+bottom}{top-bottom}$$

$$C = \frac{-zFar-zNear}{zFar-zNear}$$

```
void reshape(GLFWwindow* window, int w, int h)
{
    glViewport(0,0,(GLsizei)w,(GLsizei) h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glFrustum(-1.0,1.0,-1.0,1.0,1.5,20.0);
    glMatrixMode(GL_MODELVIEW);
}
```

http://perso.uclouvain.be/vincent.legat/teaching/meca2170/opengl/gl/frustum.html

cube.c




```
int main()
{
    glfwInit();
    GLFWwindow* window = glfwCreateWindow(250,250,"Cube",NU


    glfwSetWindowSizeCallback(window,reshape);
    glfwSetKeyCallback(window,keyboard);

    do {
          display();
          glfwSwapBuffers(window);
          glfwPollEvents();

    } while(glfwGetKey(window,GLFW_KEY_E
              glfwWindowShouldClose(wind

    glfwTerminate();
    exit(EXIT_SUCCESS);}
```

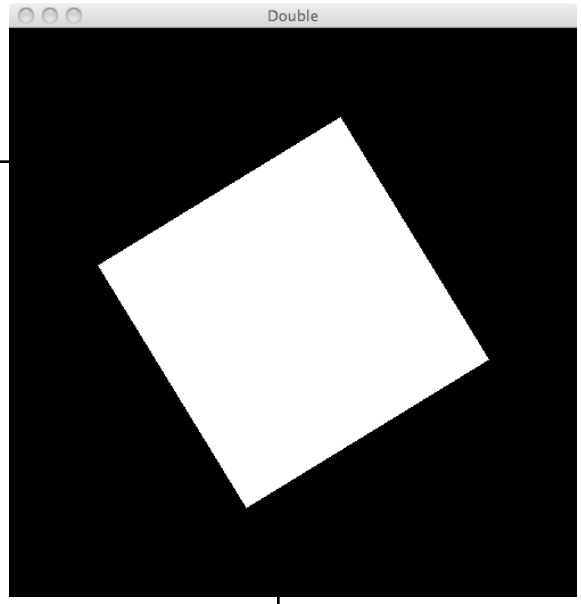
```
void reshape(GLFWwindow* window, int w, int h)
{
    glViewport(0,0,(GLsizei)w,(GLsizei) h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glFrustum(-1.0,1.0,-1.0,1.0,1.5,20.0);
    glMatrixMode(GL_MODELVIEW);
}
```

```
void keyboard(GLFWwindow* window, int key,
              int scancode, int action, int mods)
{
    switch (key) {
       case GLFW_KEY_ESCAPE:
          exit( EXIT_SUCCESS ); break;
    }
}
```

ESC allows the user to exit

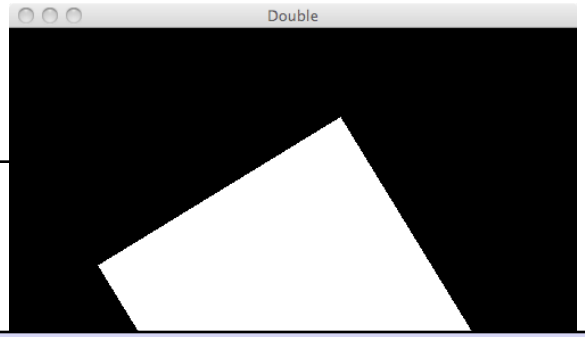
double.c

```
static GLfloat spin = 0.0;
static GLfloat animate = 0.0;

int main(int argc, char** argv)
{
   glfwInit();
   glfwOpenWindow(500,500,0,0,0,0,0,0,GLFW_WINDOW );
   glfwSetWindowTitle("Double");
   glfwSwapInterval( 1 );

   glClearColor (0.0, 0.0, 0.0, 0.0);
   glShadeModel (GL_FLAT);
   glfwSetWindowSizeCallback(reshape);
   glfwSetKeyCallback(keyboard);
   glfwSetMouseButtonCallback(mouse);

   do {
        spin = spin + animate * glfwGetTime() * 0.1f;
        display();
        glfwSwapBuffers();
   } while( glfwGetWindowParam(GLFW_OPENED));

   glfwTerminate();
   exit( EXIT_SUCCESS );
}
```




```
void display(void)
{
   glClear(GL_COLOR_BUFFER_BIT);
   glPushMatrix();
   glRotatef(spin,0.0,0.0,1.0);
   glColor3f(1.0, 1.0, 1.0);
   glRectf(-25.0,-25.0,25.0,25.0);
   glPopMatrix();
}
```

double.c

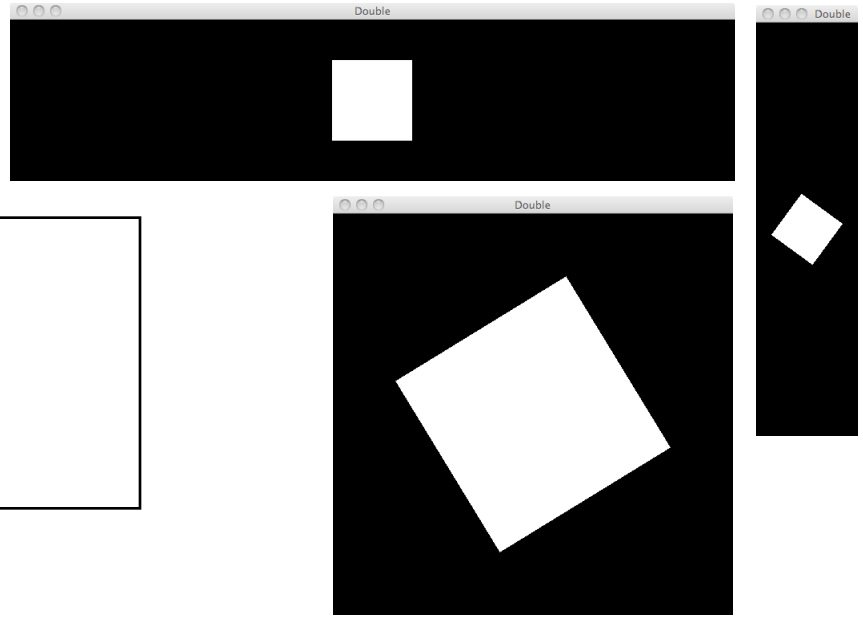



```
static GLfloat spin = 0.0;
static GLfloat animate = 0.0;

int main(int argc, char** argv)
{
   glfwInit();
   glfwOpenWindow(500,500,0,0,0,0,0,0,GLFW_W
   glfwSetWindowTitle("Double");
   glfwSwapInterval( 1 );

   glClearColor (0.0, 0.0, 0.0, 0.0);
   glShadeModel (GL_FLAT);
   glfwSetWindowSizeCallback(reshape);
   glfwSetKeyCallback(keyboard);
   glfwSetMouseButtonCallback(mouse);

   do {
        spin = spin + animate * glfwGetTime(
        display();
        glfwSwapBuffers();
   } while( glfwGetWindowParam(GLFW_OPENED))

   glfwTerminate();
   exit( EXIT_SUCCESS );
}
```

```
void GLFWCALL mouse(int button, int state)
{
   switch (button) {
      case GLFW_MOUSE_BUTTON_LEFT:
         if (state == GLFW_PRESS)
            animate = 1.0;
         break;
      case GLFW_MOUSE_BUTTON_MIDDLE:
      case GLFW_MOUSE_BUTTON_RIGHT:
         if (state == GLFW_PRESS)
            animate = 0.0;
         break;
      default:
         break;
   }
}
```

```
void GLFWCALL keyboard(int key, int action)
{
   switch (key) {
      case GLFW_KEY_ESC:
         exit(EXIT_SUCCESS); break;
   }
}
```

double.c

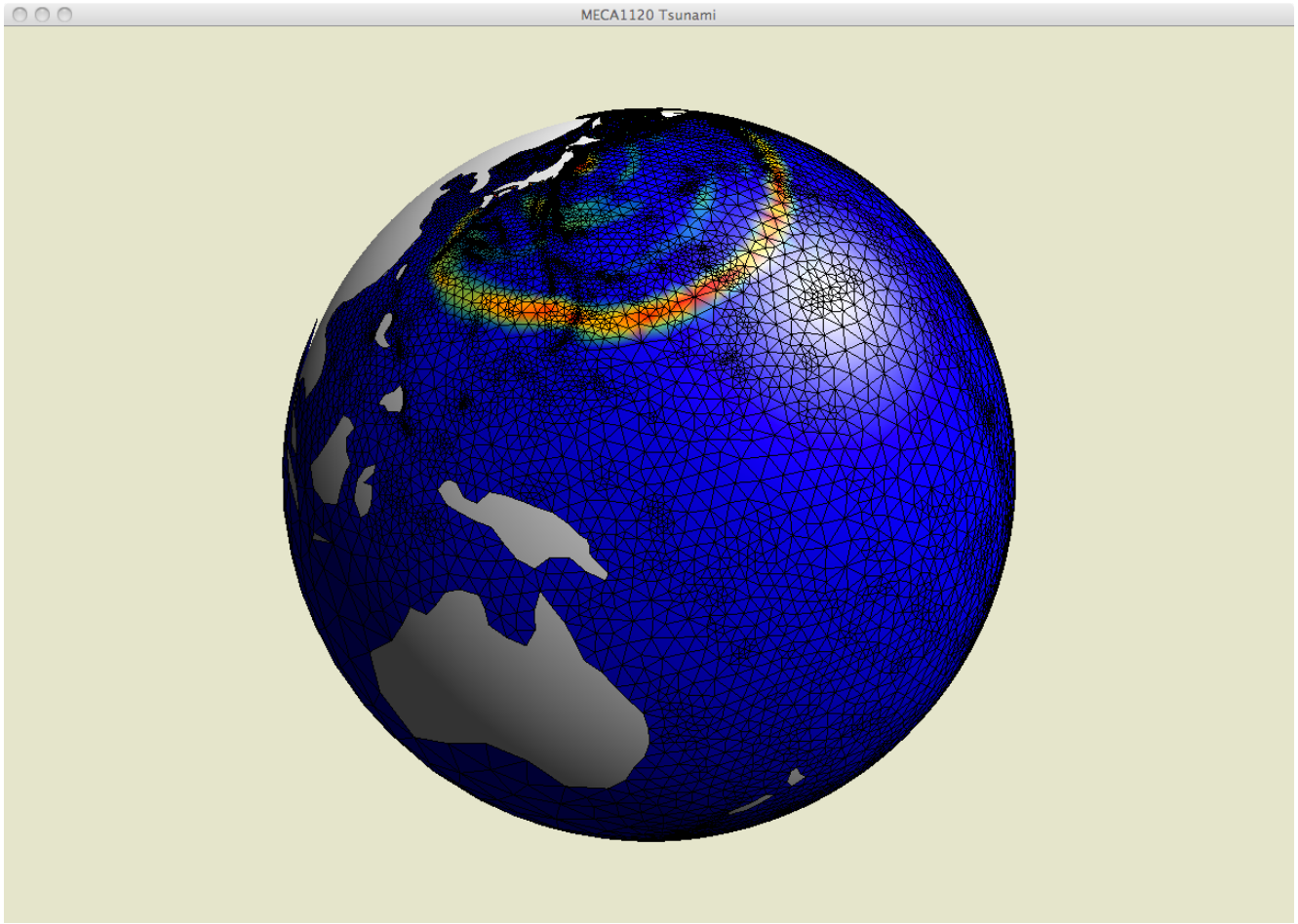



```
glfwSetWindowSizeCallback(reshape);
glfwSetKeyCallback(keyboard);
glfwSetMouseButtonCallback(mouse);
```

Petite astuce permettant d'éviter des déformations géométriques lorsque le format de la fenêtre est modifié !

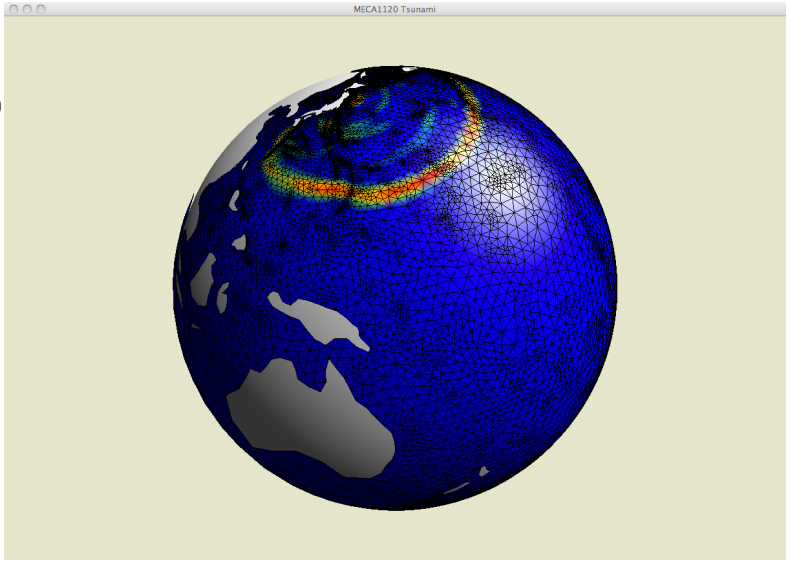
```
void GLFWCALL reshape (int w, int h)
{
    glViewport(0, 0, (GLsizei) w, (GLsizei) h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    if (w <= h)
      glOrtho(-50.0,50.0,
              -50.0*(GLfloat)h/(GLfloat)w,50.0*(GLfloat)h/(GLfloat)w,
               -1.0,1.0);
    else
      glOrtho(-50.0*(GLfloat)w/(GLfloat)h,50.0*(GLfloat)w/(GLfloat)h,
              -50.0, 50.0,
               -1.0,1.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}
```

Et le programme du projet !



```
do {
    t = glfwGetTime();
    frame0 = frame;
    frame = (int)((t-t0)*2);

    if (frame0 != frame) {
      sprintf(filename,basename,frame*nout);
      ... read file ...

      glfwGetWindowSize(&width,&height);
      height = height > 0 ? height : 1;
      glViewport(0,0,width,height);

      glClearColor(0.9f,0.9f,0.8f,0.0f);
      glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      gluPerspective(65.0f,(GLfloat)width/(GLfloat)height,1.0f,100.0f);

      glMatrixMode(GL_MODELVIEW);
      glLoadIdentity();
      gluLookAt(0.0f,1.0f,0.0f,0.0f, 20.0f, 0.0f,0
      glTranslatef(0.0f,14.0f,0.0f);

      ... draw ...
      glfwSwapBuffers();

}} while( glfwGetKey(GLFW_KEY_ESC) != GLFW_PRESS
        && glfwGetWindowParam(GLFW_OPENED) );

glfwTerminate();
exit( EXIT_SUCCESS );
```

C'est exactement la même chose !

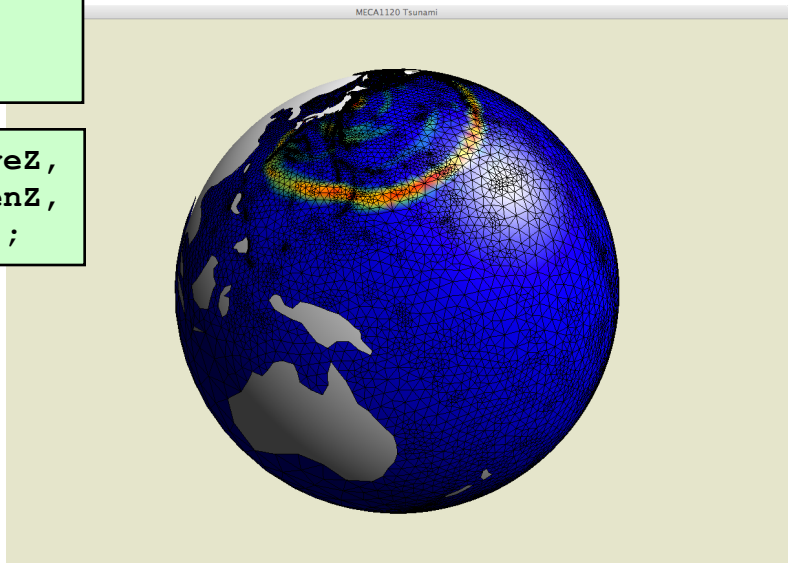


```
glfwGetWindowSize(&width,&height);
glViewport(0,0,width,height);

glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(65.0f,
               (GLfloat)width/(GLfloat)height,
                1.0f,100.0f);

glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(0.0f,1.0f,0.0f,
          0.0f,20.0f,0.0f,
          0.0f,0.0f,1.0f);
glTranslatef(0.0f,14.0f,0.0f);
```

Transformation perspective

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

```
void gluLookAt (GLdouble eyeX eyeY eyeZ,
                GLdouble cenX cenY cenZ,
                GLdouble upX upY upZ);
```



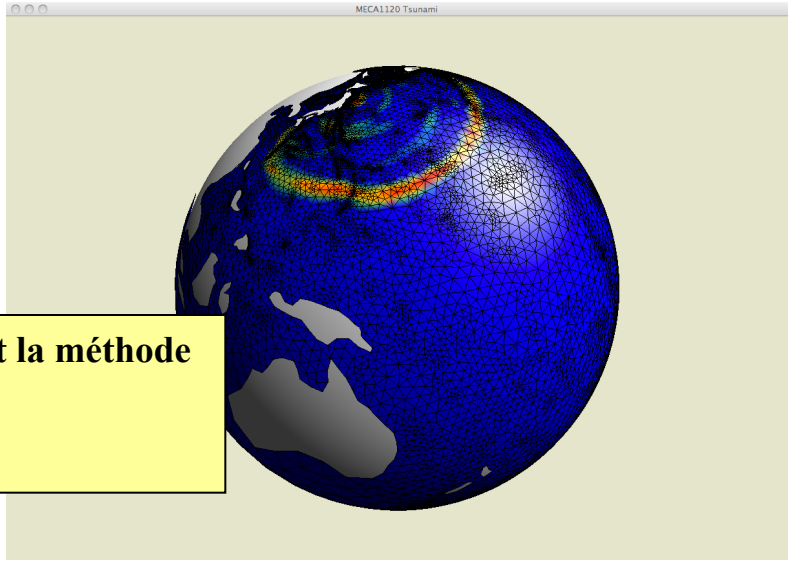
Idéal pour voir une sphère
de rayon 6 centré à l'origine…

Le calcul de la transformation adéquate
pour la Terre est laissé à votre sagacité :-)

```
for (i=0; i < nElem; ++i) {
   for (j=0; j < 3; ++j) {
      index = elem[3*i+j] - 1;

      value = E[3*i+j]*10;
      if (value < 0) value = 0;
      if (value > 1) value = 1;
      colors[j*3+0] = 3.5*(value)*(value);
      colors[j*3+1] = (1-value)*(value)*3.5;
      colors[j*3+2] = (1-value)*(1-value);

      x = X[index];
      y = Y[index];
      factor = (4*R*R + x*x + y*y)*(R/6);
      coord[j*3+0] = 4*R*R * x / factor;
      coord[j*3+1] = 4*R*R * y / factor;
      coord[j*3+2] = (4*R*R - x*x - y*y)*R / factor;  }

   glEnableClientState(GL_VERTEX_ARRAY);
   glEnableClientState(GL_COLOR_ARRAY);
   glVertexPointer(3, GL_FLOAT, 0, coord);
   glColorPointer(3, GL_FLOAT, 0, colors);
   glDrawArrays(GL_TRIANGLES, 0, 3);
   glDisableClientState(GL_COLOR_ARRAY);
   glDisableClientState(GL_VERTEX_ARRAY); }
```

Uniquement l'élévation positive est dessinée :-)
Purement esthétique pour conserver une mer bleue

Plotter l'élévation



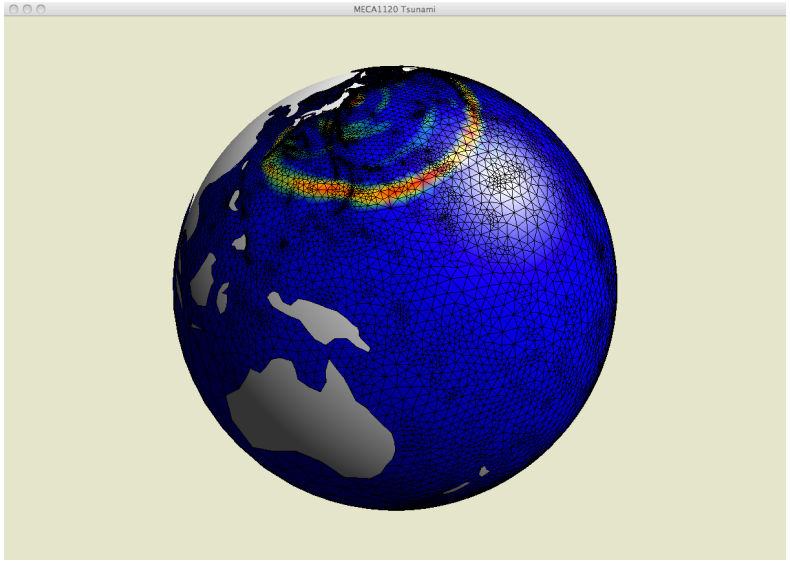
Passer directement un tableau à OpenGL est plus efficace et la méthode conseillée sur les nouvelles implémentations…

glVertex deviendra bientôt obsolète

```
for (i=0; i < nElem; ++i) {
   for (j=0; j < 3; ++j) {
      index = elem[3*i+j] - 1;

      x = X[index];
      y = Y[index];
      factor = (4*R*R + x*x + y*y)*(R/6);
      coord[j*3+0] = 4*R*R * x / factor;
      coord[j*3+1] = 4*R*R * y / factor;
      coord[j*3+2] = (4*R*R - x*x - y*y)*R / factor;  }

   glColor3f(0.0, 0.0, 0.0);
   glEnableClientState(GL_VERTEX_ARRAY);
   for (j=0; j < 9; ++j)
      coord[j] = coord[j] * 1.001;
   glVertexPointer(3, GL_FLOAT, 0, coord);
   glDrawArrays(GL_LINE_LOOP, 0, 3);
   glDisableClientState(GL_VERTEX_ARRAY); }
```

On dessine le maillage en l'extrudant légèrement pour éviter de le cacher derrière le plot d'élévation

Dessiner le maillage

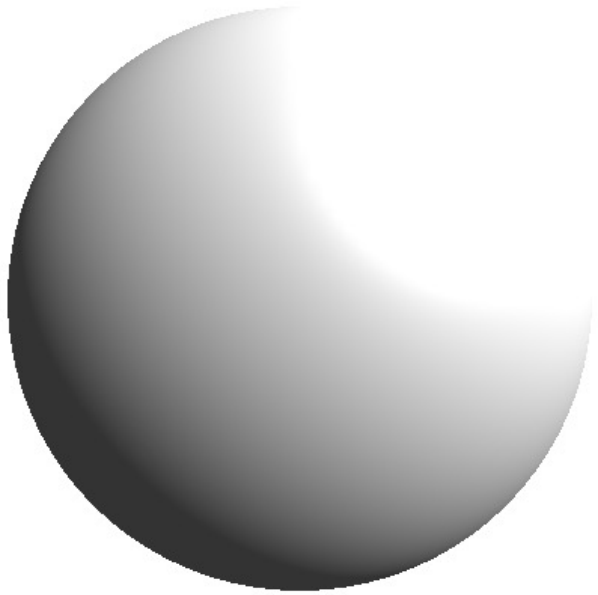


```
GLUquadricObj *quadratic = gluNewQuadric();
gluQuadricNormals(quadratic, GLU_SMOOTH);
glColor3f(1.0,1.0,1.0);
gluSphere(quadratic,5.95,400,200);
```

En réalité, on dessine un maillage de triangles dont le nombre de subdivisions en longitudes et latitudes est (400,200)

Eh oui, OpenGL finalement ne dessine que des triangles !

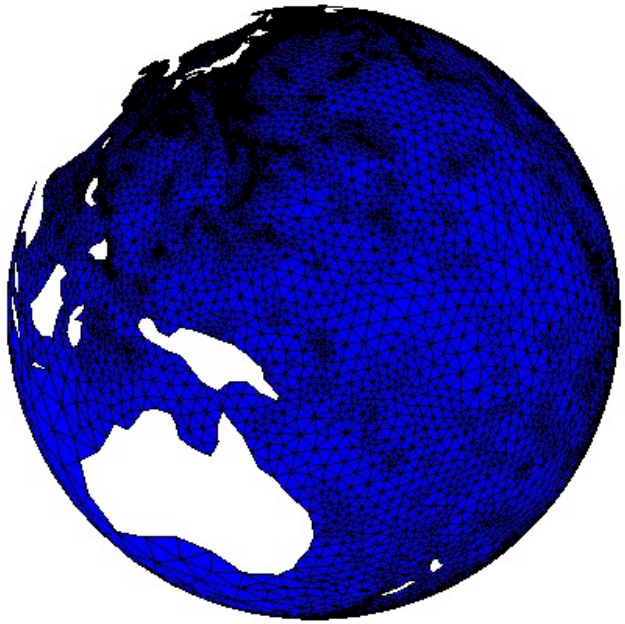
Dessiner la sphère



Eclairer la scène

```
GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 0.0 };
GLfloat mat_shininess[] = { 50.0 };
GLfloat light_position[] = { 8.0, 8.0, 8.0, 0.0 };

glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, GL_TRUE);
glLightfv(GL_LIGHT0, GL_POSITION, light_position);
GLfloat light_radiance[] = {1., 1., 1., 1.};

glLightfv(GL_LIGHT0, GL_DIFFUSE, light_radiance);
glLightfv(GL_LIGHT0, GL_SPECULAR, light_radiance);
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
glEnable(GL_COLOR_MATERIAL);
glEnable(GL_NORM
```

```
glEnableClientState(GL_VERTEX_ARRAY);
glEnableClientState(GL_COLOR_ARRAY);
glEnableClientState(GL_NORMAL_ARRAY);
glVertexPointer(3,GL_FLOAT,0,coord);
glColorPointer(3,GL_FLOAT,0,colors);
glNormalPointer(GL_FLOAT,0,coord);
glDrawArrays(GL_TRIANGLES,0,3);
```



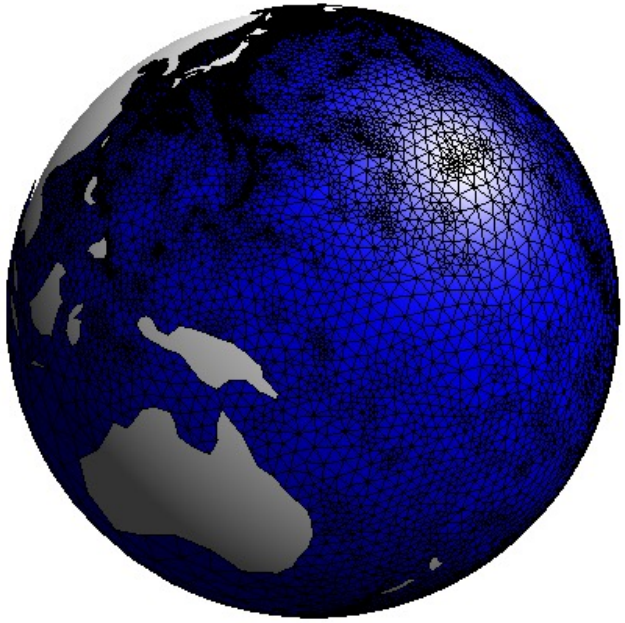
Il faut aussi définir les normales de la sphère (facile !)

Eclairer la scène

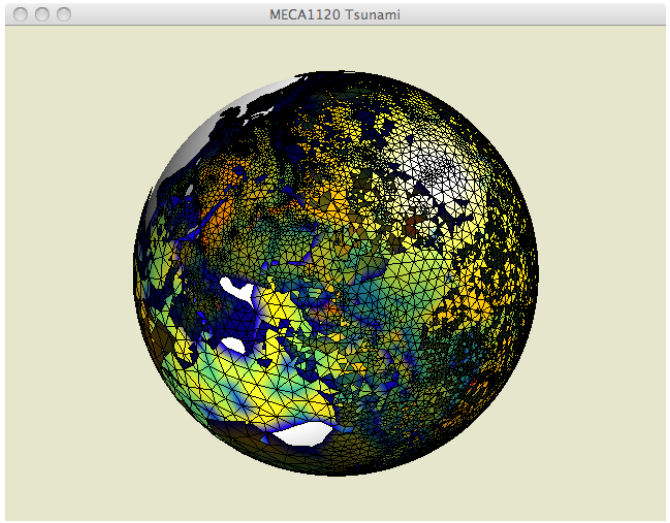
```
GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 0.0 };
GLfloat mat_shininess[] = { 50.0 };
GLfloat light_position[] = { 8.0, 8.0, 8.0, 0.0 };

glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, GL_TRUE);
glLightfv(GL_LIGHT0, GL_POSITION, light_position);
GLfloat light_radiance[] = {1., 1., 1., 1.};

glLightfv(GL_LIGHT0, GL_DIFFUSE, light_radiance);
glLightfv(GL_LIGHT0, GL_SPECULAR, light_radiance);
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
glEnable(GL_COLOR_MATERIAL);
glEnable(GL_NORMALIZE);
```

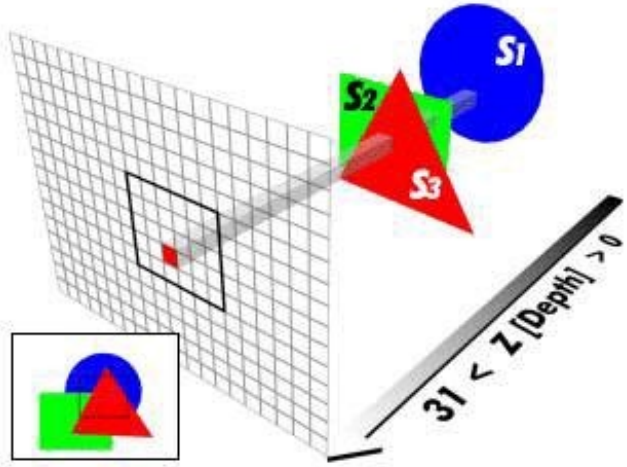
```
glEnableClientState(GL_VERTEX_ARRAY);
glEnableClientState(GL_COLOR_ARRAY);
glEnableClientState(GL_NORMAL_ARRAY);
glVertexPointer(3,GL_FLOAT,0,coord);
glColorPointer(3,GL_FLOAT,0,colors);
glNormalPointer(GL_FLOAT,0,coord);
glDrawArrays(GL_TRIANGLES,0,3);
```

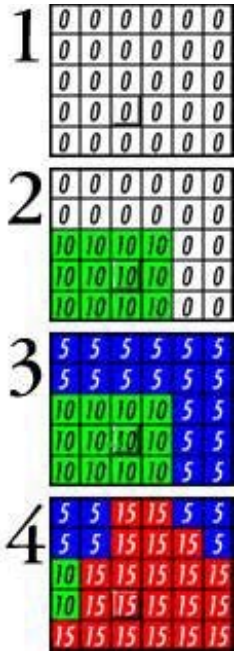
Il faut aussi définir les normales de la sphère (facile !)



```
glfwOpenWindow(640,480,0,0,0,0,1,0,GLFW_WINDOW );

glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

glDepthFunc(GL_LEQUAL);
glEnable(GL_DEPTH_TEST);
```

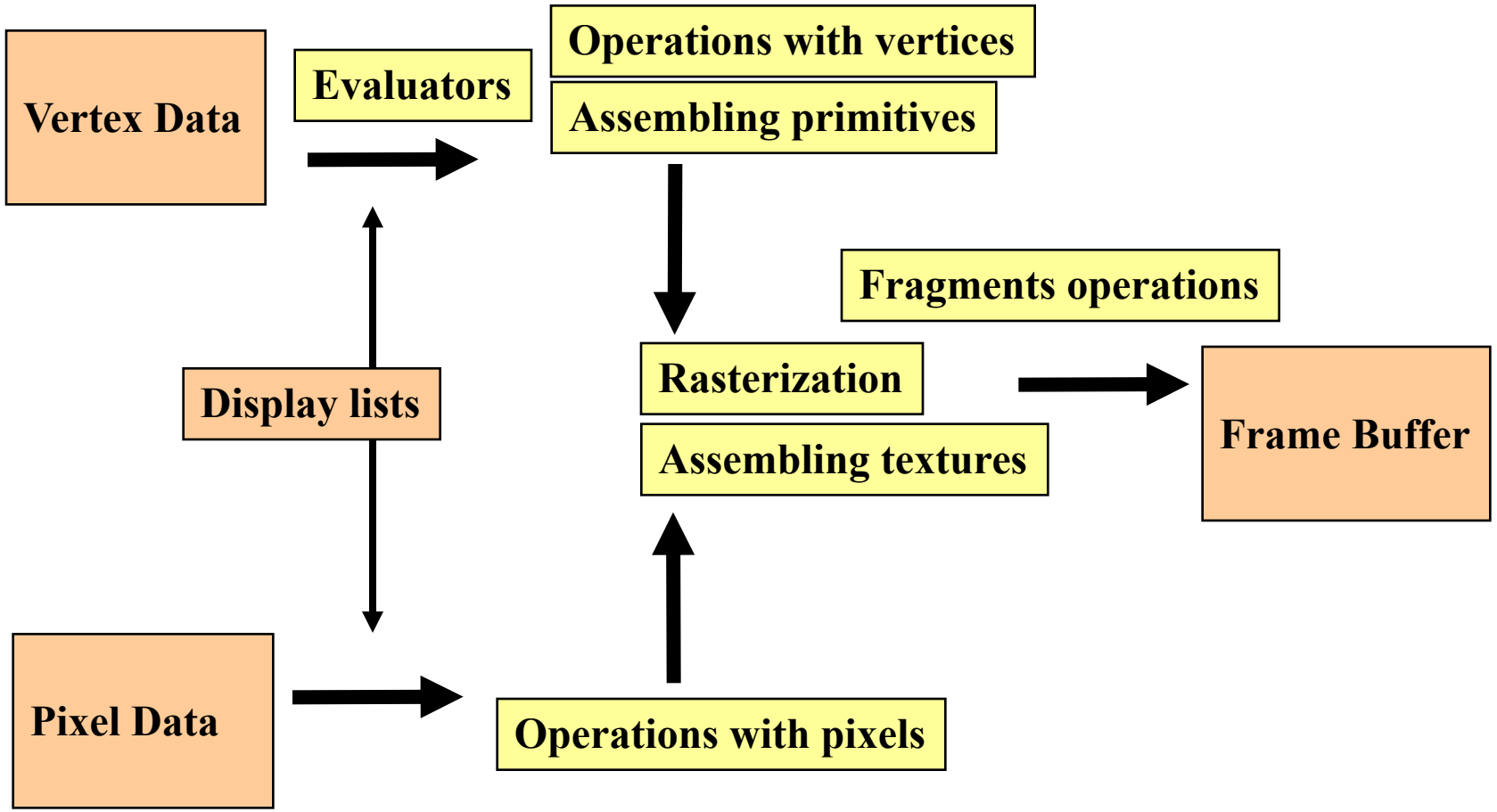


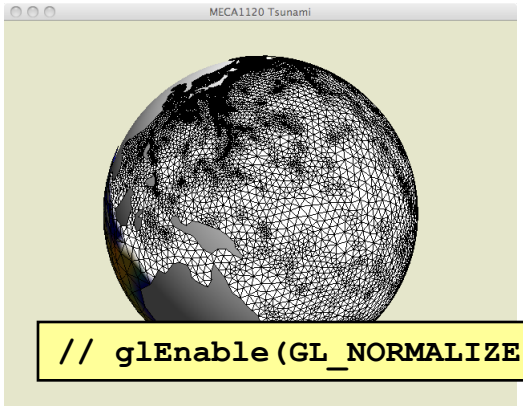




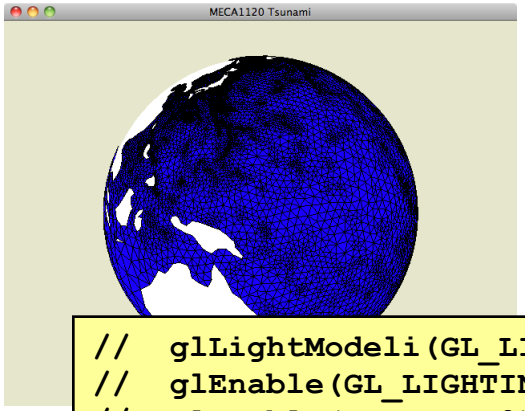
Z-buffer algorithm

OpenGL's rendering pipeline

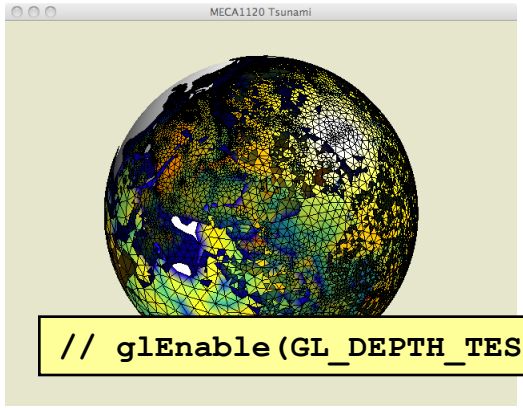





```
// glEnable(GL_NORMALIZE);
```




```
//  glLightModeli(GL_LIGHT_MODEL_TWO_SIDE,GL_TRUE);
//  glEnable(GL_LIGHTING);
//  glEnable(GL_LIGHT0);
```




```
// glEnable(GL_DEPTH_TEST);
```

Les éléments finis,
c'est fini pour 2022 !