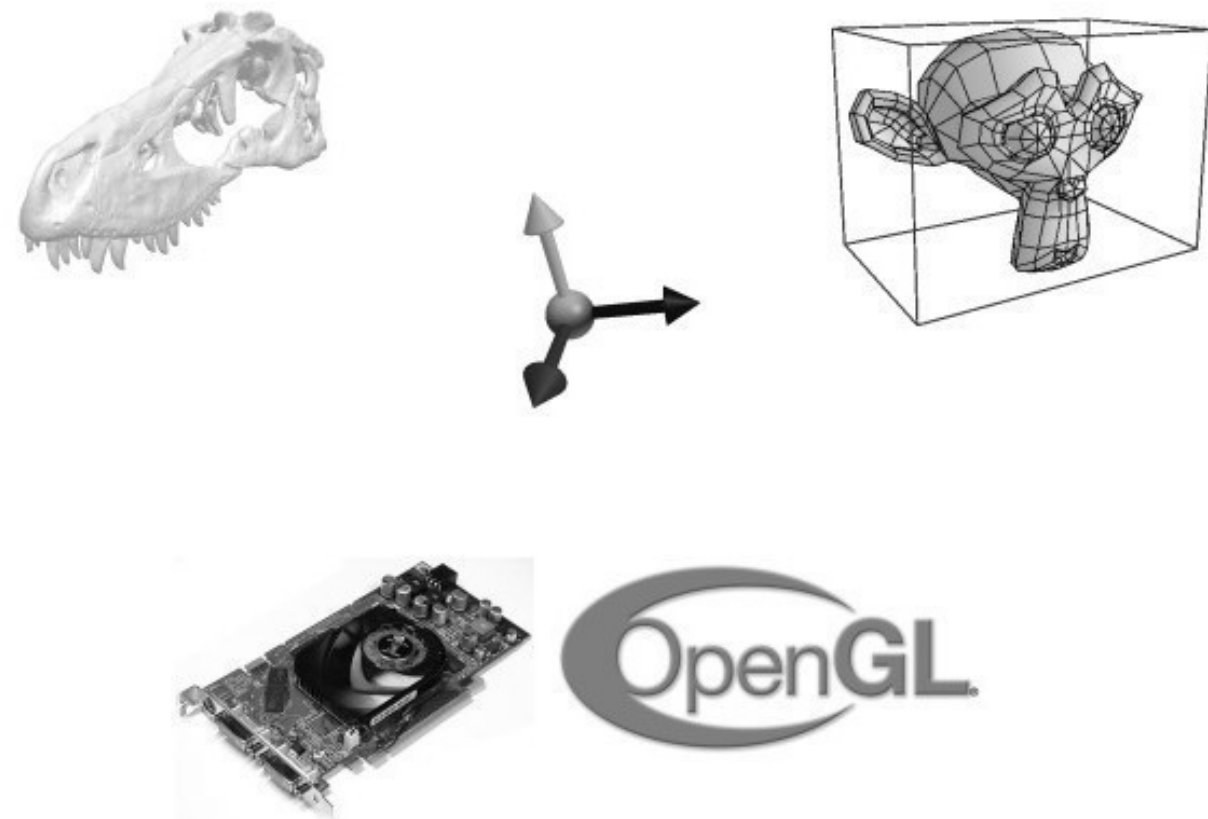
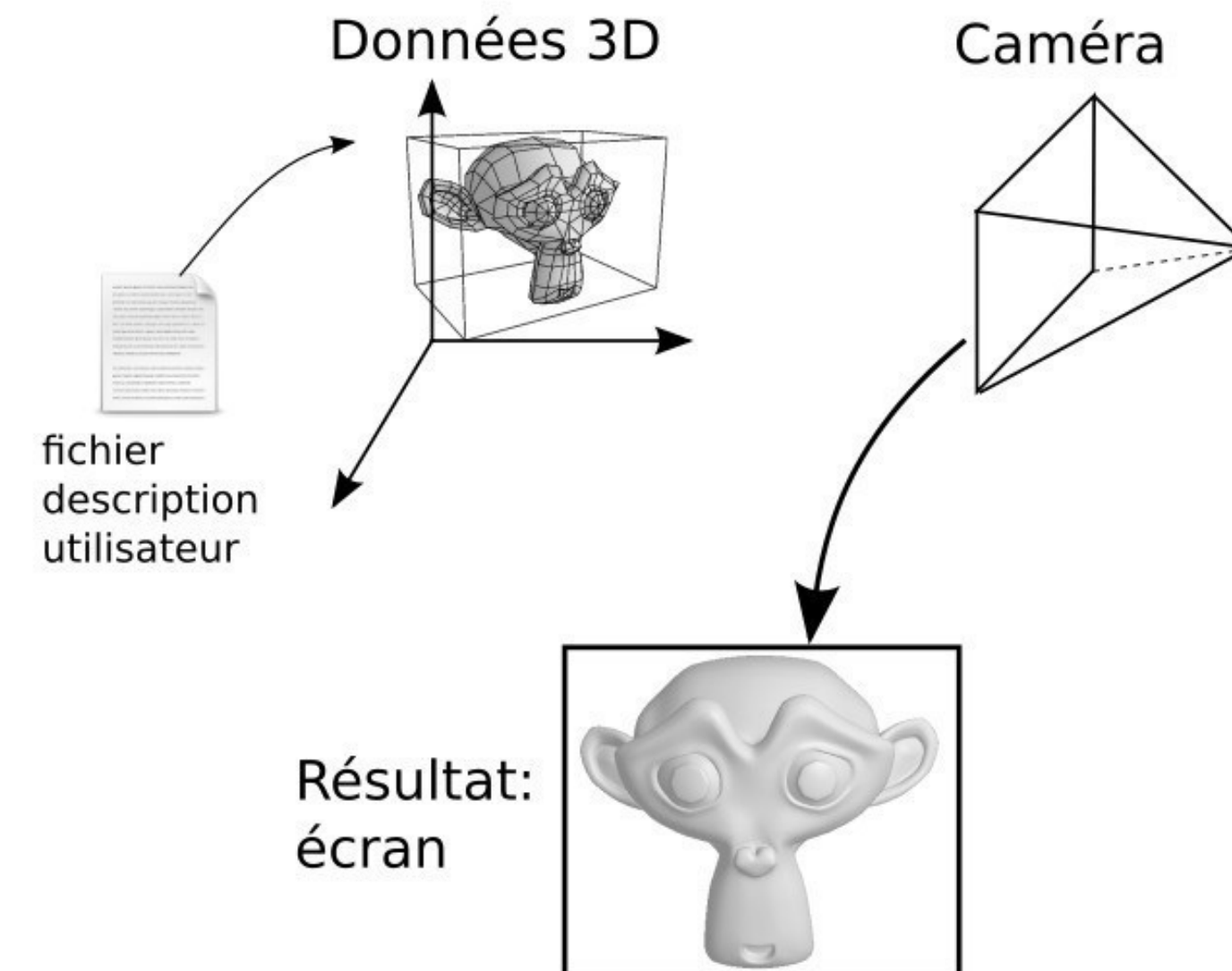


Synthèse d'images



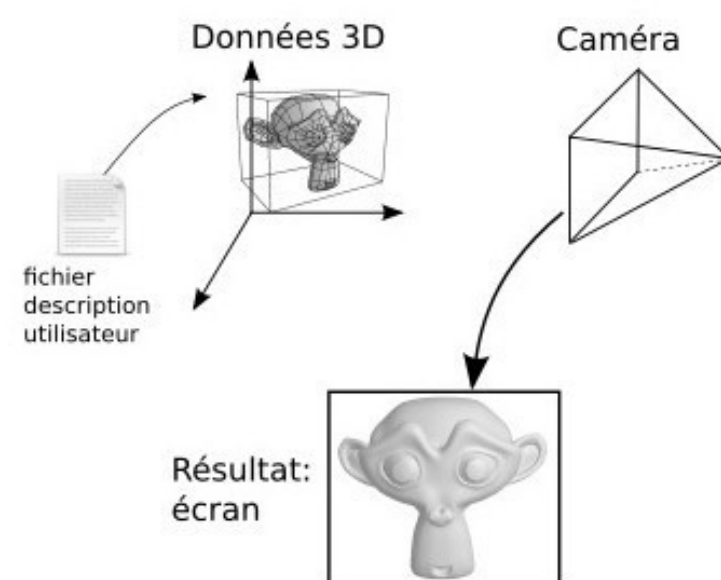
000

Affichage 3D



001

Affichage 3D



Ordre de grandeurs:

Objets 3D:
100 - 10⁵ triangles

1 Triangle ~3000 pixels

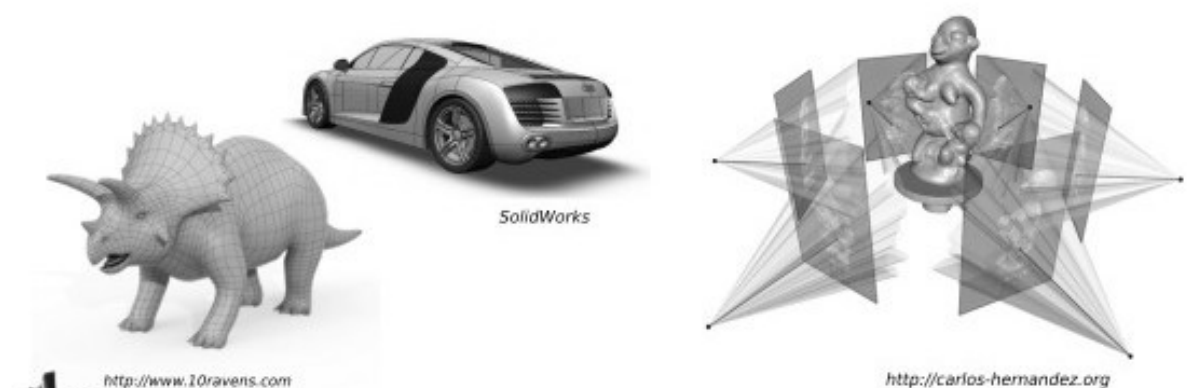
25 images par secondes

Affichage 3D = calcul intensif

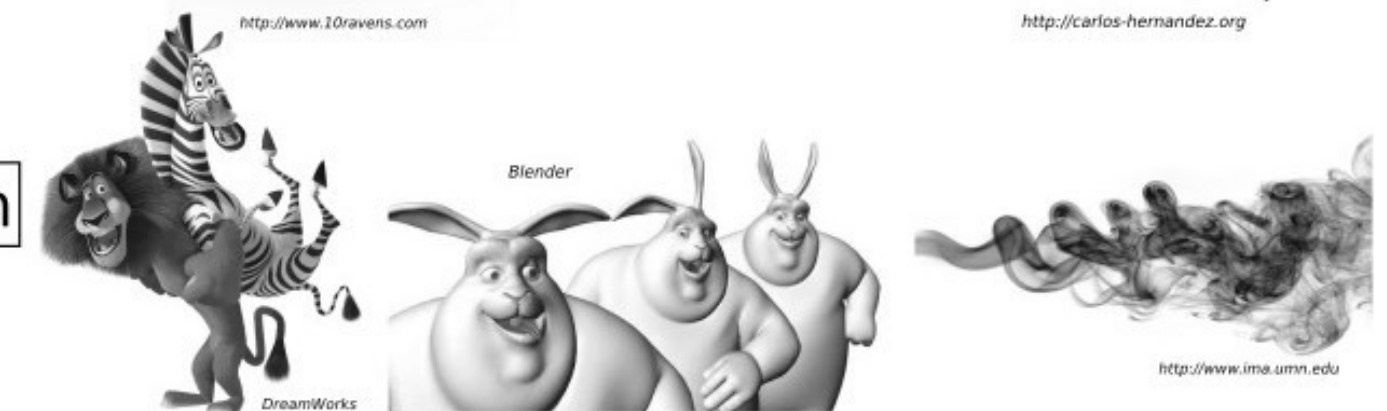
002

Les domaines de la 3D

Modélisation



Animation



Rendu

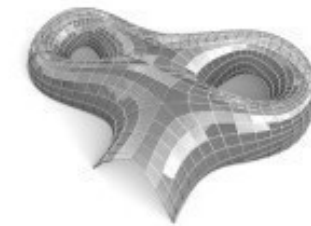


003

Les applications de la 3D

Domaine emergent

- Loisir (Jeux-Vidéo, Cinéma, ...)
- Simulation, Calcul, Analyse données
- Design, prototypage, reconstruction
- Interaction (RV, Réalité augmentée, ...)



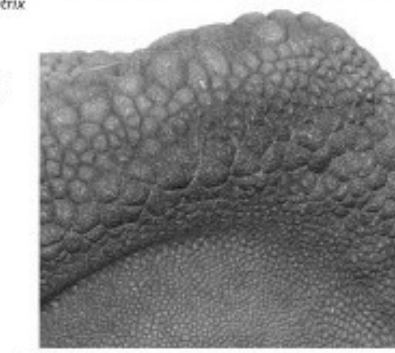
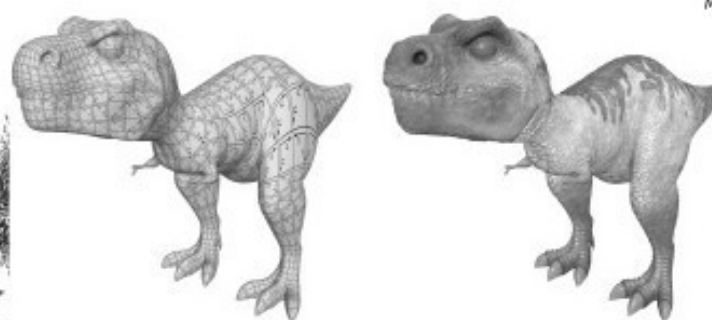
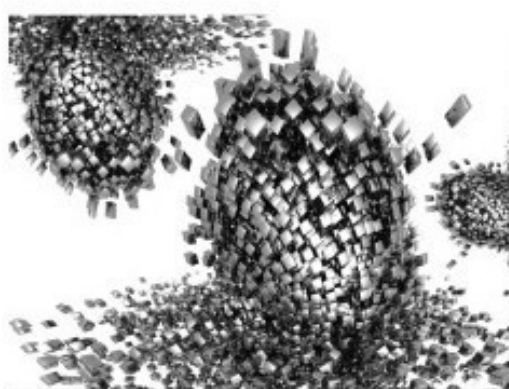
004

Loisir: Jeux Vidéos



005

Loisir: Cinéma



006

Loisirs

Types de métiers

Développement
C++
Rendu/Shader
GPU
Scripting
Automatisation
Pipe-line création



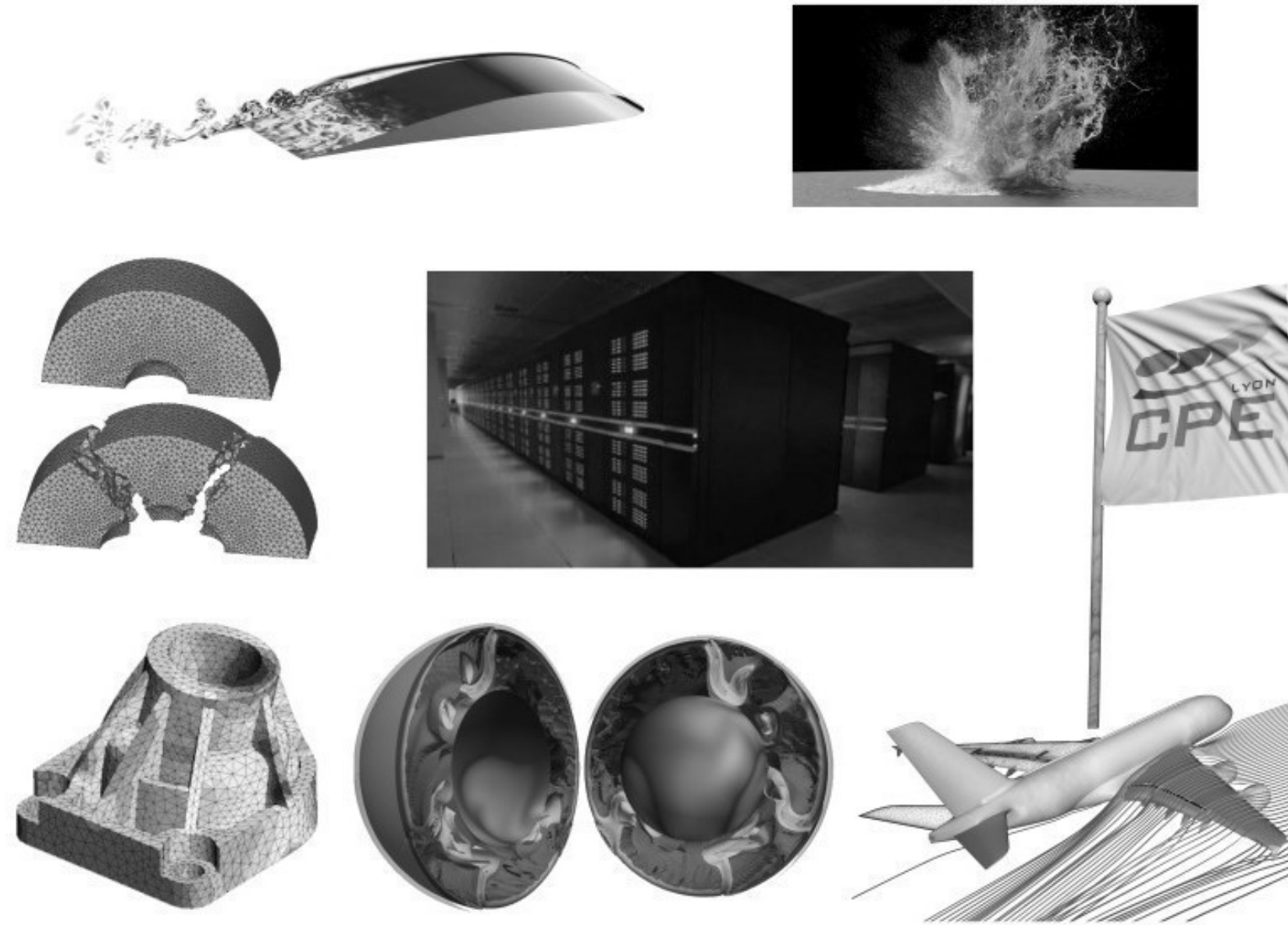
Stages/Anciens CPE

Gestion occulsion
(GameLoft)
Dev. Android
(AppSolute)
Gestion de version, évolution
(Asobo)
Personnage non joueurs
(Ubisoft)
Dev. The Crew
(Ivory Tower)



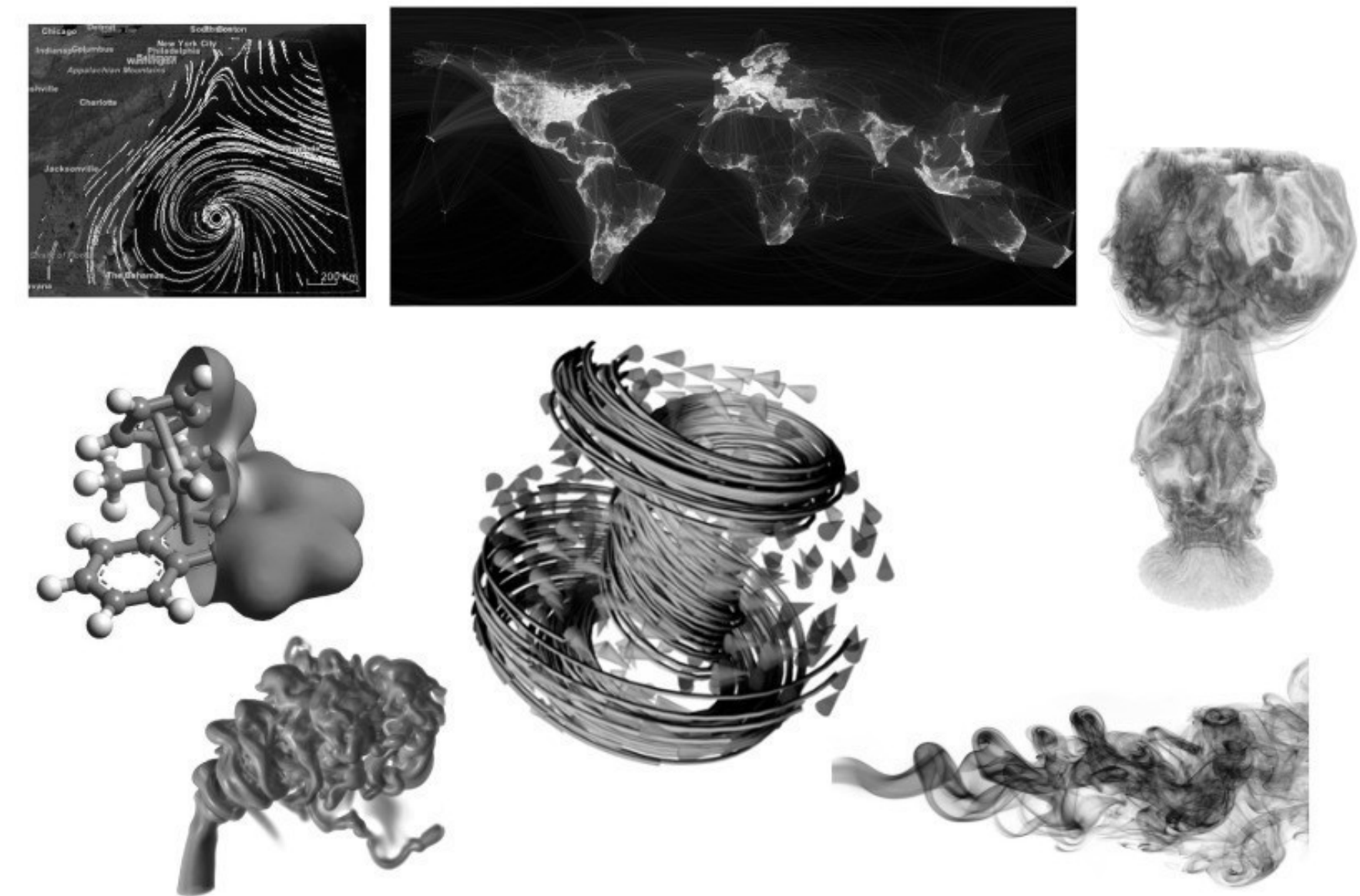
007

Simulation: calcul numérique



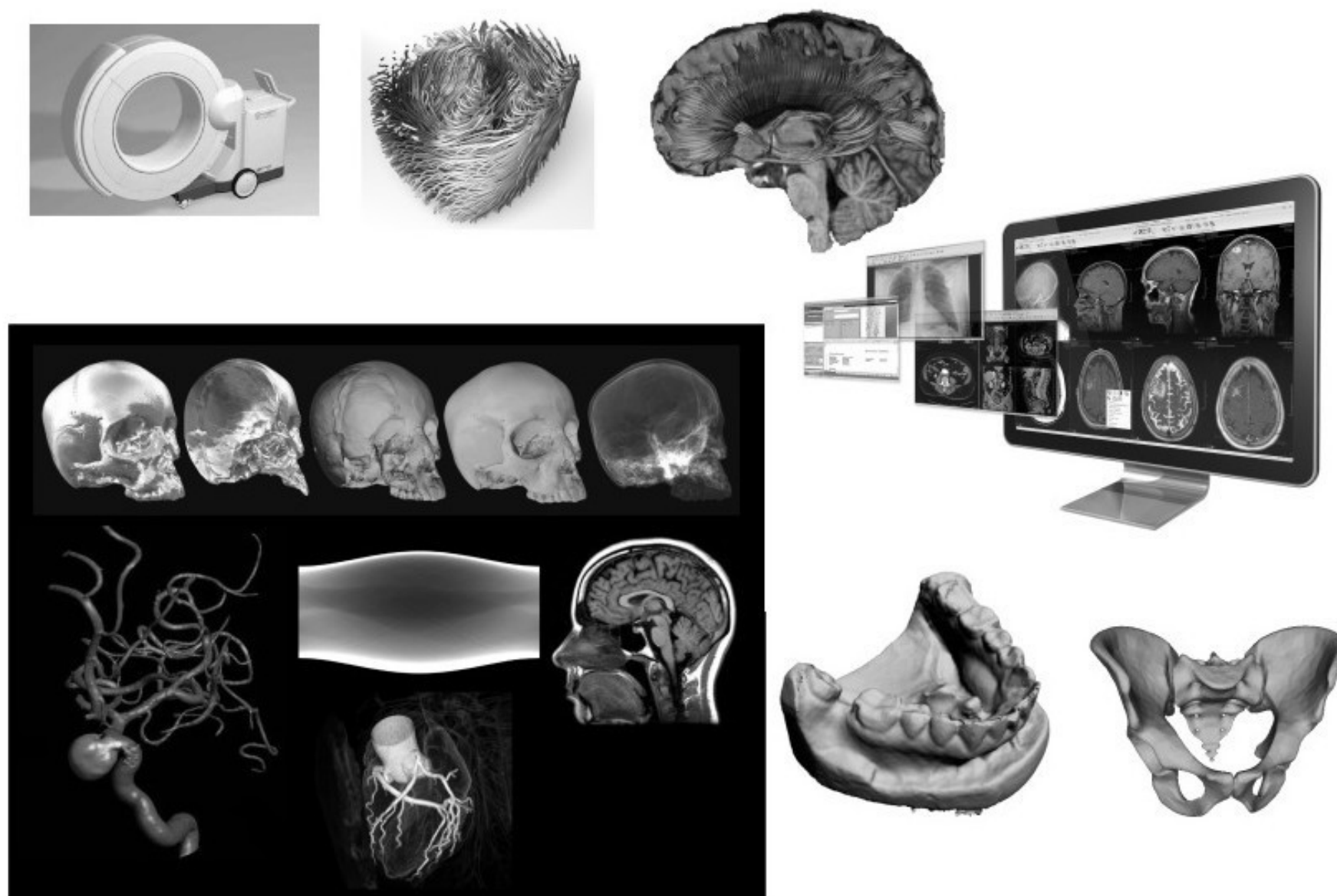
008

Simulation: Visu scientifique



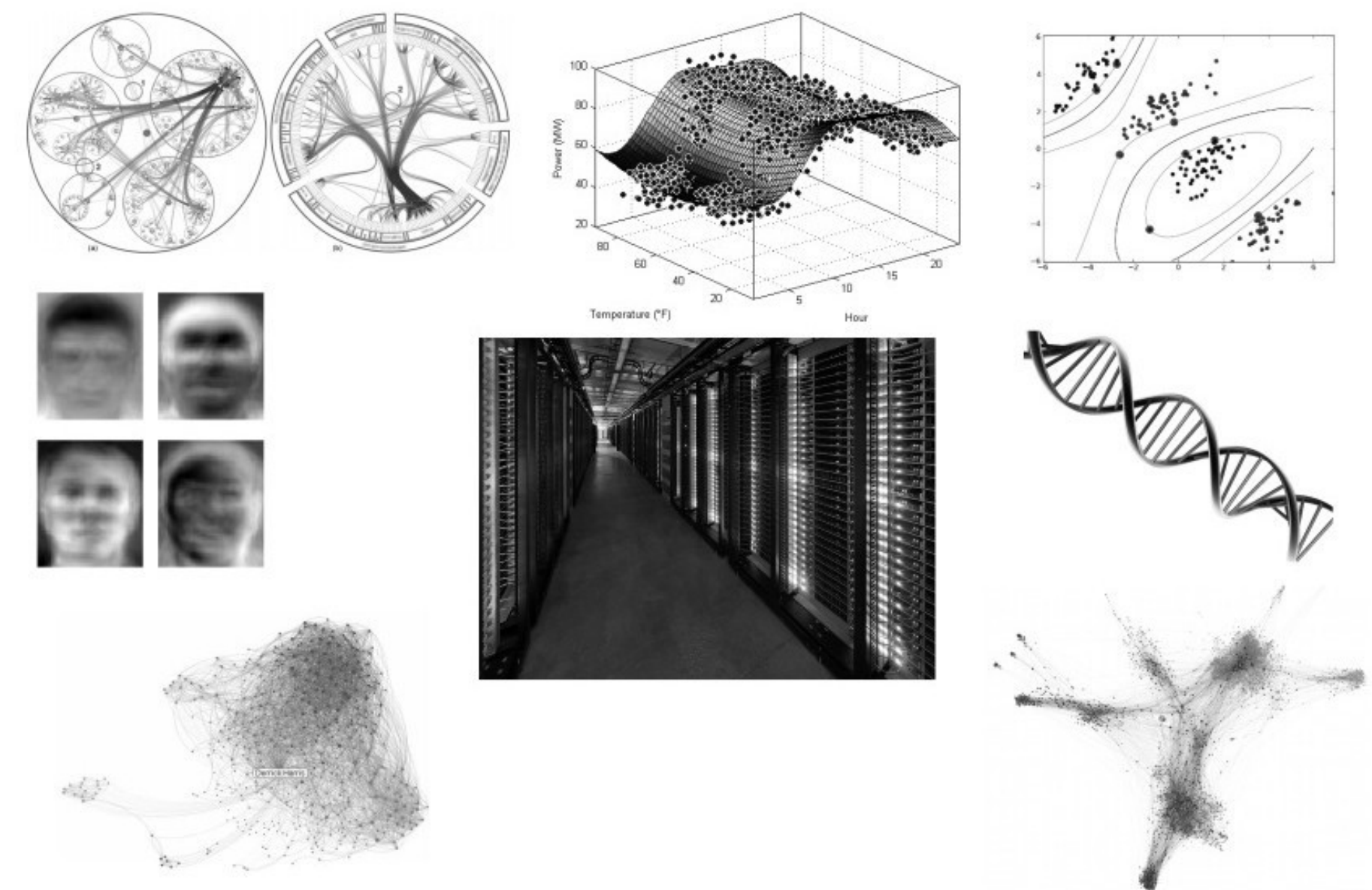
009

Simulation: Imagerie médicale



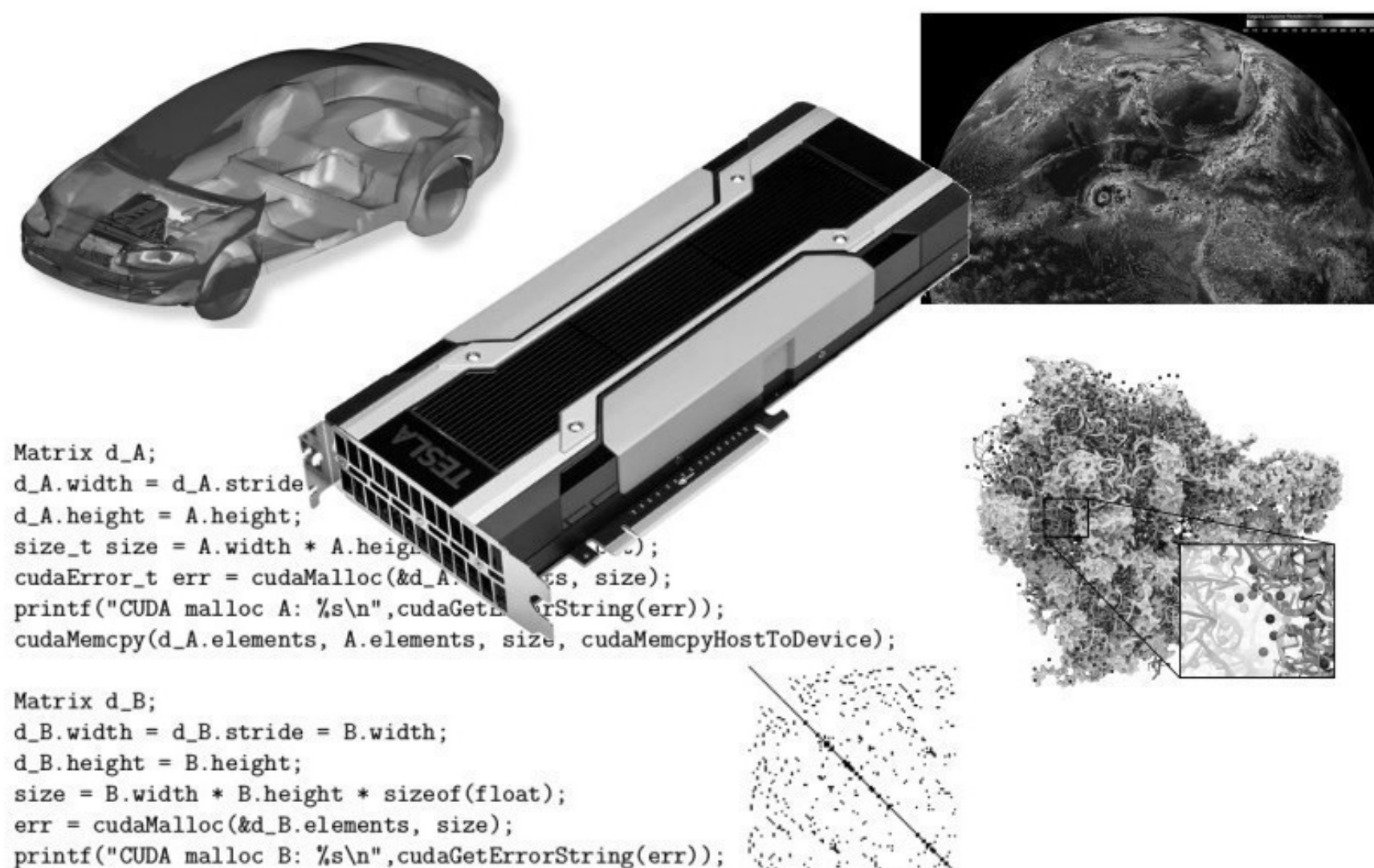
010

Analyse de données: Big Data



011

Analyse de données: GPGPU



012

Simulation, Calcul, Analyse de données

Types de métiers

Calcul numérique
Domaine physique/médicale
Analyse de données
Programmation efficace
Visualisation scientifique
(champs scalaires/vecteurs, ...)
Grands jeux de données
GPU



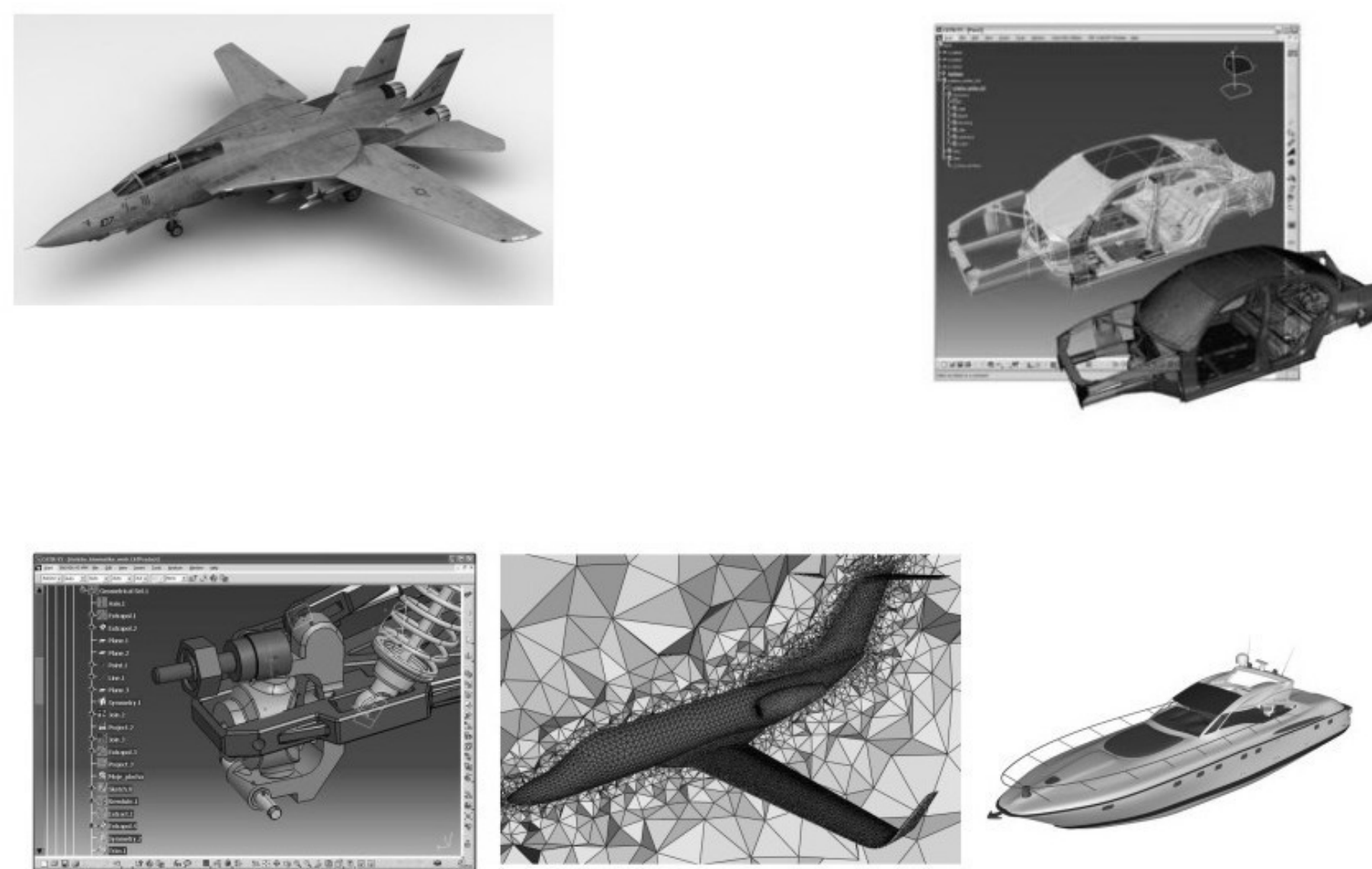
Stages/Anciens CPE

Imagerie du cerveau
(*Scient. Comp. II*)
Visu. dents 3D
(*Michigan Univ.*)
Visu 3D.
(*GE Healthcare*)
Recallage
(*Michelin*)
Data analysis
(*L'Oreal*)
Simulation cheveux
(*LIRIS*)



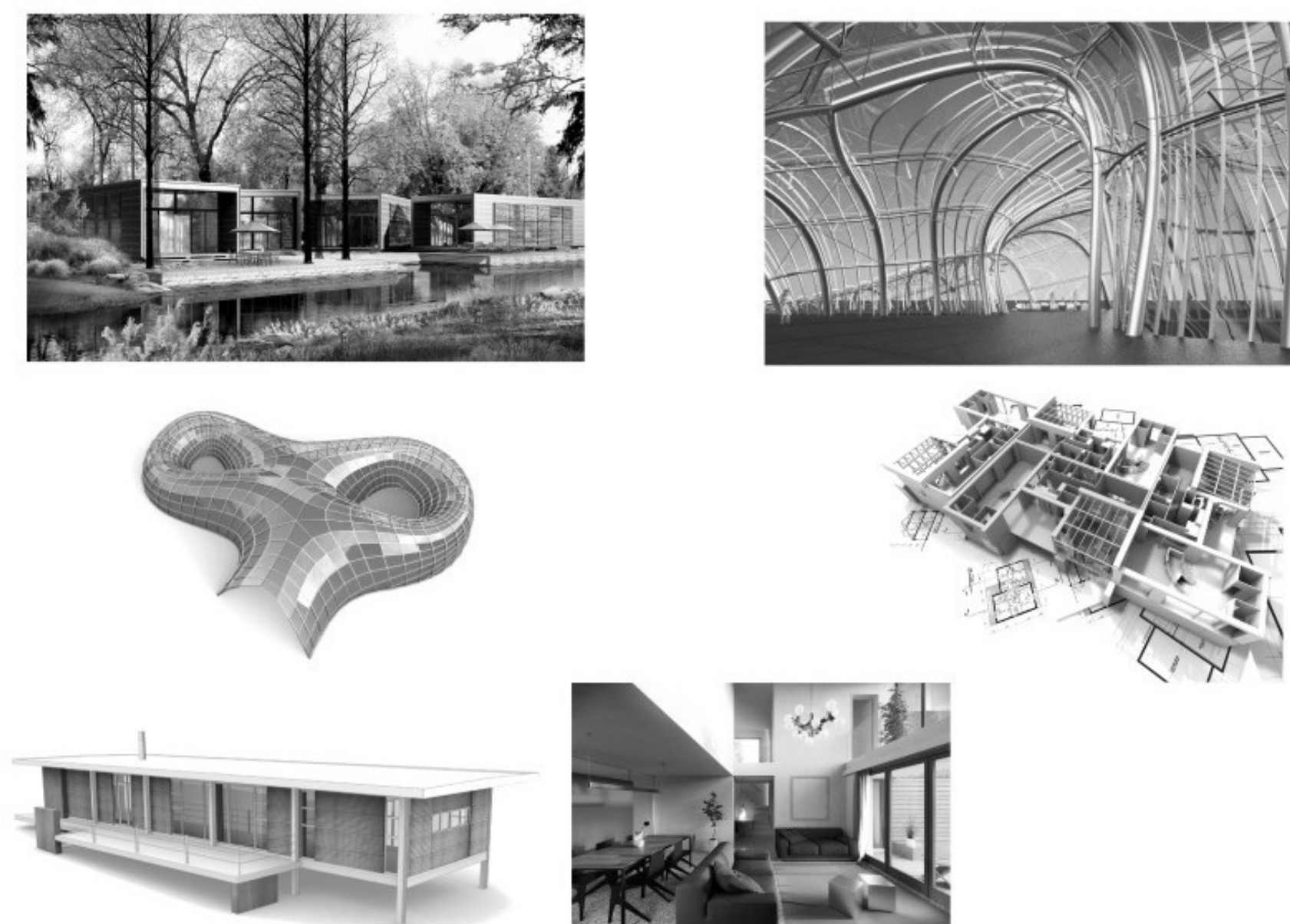
013

Design: CAO



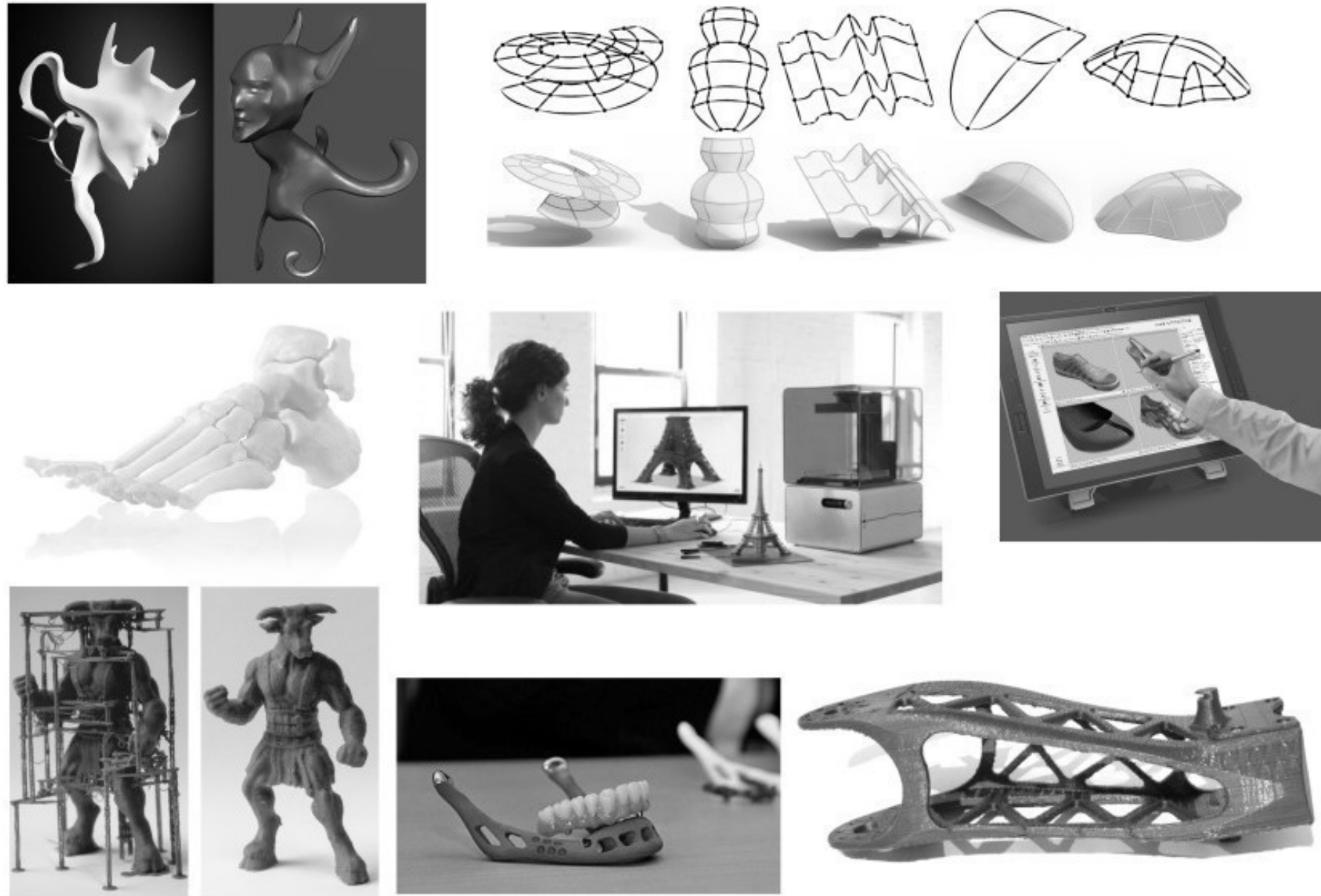
014

Design: Architecture



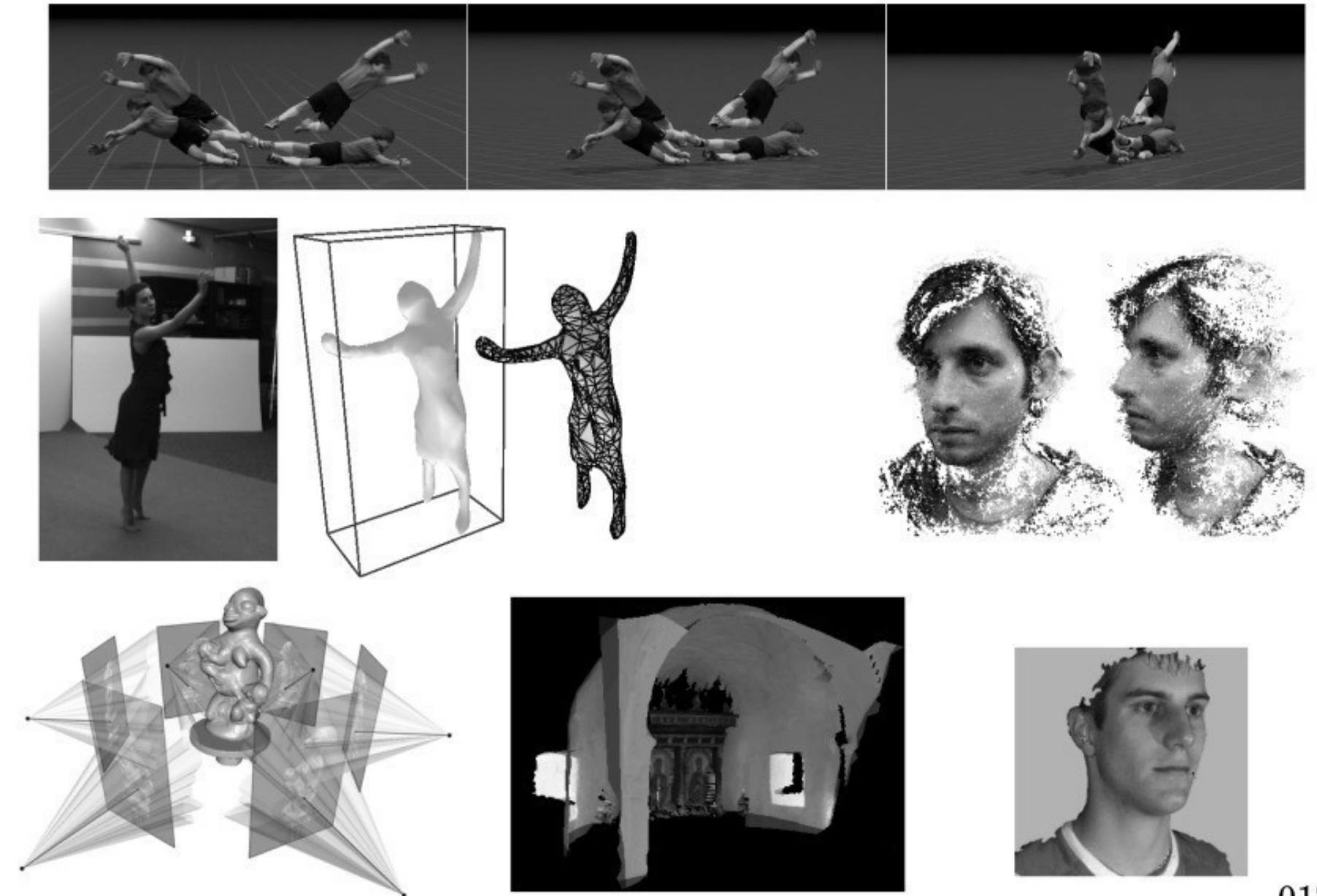
015

Design: Prototypage



016

Design: Reconstruction 3D



017

Design: Patrimoine

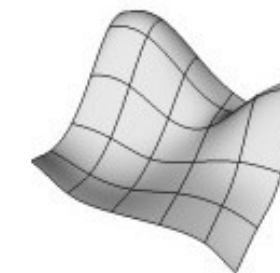


018

Design

Types de métiers

- Maillage
- Capteurs
- Reconstruction
- Interaction, IHM
- Mecanique
- Courbes & Surfaces



Stages/Anciens CPE

- Reconstruction 3D
(Kagoshima Univ.)
- Sculpture virtuelle
(LIRIS)
- Scène 3D
(Technodigit)
- Capteur vidéo 3D
(Thales)
- Gestion collisions
(Delcam)
- Modélisation 3D
(Rauscher)
- Vision lunette 3D
(Rodenstock)



019

Interaction: Réalité virtuelle (VR)



020

Interaction: Réalité augmentée



021

Interaction

Types de métiers

Reconstruction 3D
Simulation rapide
IHM
Capteurs
Dev. smartphone



Stages/Anciens CPE

Réalité augmentée médicale
(*Kitware*)
Réalité augmentée ville
(*Wikitude*)
Logiciel d'esquisse
(*Dassault*)
Ville virtuelle
(*LIRIS*)
A350 air system modeling
(*Airbus*)
Réalité augmentée Android
(*CEA*)



022

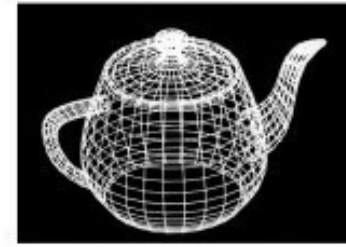
Historique

023

Historique

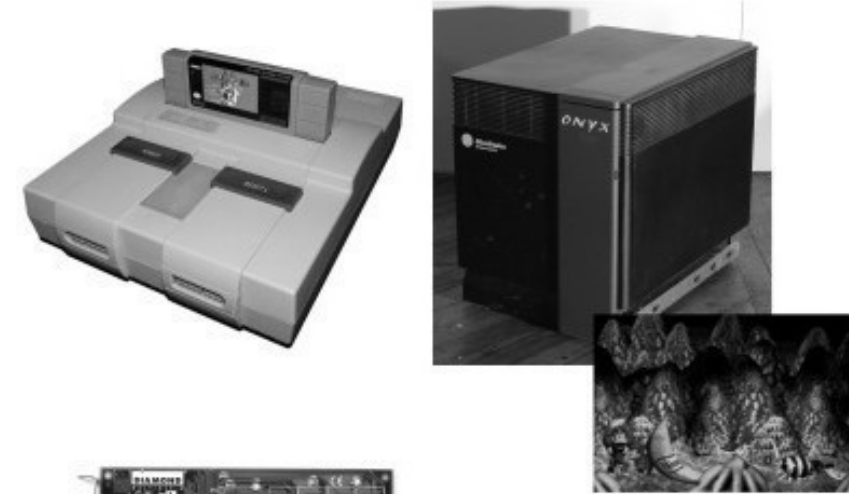
1ere 3D (années 70-80)

Supercalculateurs



80-90

*Architecture dédiée
(3D "cablée")*



Fin 90

Cartes vidéo 3D



024

Historique

Début 2000 Apparition des cartes graphiques

Programmation 3D possible sur un PC standard
Langage proche de l'assembleur



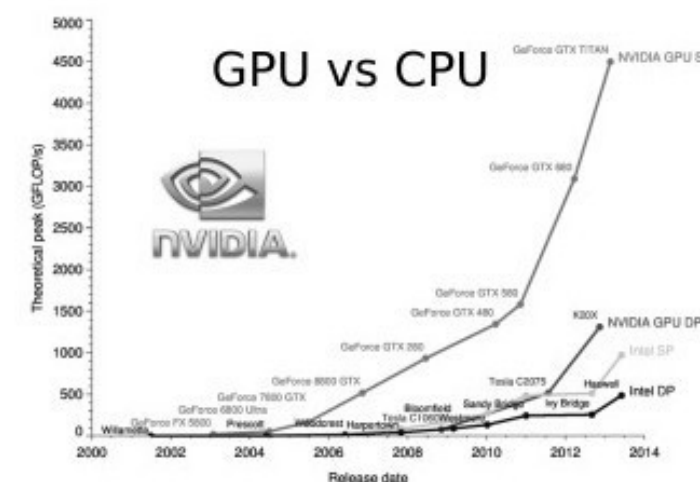
025

Historique

2014

Language de
programmation
parallèle.

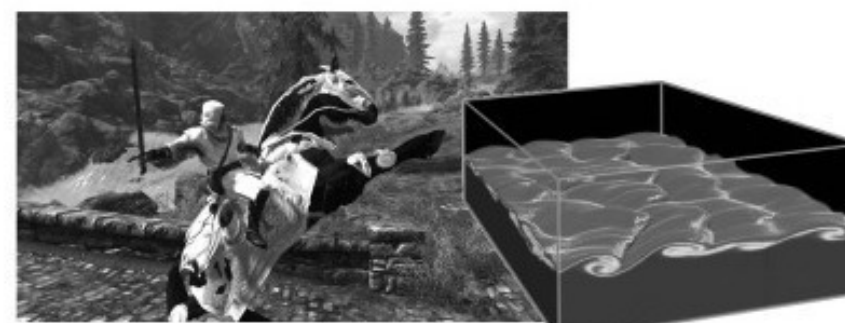
GPGPU



<http://michaelgalloy.com/>



Crytek Engine



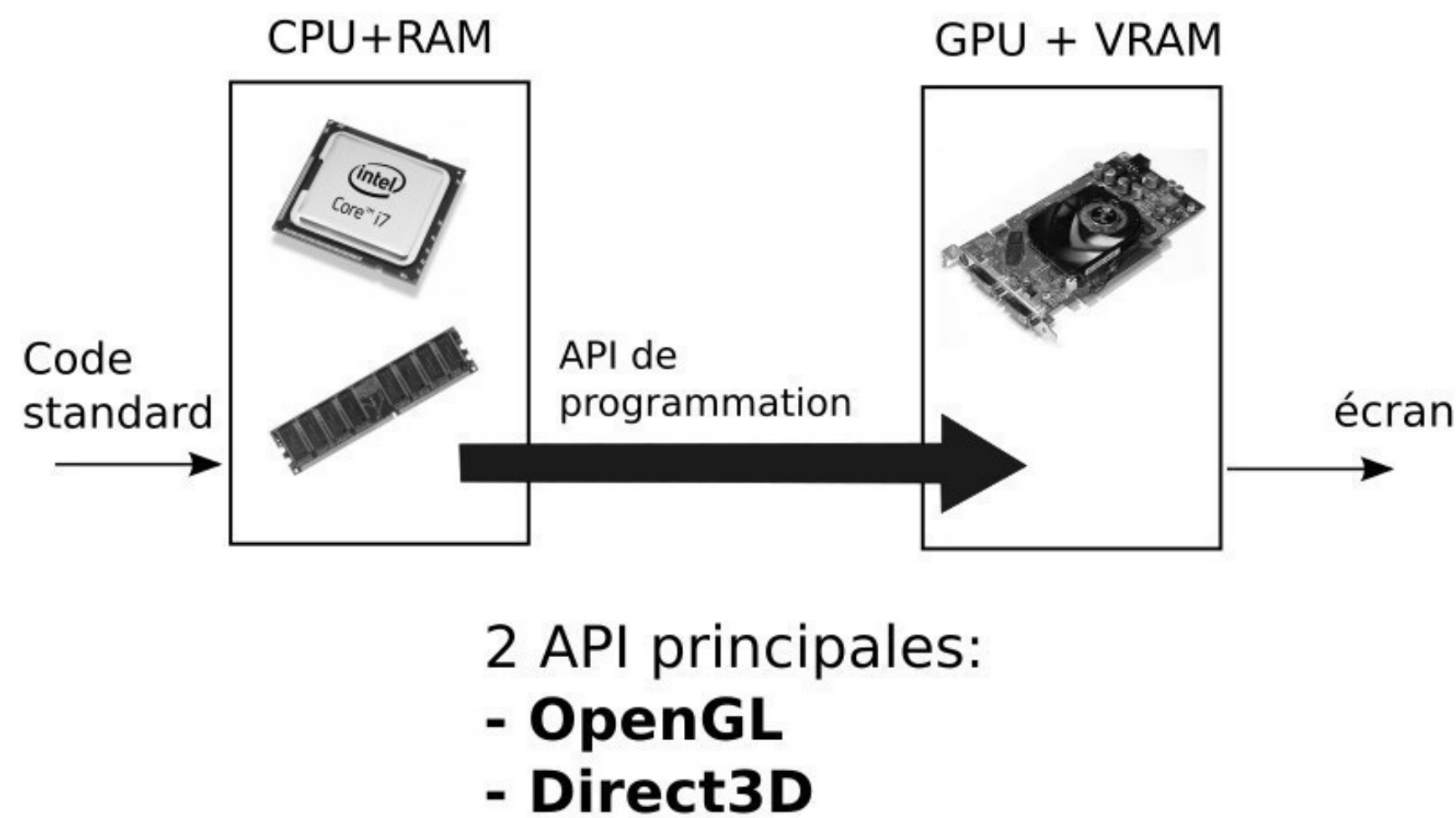
Skyrim

026

OpenGL

027

Communication avec GPU



028

OpenGL

- Un ensemble de **spécifications** (cahier des charges) sous forme d'API pour communiquer avec la carte graphique

- Possède plusieurs implémentations suivant la carte graphique et les drivers

- Version revendeur: NVIDIA, ATI, Intel
- Mesa3D (implémentation logicielle libre)
- OpenGL ES pour Android / iPhone
- Consoles: Wii, PS3, DS

- Bibliothèque de fonctions C
Nombreux wrappers



Sources: David Odin

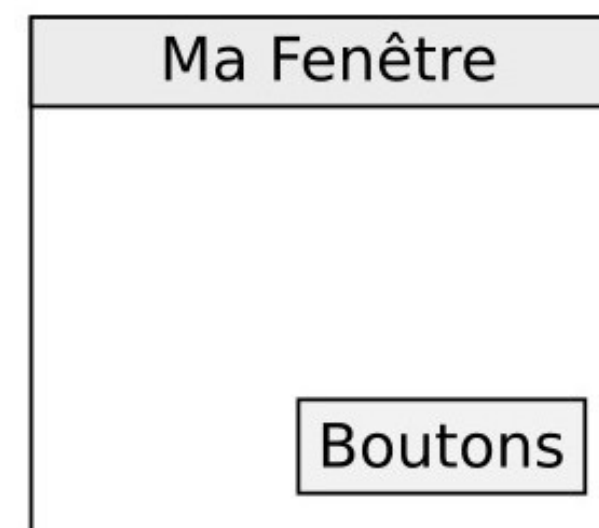
029

Programmation 3D

Etape 1:
Afficher une fenêtre
(indépendant d'OpenGL)

Etape 2:
Démarrer un contexte
OpenGL dans cette fenêtre

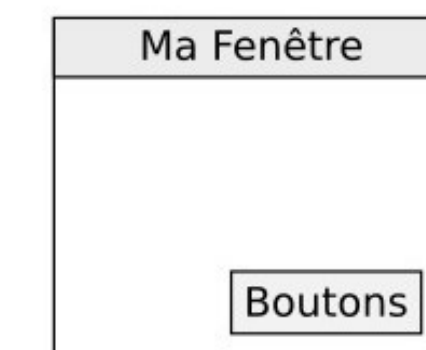
Etape 3:
Appeler le code OpenGL
dans une boucle permanente



030

Programmation 3D

Etape 1:
Afficher une fenêtre
(indépendant d'OpenGL)



Il faut donc un gestionnaire de fenêtres & evenements

Plusieurs choix:

- Glut (simple, léger, limité)
- SDL (spécialisé pour les jeux)
- GLFW (léger, récent)
- GTK (gestionnaire générique fenêtre & événements)
- Qt (gestionnaire générique, très complet, lourd)
- ...

031

Utilisation de GLUT

GLUT - The OpenGL Utility Toolkit

Un gestionnaire de fenêtre dédié pour OpenGL

```
//fonction d'affichage
static void display_callback()
{
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv); //initialise glut
    glutInitDisplayMode(GLUT_DOUBLE |
        GLUT_RGB |
        GLUT_DEPTH); //mode d'affichage
    glutInitWindowSize(800, 800); //taille de la fenetre
    glutCreateWindow("Ma Fenetre"); //creation de la fenetre
    glutDisplayFunc(display_callback); //affichage dans la fenetre
    glewInit(); //initialisation des fonctions de glew
    glutMainLoop(); //boucle permanente

    return 0;
}
```

Pour compiler: g++ pgm.c -lGL -lGLU -lglut -lGLEW

Sous Linux: Installer les packages de **freeglut**, **Glew**

032

Fonction d'affichage

Pour afficher quelque chose

```
static void display_callback()
{
    glClearColor(0.5, 0.6, 0.9, 1.0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glutSwapBuffers();
}
```

glNAME : Fonction OpenGL

glClearColor(r,g,b,a) : Couleur d'effacement de l'écran

glClear() : Efface l'écran (+ tampon profondeur)

glutSwapBuffers() : Echange tampon/affichage

↖ fonction liée à glut

033

Callback GLUT

```
static void display_callback()
{ ... }
static void keyboard_callback(unsigned char key,
    int x_mouse, int y_mouse)
{
    printf("key %c with mouse at position (%d,%d) \n",
        key, x_mouse, y_mouse);

    if(key=='q')
    {
        puts("Goodbye");
        exit(0);
    }
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv); //initialise glut
    glutInitDisplayMode(GLUT_DOUBLE |
        GLUT_RGB |
        GLUT_DEPTH); //mode d'affichage
    glutInitWindowSize(800, 800); //taille de la fenetre
    glutCreateWindow("Ma Fenetre"); //creation de la fenetre

    glutDisplayFunc(display_callback);
    glutKeyboardFunc(keyboard_callback); → Fonction à appeler

    glewInit(); //initialisation des fonctions de glew
    glutMainLoop(); //boucle permanente

    return 0;
}
```

034

Callback GLUT (mouse)

Récupération des informations du click souris

```
static void mouse_click_callback(int button, int state,
    int x, int y)
{
    printf("mouse click %d,%d , (x,y)=(%d,%d)\n", button, state, x, y);
}
```

```
glutMouseFunc(mouse_click_callback);
```

Evènements par "callback" = fonction utilisateur appelée lors d'un évènement

035

Callback GLUT

Autres évènements possibles:

glut ReshapeFunc	(redimensionnement de fenêtre)
glut KeyboardFunc	(appui clavier)
glut SpecialFunc	(touches spéciales du clavier: flèches)
glut MouseFunc	(clic souris)
glut MotionFunc	(déplacement souris)
glut TabletButtonFunc	(appui bouton tablette)
glut IdleFunc	(lorsque rien ne se passe)
glut TimerFunc	(appel au bout d'un temps donné)

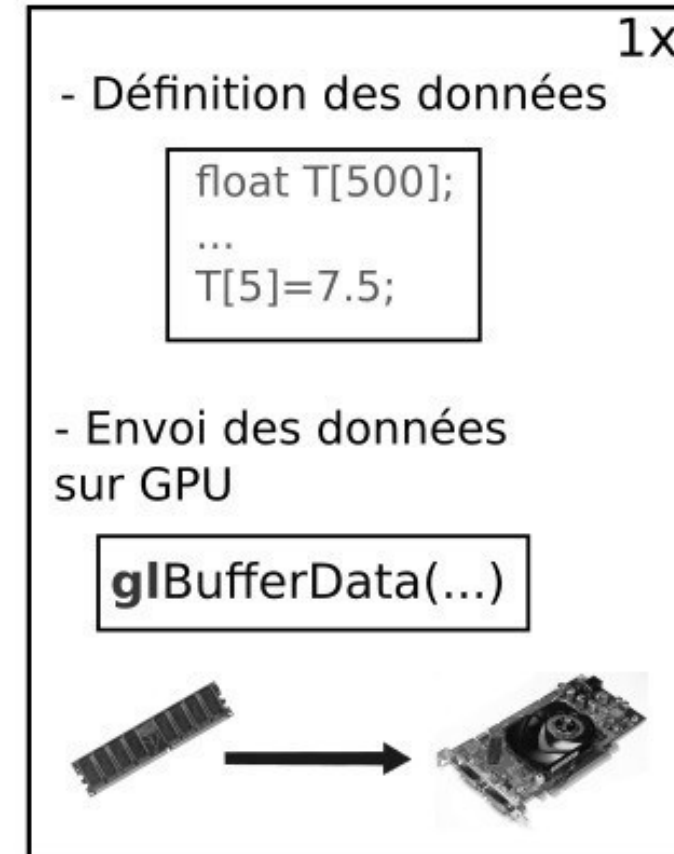
<http://www.opengl.org/resources/libraries/glut/spec3/spec3.html>

036

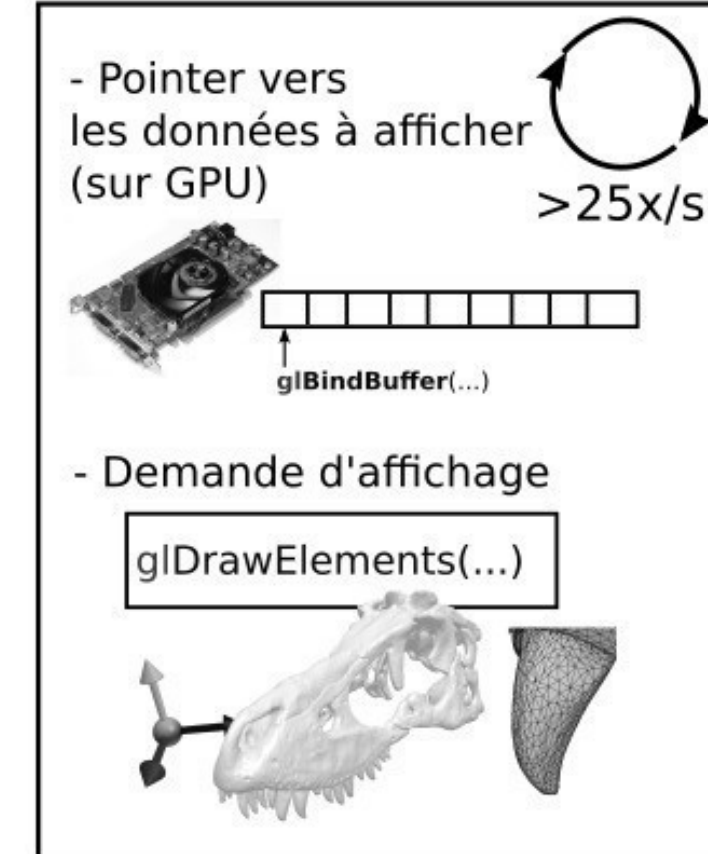
Appels OpenGL

Principe général

Initialisation



Boucle d'affichage



037

Programme minimaliste Affichage de triangle

Pgm minimal, triangle

```
//un identifiant de buffer
GLuint vbo=0;

//fonction d'affichage
static void display_callback()
{...}

//fonction d'initialisation
void init() {...}

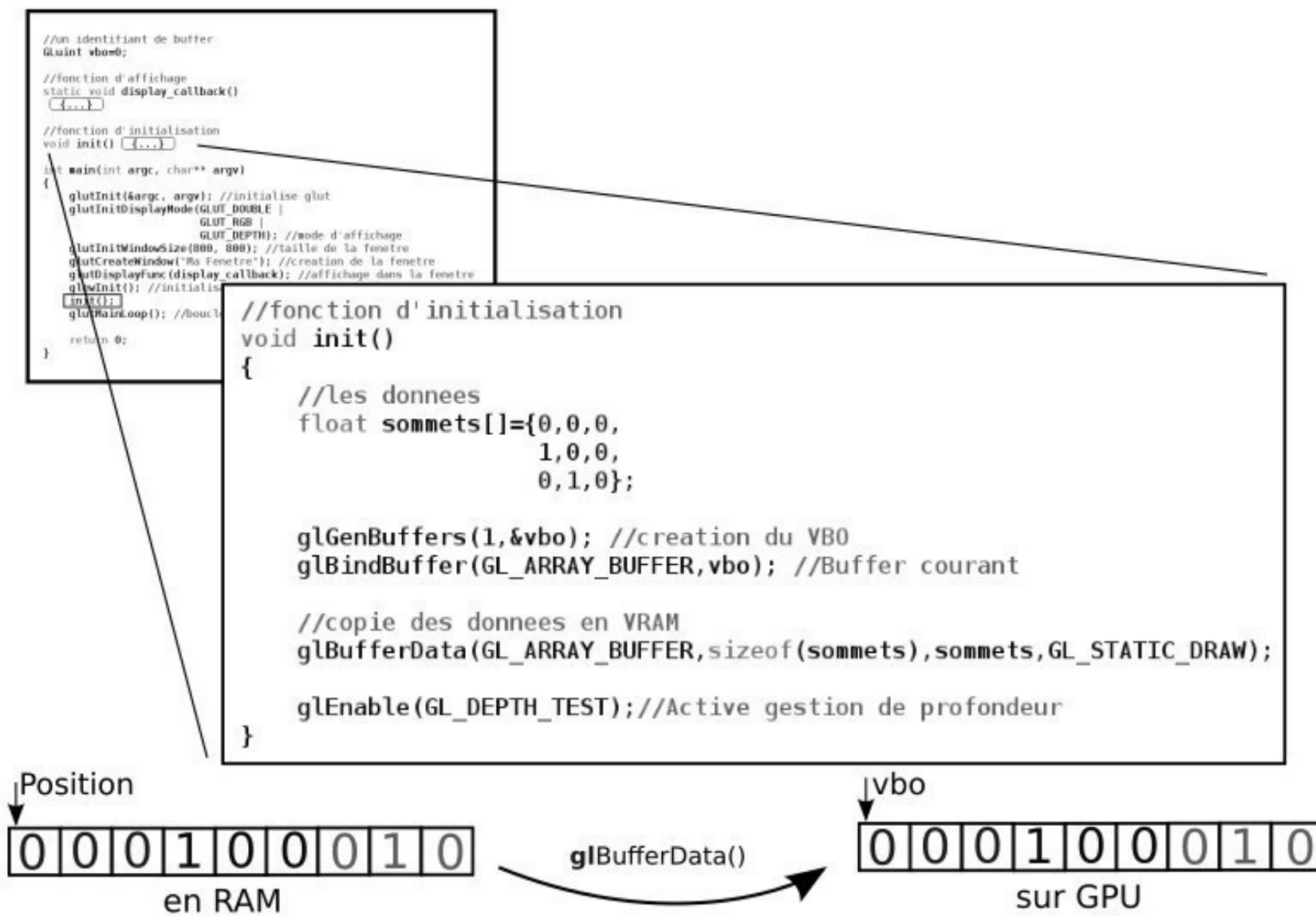
int main(int argc, char** argv)
{
    glutInit(&argc, argv); //initialise glut
    glutInitDisplayMode(GLUT_DOUBLE |
                        GLUT_RGB |
                        GLUT_DEPTH); //mode d'affichage
    glutInitWindowSize(800, 800); //taille de la fenetre
    glutCreateWindow("Ma Fenetre"); //creation de la fenetre
    glutDisplayFunc(display_callback); //affichage dans la fenetre
    glewInit(); //initialisation des fonctions de glew
    init();
    glutMainLoop(); //boucle permanente

    return 0;
}
```

038

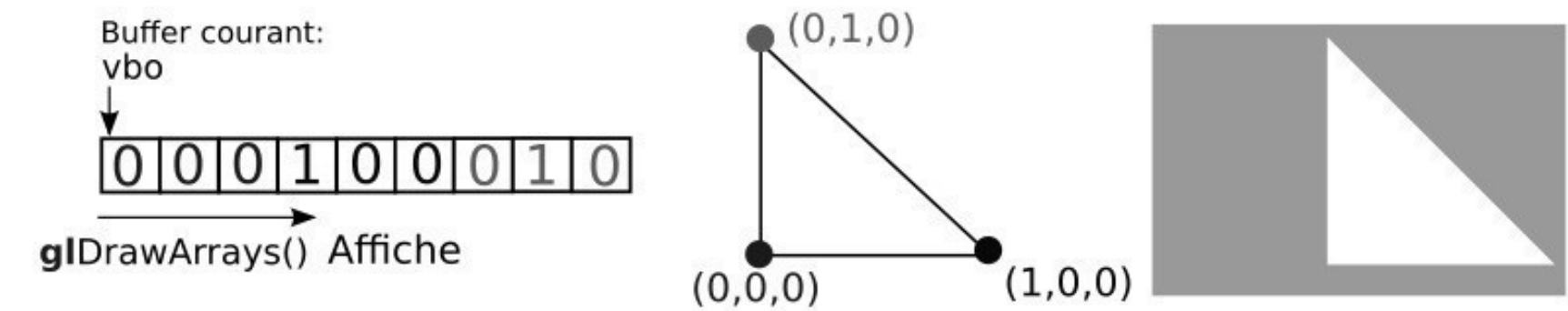
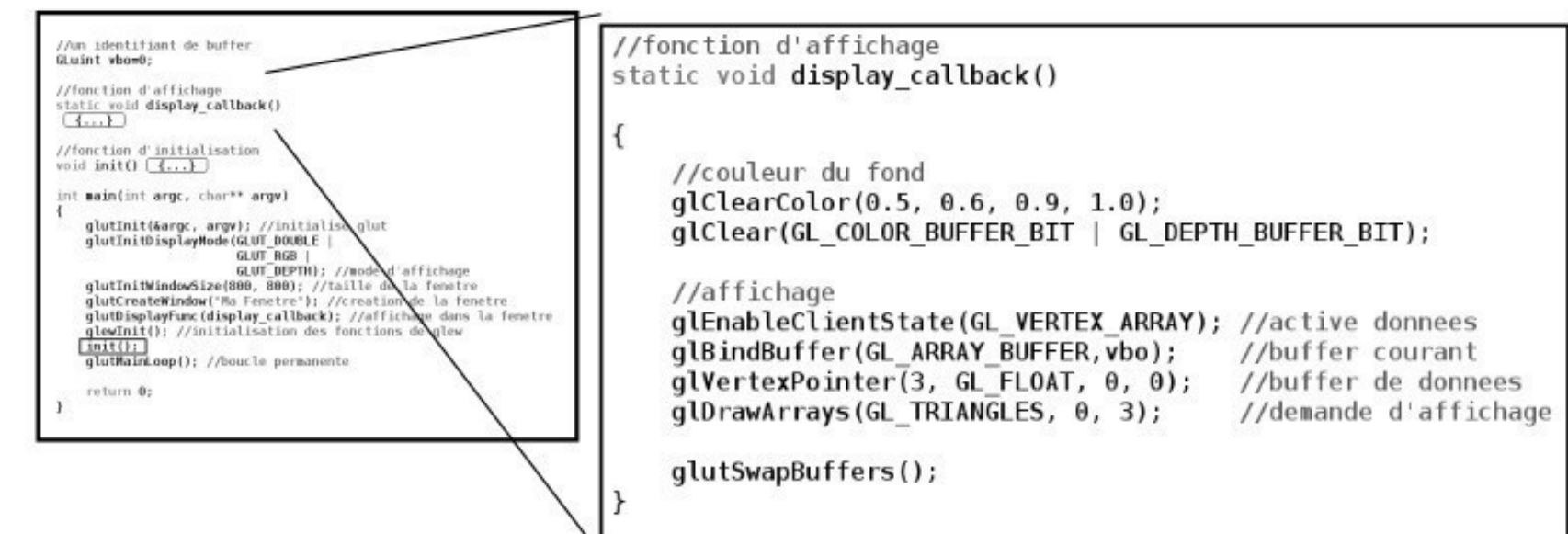
039

Pgm minimal, triangle



040

Pgm minimal, triangle



041

Modes affichages

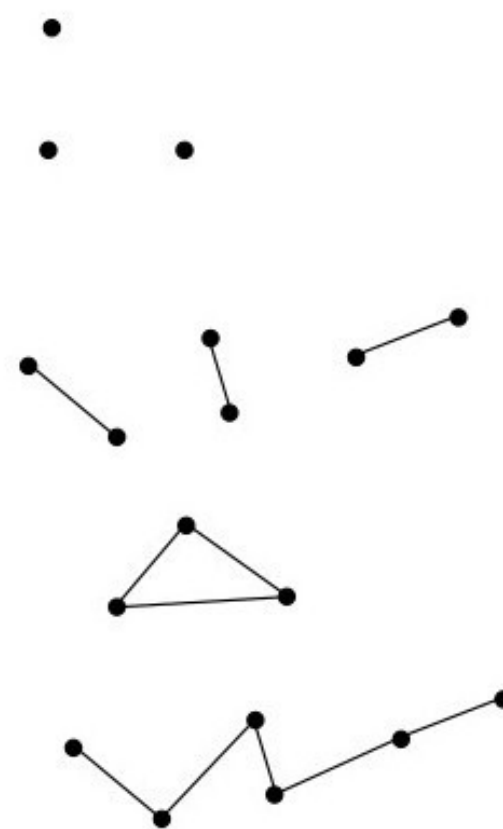
`glDrawArrays(GLenum mode, GLint first, GLsizei count);`

`glPointSize(5);`
`glDrawArrays(GL_POINTS, 0, N);`

`glDrawArrays(GL_LINES, 0, N);`

`glDrawArrays(GL_LINES_LOOP, 0, N);`

`glDrawArrays(GL_LINES_STRIP, 0, N);`



042

Introduction aux Shaders

043

Notion de shaders

Programme en C, C++, etc
Sur CPU

```
int main()
{ ...
```

Création des données
Envoie sur GPU
Demande d'affichage
Gestion évènements

Sur GPU
2 programmes de contrôles

Vertex Shader

appelé pour
tous les sommets

projection
position

Fragment Shader

appelé pour tous les
"pixels colorés"

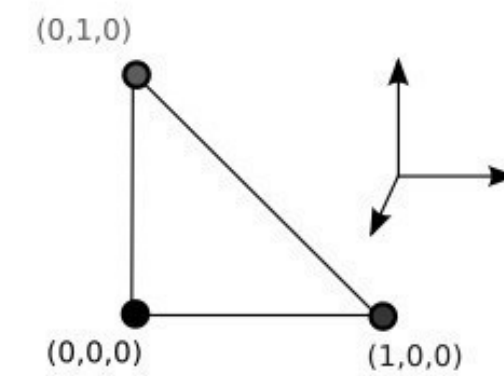
couleurs pixels



044

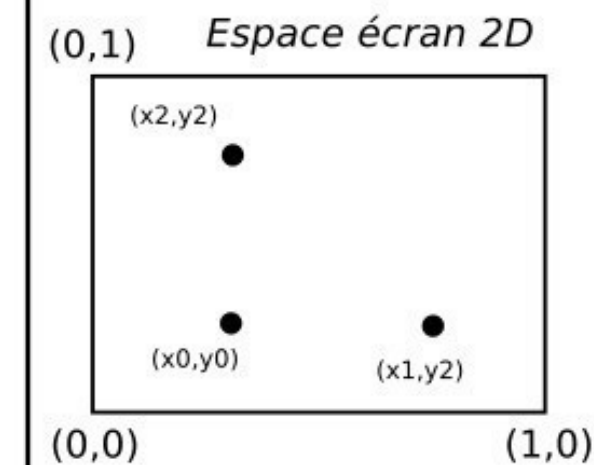
But des shaders

Données 3D



vbo
0 0 0 1 0 0 0 1 0
glDrawArrays()

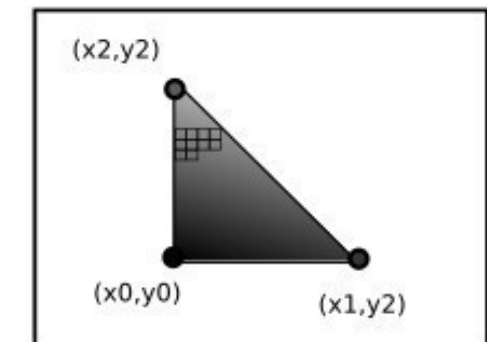
Vertex Shader



entrée:
coordonnées 3D

sortie:
coordonnées 2D
espace écran

Fragment Shader



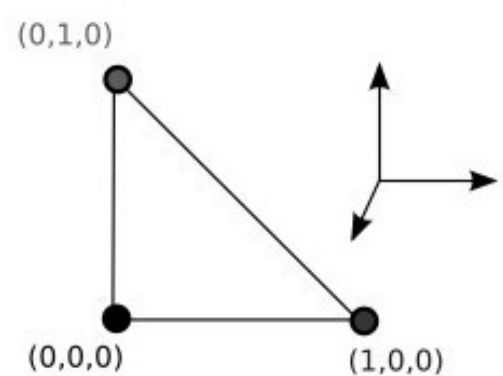
entrée:
coordonnée
fragment courant
+ interpolation
linéaire données

sortie:
couleur du
fragment

045

Parallélisme

Données



vbo
0 0 0 1 0 0 0 1 0
glDrawArrays()

Vertex shader

entrée
(coordonnées 3D)

gl_Vertex=

0 0 0

sortie
(coordonnée écran)

(x0,y0)

1 0 0

(x1,y1)

0 1 0

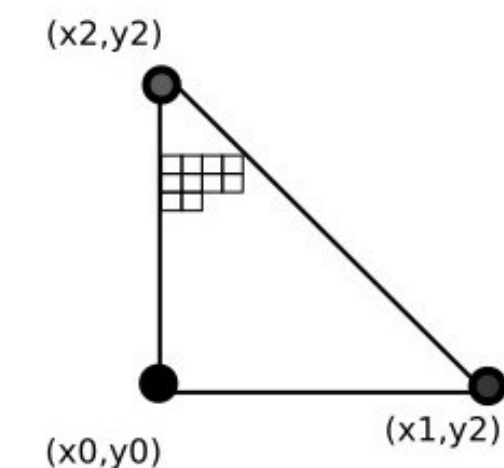
(x2,y2)

en parallèle

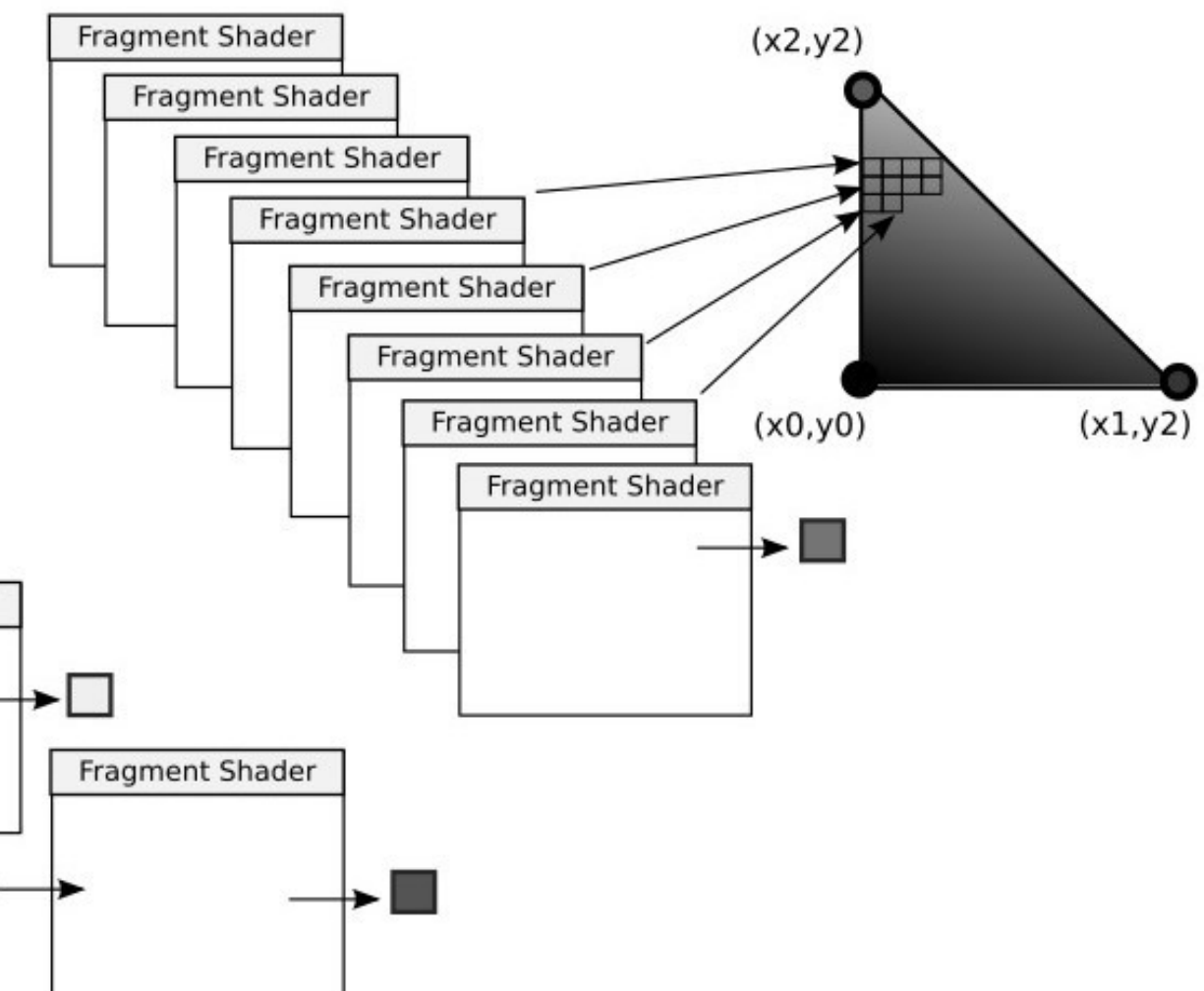
046

Parallélisme

Fragment shader



(x,y)
(x',y')



047

Shaders en pratique

Créez 2 fichiers textes:

```
#version 120

//Un Vertex Shader minimaliste
void main (void)
{
    //Coordonnees du sommet
    gl_Position = gl_Vertex;
}
```

shader.vert

```
#version 120

//Un Fragment Shader minimaliste
void main (void)
{
    //couleur du fragment
    gl_FragColor = vec4(1.0f,0.0f,0.0f,1.0f);
}
```

shader.frag

Chargez les fichiers dans le programme principal

```
glCreateProgram();
glCreateShader(...);
glShaderSource(...);
glCompileShader(...);
glAttachShader(...);
glLinkProgram(...);
glUseProgram(...);
```

048

Shaders en pratique

```
#version 120

//Un Vertex Shader minimaliste
void main (void)
{
    //Coordonnees du sommet
    gl_Position = gl_Vertex;
}
```

shader.vert

Ressemble
à du C

Variable déjà connue
position de sortie des sommets

Variable déjà connue
position d'entrée des sommets

049

Shaders en pratique

```
#version 120

//Un Fragment Shader minimaliste
void main (void)
{
    //couleur du fragment
    gl_FragColor = vec4(1.0f,0.0f,0.0f,1.0f);
}
```

shader.frag

Variable de sortie:
couleur du pixel

Contrôle
de la couleur

Ce n'est pas du C
Ressemble à du C++
=> C'est du GLSL



050

Notions de GLSL

Langage proche du C, et C++ en plus simple
Possède les éléments de calculs

int
float

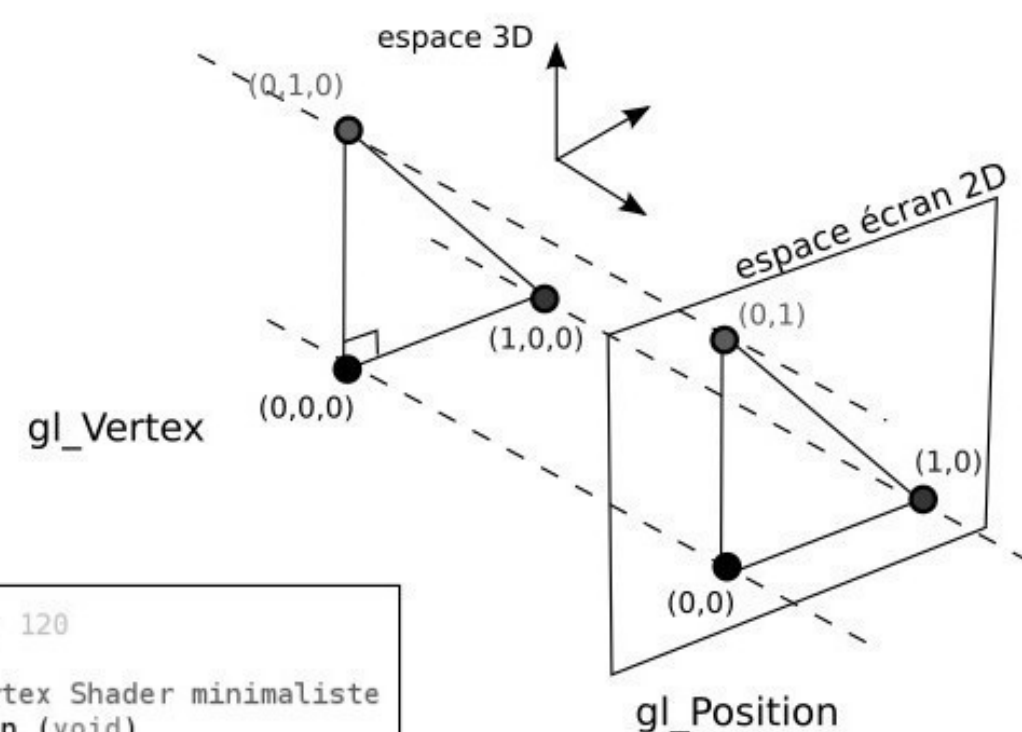
vec2(x,y);
vec3(x,y,z);
vec4(x,y,z,w);

vec3 a(1,4,3);
vec2 b=a.xy;
vec2 c=a.zx;
vec2 c=vec2(a.y,4);

mat2();
mat3();
mat4();

051

Projection

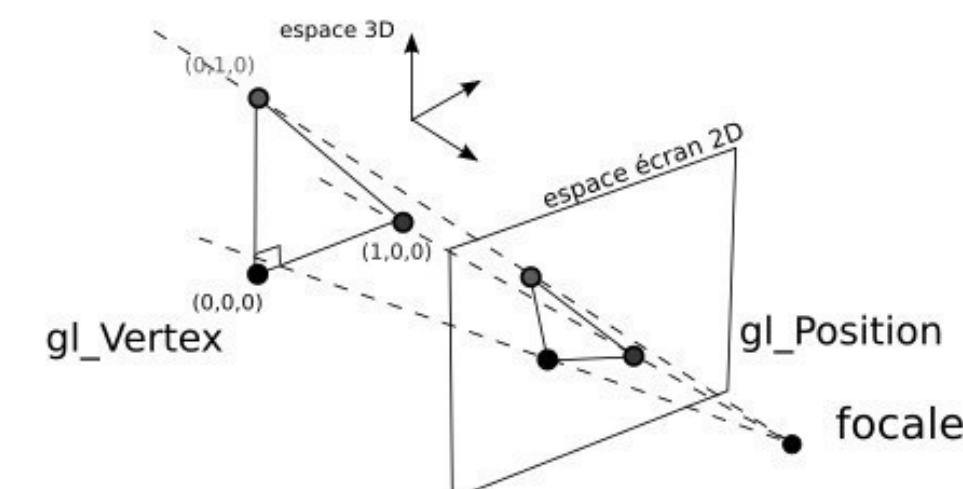
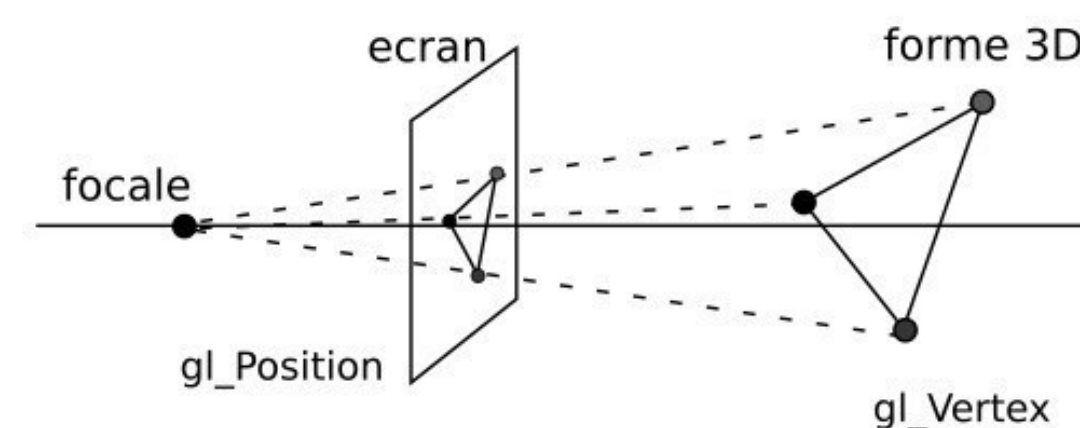


```
#version 120
//Un Vertex Shader minimaliste
void main (void)
{
    //Coordonnees du sommet
    gl_Position = gl_Vertex;
}
```

Projection orthogonale

052

Projection



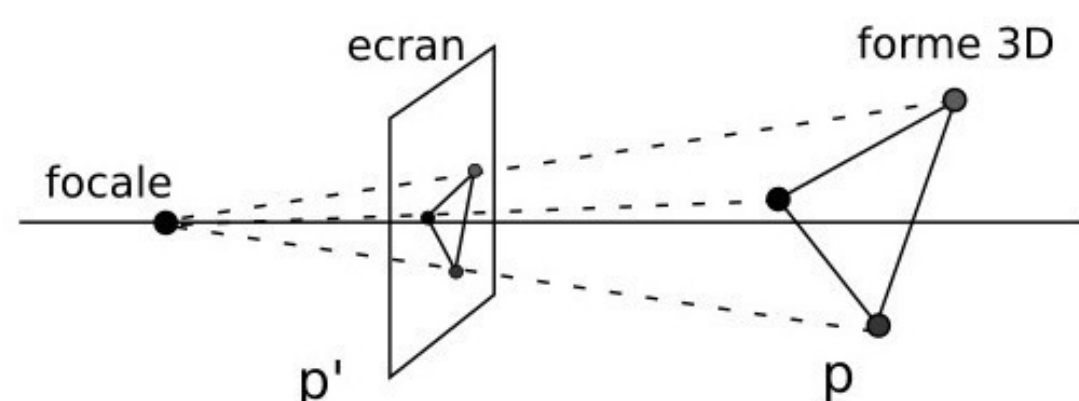
$gl_Position = projection_perspective(gl_Vertex)$

053

Projection perspective

$gl_Position = projection_perspective(gl_Vertex)$

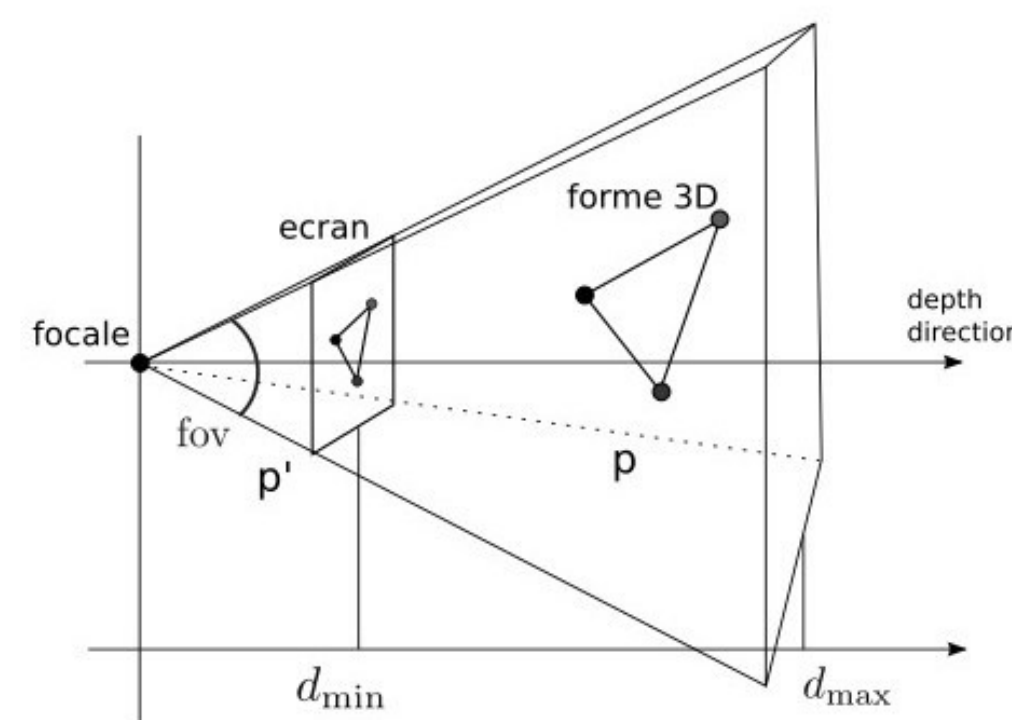
projection_perspective : Matrice de projection 4x4
(voir espace projectif 2nd semestre)



$$p' = M p$$

054

Projection perspective



a: aspect ratio y/x

$$p' = M p$$

$$M = \begin{pmatrix} f_x & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & C & D \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

$$f = \cotan(fov/2)$$

$$f_x = f/a$$

$$C = \frac{d_{max} + d_{min}}{d_{max} - d_{min}}$$

$$D = -2 \frac{d_{max} d_{min}}{d_{max} - d_{min}}$$

055

Projection perspective

Exemple de vertex shader

```
mat4 projection=mat4(vec4(1.7321f,0.0f,0.0f,0.0f),
                     vec4(0.0f,1.7321f,0.0f,0.0f),
                     vec4(0.0f,0.0f,1.1053f,1.0f),
                     vec4(0.0f,0.0f,-1.0526f,0.0f));
//Un Vertex Shader minimaliste
void main (void)
{
    //Coordonnees du sommet
    vec4 p=projection*gl_Vertex;

    //position dans l'espace ecran
    gl_Position = p;
}
```

attention:
vecteurs colonnes

fov=60 degrés
a=1
d_min=0.5
d_max=10



z=2



z=6

056

Passage d'arguments uniform

Il est possible de passer des paramètres du CPU vers les shaders

display_callback()

```
float valeur_a_passer=rotation_x/360.0;
glUniform1f (get_uni_loc(shader_program_id, "valeur"), valeur_a_passer);
```

shader.frag()

```
uniform float valeur;

//Un Fragment Shader minimaliste
void main (void)
{
    //couleur du fragment
    gl_FragColor = vec4(valeur,valeur,0,1);
}
```

uniform:
paramètre ayant la même valeur pour l'ensemble des shaders.

057

Passage d'arguments uniform

Utilisation: - Interaction clavier, souris
- Temps

```
glUniform1f (get_uni_loc(shader, string), value);
1u
2f
3d
Matrix4fv
...
```

058

Rem. fonctions OpenGL

L'API d'OpenGL est en C: 1 nom par types d'arguments

```
gl[NOM]f(...);
gl[NOM]i(...);
gl[NOM]u(...);
gl[NOM]d(...);
gl[NOM]vf(...);
```

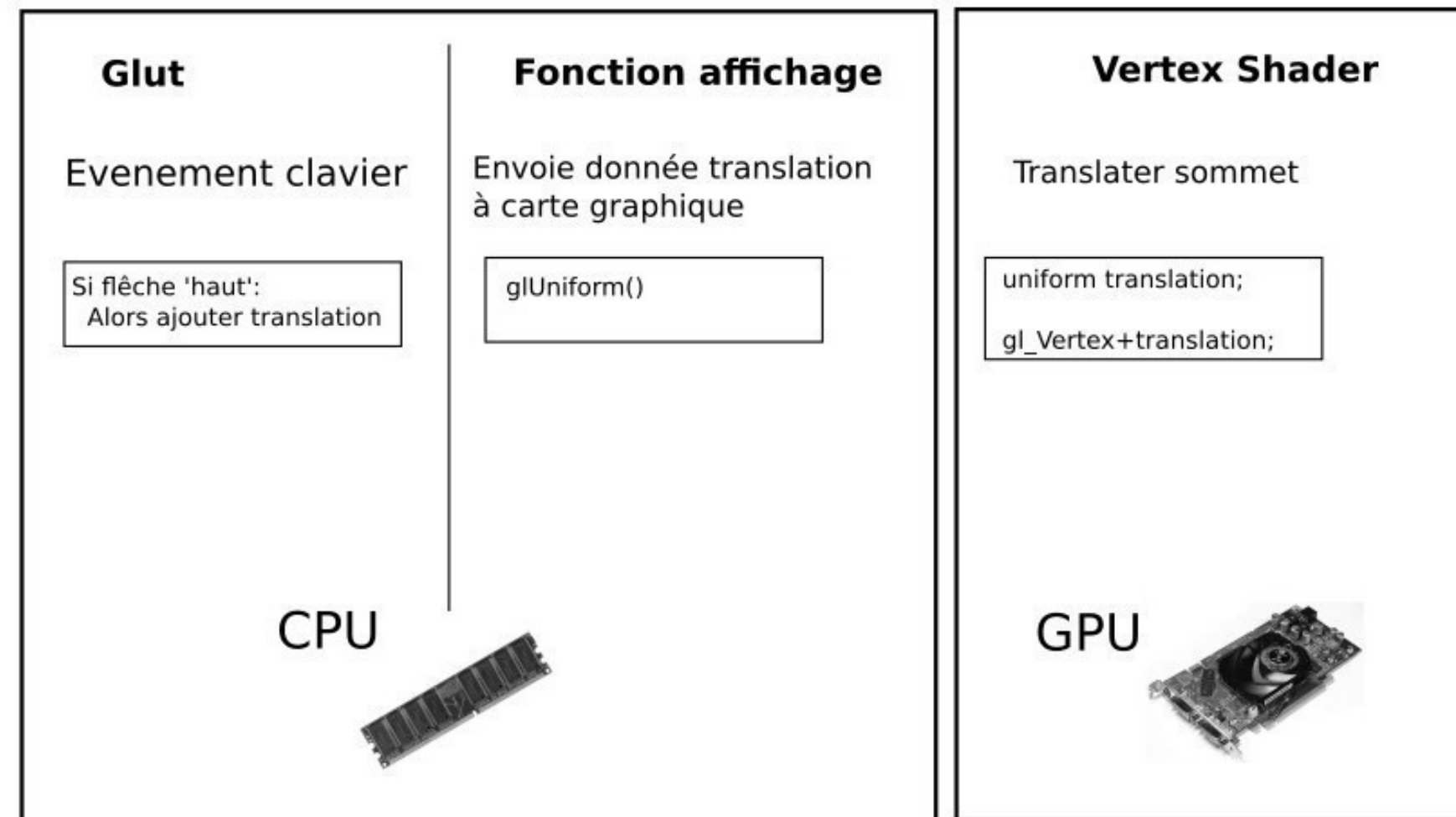
ex.
glUniform1f(...)
glUniform2u(...)

```
gl[NOM]3f(...);
gl[NOM]2u(...);
```

059

Interaction

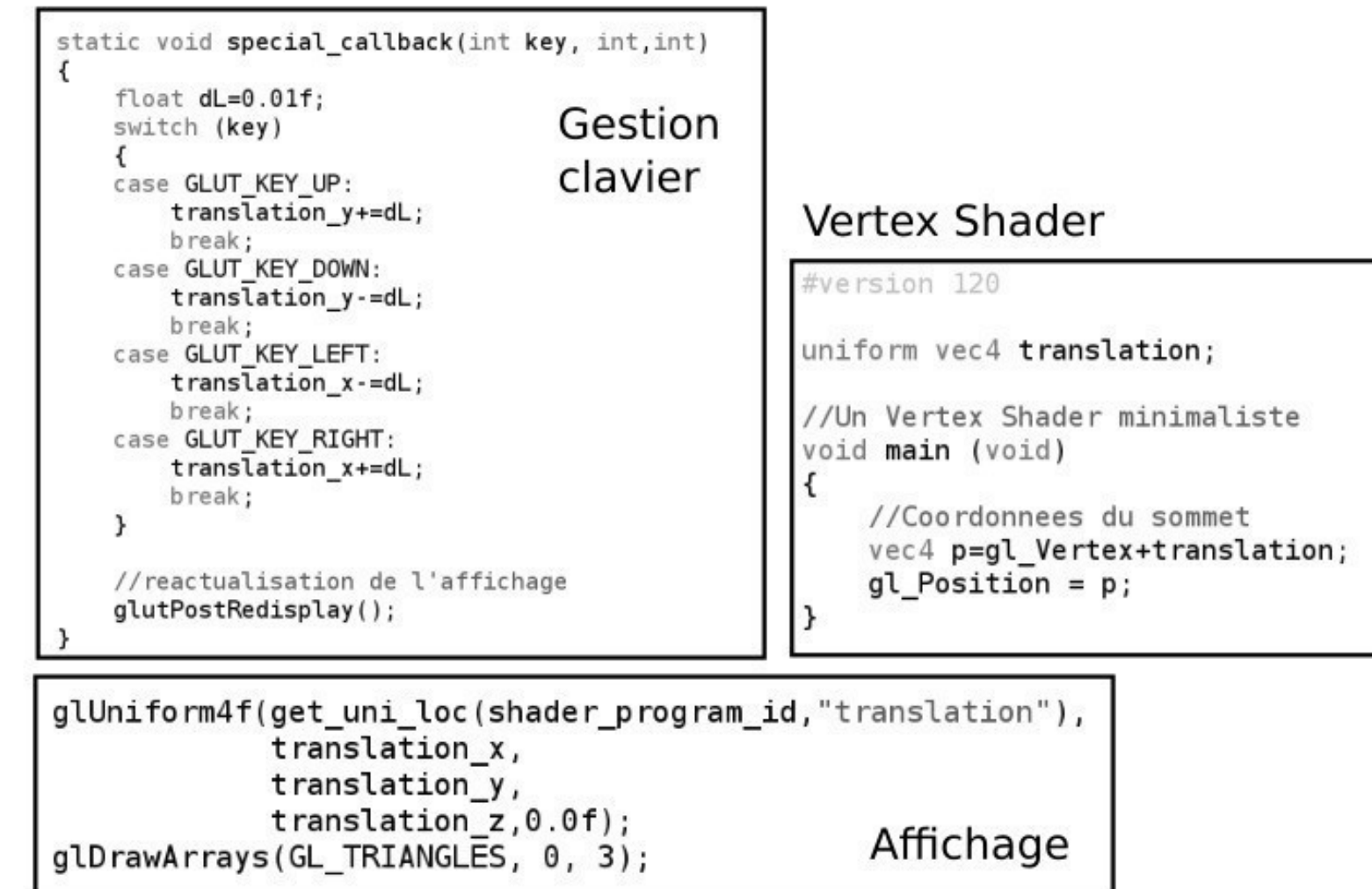
Possibilité de déplacer un objet dans le vertex shader



060

Interaction

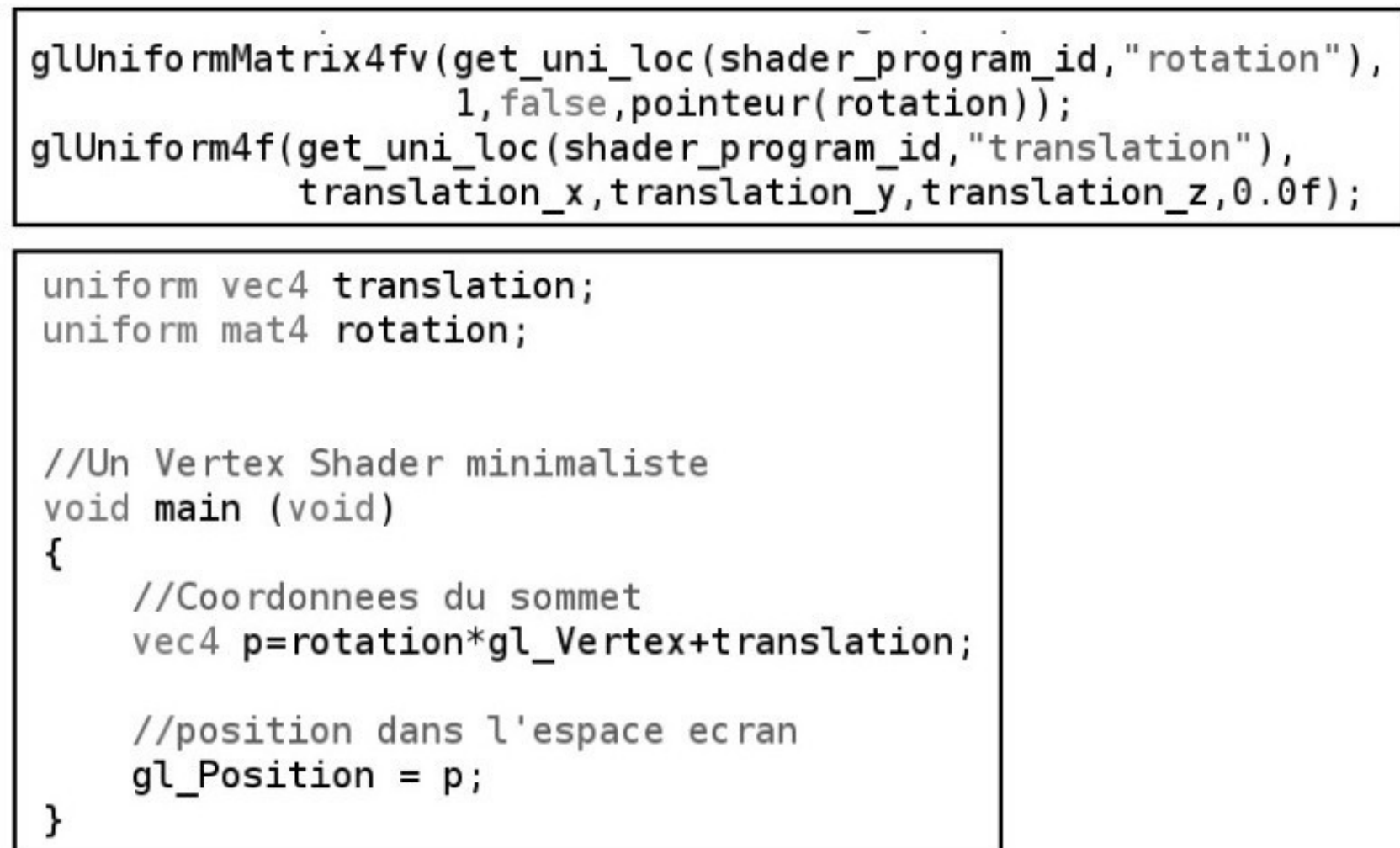
Possibilité de déplacer un objet dans le vertex shader



061

Interaction

Rotation possible également

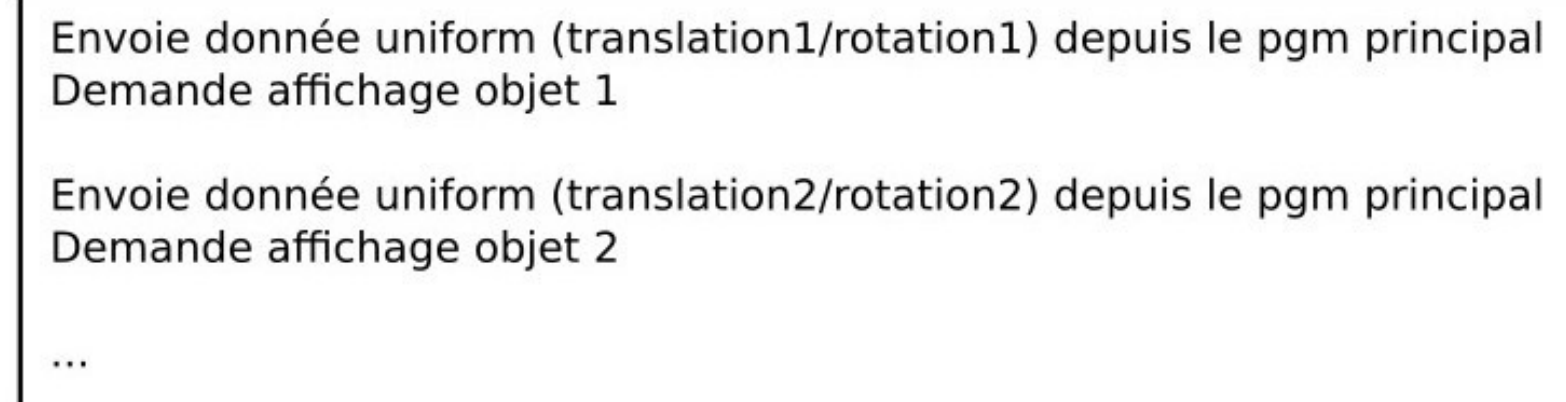


062

Interaction

Il est possible d'appliquer différentes rotation/translation à différents objets

Principe:



063

Interaction

Une translation/rotation appliquée à un objet et pas aux autres
=> Déformation d'un objet

Une translation/rotation appliquée à tous les objets
=> Déplacement de la caméra

Il n'y a aucune différence entre déplacer la 'caméra'
et déplacer tous les objets de la scène.

$$p' = \text{rotation} * p + \text{translation}$$

064

Vertex shader + déformation

```
#version 120

uniform vec4 translation;
uniform mat4 rotation;
uniform mat4 projection;

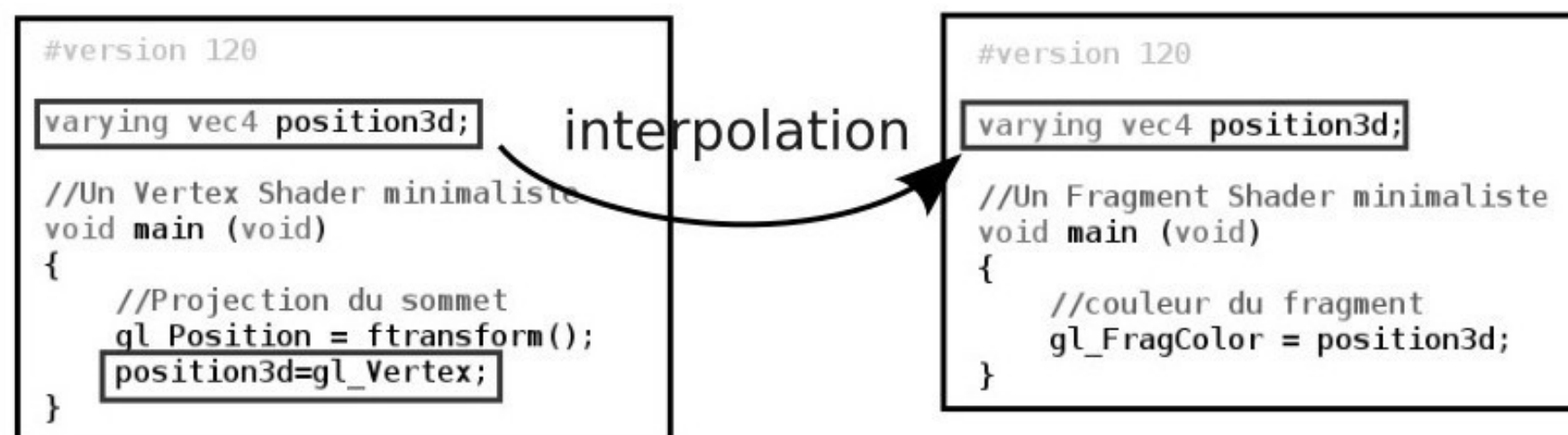
//Un Vertex Shader minimaliste
void main (void)
{
    //Coordonnees du sommet
    vec4 p = rotation * gl_Vertex + translation;
    p = projection * p;

    //position dans l'espace ecran
    gl_Position = p;
}
```

recu du programme principal

065

Passage d'arguments varying



(0,1,0)

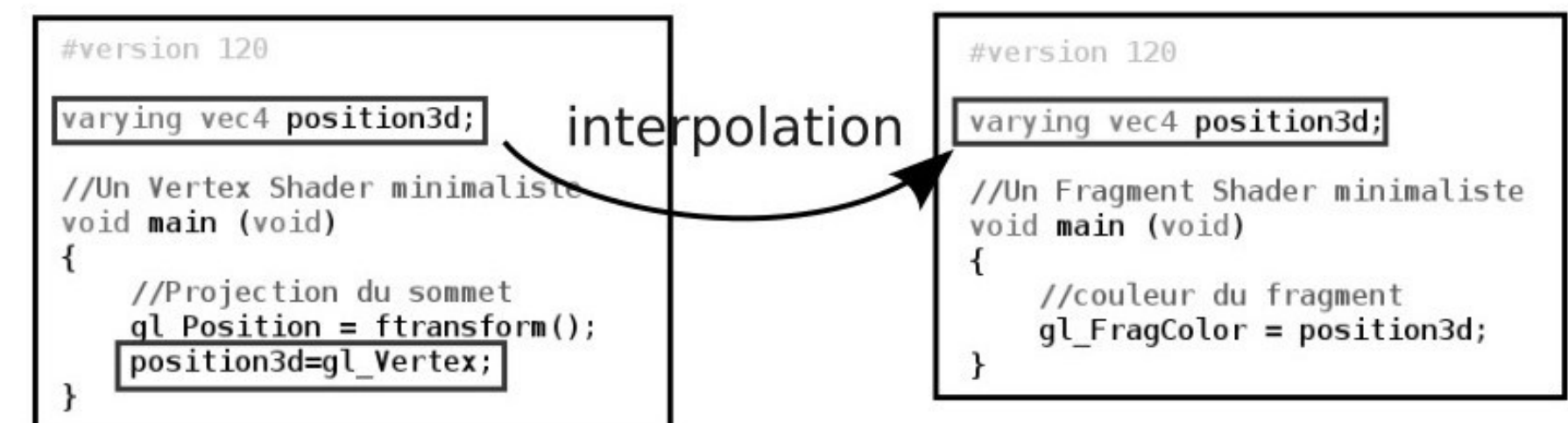
(0,0,0)

(1,0,0)



066

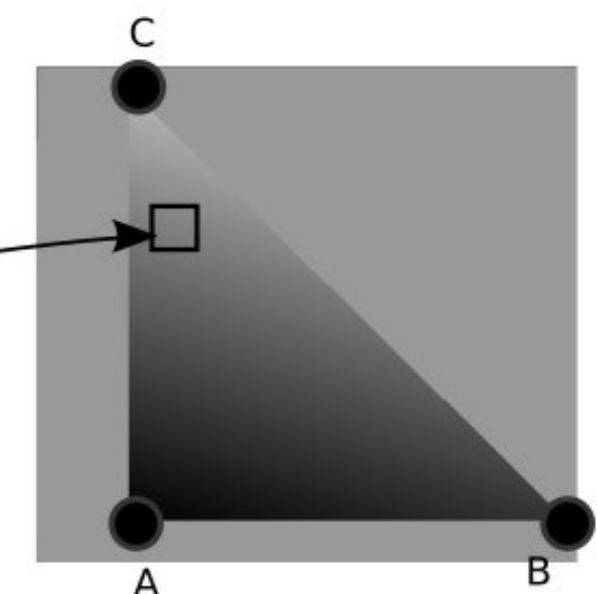
Passage d'arguments varying



Interpolation barycentrique (linéaire)
réalisée automatiquement par la carte graphique:

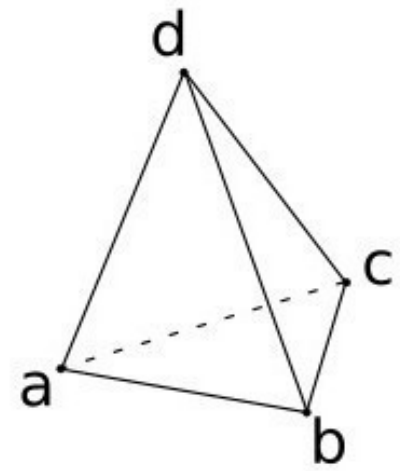
$$\begin{aligned} \text{position}_{3d} &= \alpha A + \beta B + \gamma C \\ \alpha + \beta + \gamma &= 1 \\ (\alpha, \beta, \gamma) &\in [0, 1]^3 \end{aligned}$$

pour tous
les fragments



067

Maillages



```
float vertex[] =
{xa, ya, za, xb, yb, zb, xd, yd, zd,
 xb, yb, zb, xc, yc, zc, xd, yd, zd,
 xc, yc, zc, xd, yd, zd, xa, ya, za,
 xa, ya, za, xb, yb, zb, xc, yc, zc};

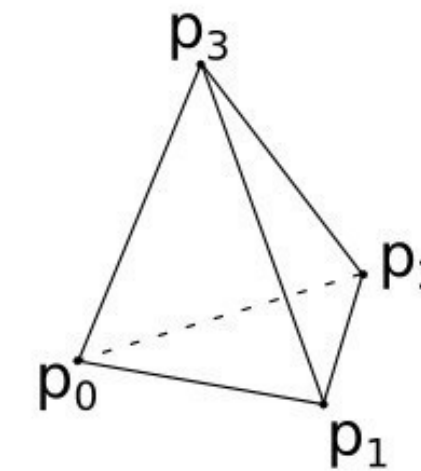
glBufferData(...)
```

Répétition:

- perte de place
- modification complexe

068

Maillages, format indexé



Séparation:
géométrie 3D / connectivité

```
float vertex[] =
{p0x, p0y, p0z, p1x, p1y, p1z,
 p2x, p2y, p2z, p3x, p3y, p3z};

float indices[] =
{0, 1, 3, 1, 2, 3, 0, 3, 2, 0, 2, 1};
```

2 Types de buffers sur le GPU

Buffer de géométrie

p0x p0y p0z p1x ...
float

Buffer d'indices

0 1 3 1 2 3 0 ...
unsigned int

+ Coordonnées écrites une seule fois

069

Format indexé

Initialisation

```
vec3 p0(0,0,0);
vec3 p1(1,0,0);
vec3 p2(0,1,0);
vec3 p3(0,0,1);
vec3 vertex[]={p0,p1,p2,p3};

triangle_index triangle0(0,1,2);
triangle_index triangle1(0,1,3);
triangle_index triangle2(0,2,3);
triangle_index triangle3(3,1,2);
triangle_index index[]={triangle0,triangle1,triangle2,triangle3};

glGenBuffers(1,&vbo);
glBindBuffer(GL_ARRAY_BUFFER,vbo);
glBufferData(GL_ARRAY_BUFFER,sizeof(vertex),vertex,GL_STATIC_DRAW);

glGenBuffers(1,&vboi);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER,vboi);
glBufferData(GL_ELEMENT_ARRAY_BUFFER,sizeof(index),index,GL_STATIC_DRAW);
```

Affichage

```
glDrawElements(GL_TRIANGLES, 4*3, GL_UNSIGNED_INT, 0);
```

070

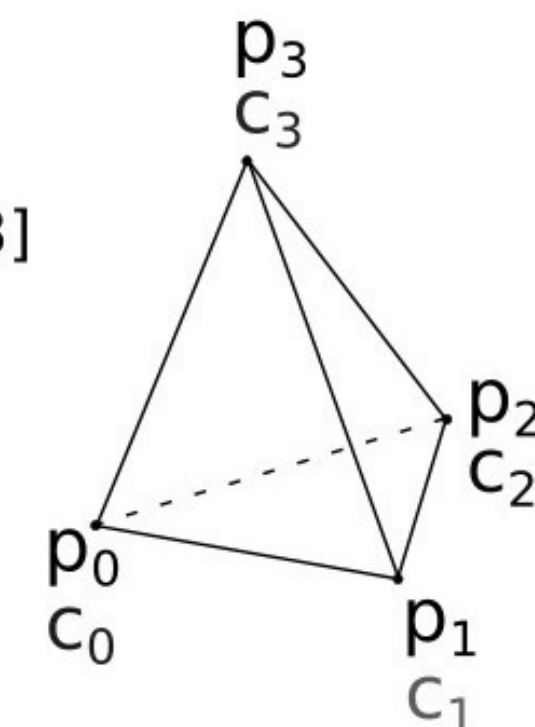
Entrelacement de données

On souhaite envoyer au GPU:

4 Sommets

- géométrie (x,y,z)
- couleur (r,g,b)

donnees:
[p0,c0,p1,c1,p2,c2,p3,c3]
↑ ↑
3 floats 3 floats



071

Entrelacement de données

```
vec3 p0(0,0,0); vec3 c0(1,0,0);
vec3 p1(1,0,0); vec3 c1(0,1,0);
vec3 p2(0,1,0); vec3 c2(0,0,1);
vec3 p3(0,0,1); vec3 c3(0,1,1);
vec3 vertex[]={p0,c0 , p1,c1 , p2,c2 , p3,c3};

triangle_index triangle0(0,1,2); triangle_index triangle1(0,1,3);
triangle_index triangle2(0,2,3); triangle_index triangle3(3,1,2);
triangle_index index[]={triangle0,triangle1,triangle2,triangle3};

glGenBuffers(1,&vbo);
glBindBuffer(GL_ARRAY_BUFFER,vbo);
glBufferData(GL_ARRAY_BUFFER,sizeof(vertex),vertex,GL_STATIC_DRAW);

glGenBuffers(1,&vboi);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER,vboi);
glBufferData(GL_ELEMENT_ARRAY_BUFFER,sizeof(index),index,GL_STATIC_DRAW);
```

Init



```
glEnableClientState(GL_VERTEX_ARRAY);
glVertexPointer(3, GL_FLOAT, 2*3*sizeof(float), 0);

glEnableClientState(GL_COLOR_ARRAY);
long int offset_couleur=3*sizeof(float);
glColorPointer(3, GL_FLOAT, 2*3*sizeof(float), (GLubyte*)offset_couleur);

glDrawElements(GL_TRIANGLES, 4*3, GL_UNSIGNED_INT, 0);
```

Display

072

Entrelacement de données

```
vec3 p0(0,0,0); vec3 c0(1,0,0);
vec3 p1(1,0,0); vec3 c1(0,1,0);
vec3 p2(0,1,0); vec3 c2(0,0,1);
vec3 p3(0,0,1); vec3 c3(0,1,1);
vec3 vertex[]={p0,c0 , p1,c1 , p2,c2 , p3,c3};

triangle_index triangle0(0,1,2); triangle_index triangle1(0,1,3);
triangle_index triangle2(0,2,3); triangle_index triangle3(3,1,2);
triangle_index index[]={triangle0,triangle1,triangle2,triangle3};

glGenBuffers(1,&vbo);
glBindBuffer(GL_ARRAY_BUFFER,vbo);
glBufferData(GL_ARRAY_BUFFER,sizeof(vertex),vertex,GL_STATIC_DRAW);

glGenBuffers(1,&vboi);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER,vboi);
glBufferData(GL_ELEMENT_ARRAY_BUFFER,sizeof(index),index,GL_STATIC_DRAW);

glEnableClientState(GL_VERTEX_ARRAY);
glVertexPointer(3, GL_FLOAT, 2*3*sizeof(float), 0);

glEnableClientState(GL_COLOR_ARRAY);
long int offset_couleur=3*sizeof(float);
glColorPointer(3, GL_FLOAT, 2*3*sizeof(float), (GLubyte*)offset_couleur);

glDrawElements(GL_TRIANGLES, 4*3, GL_UNSIGNED_INT, 0);
```

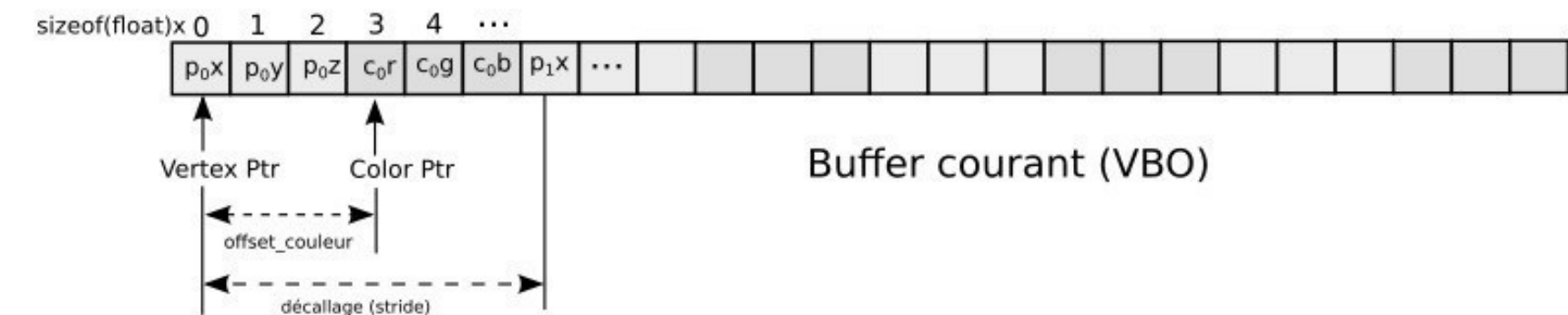
Init



Display

position (float)

couleur (float)



073

Autre possibilité

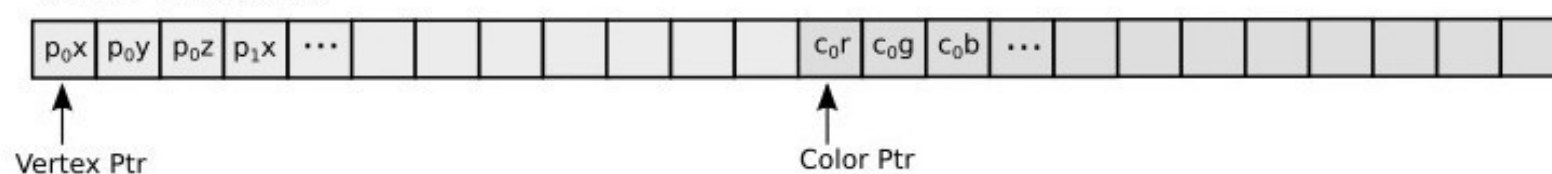
```
vec3 p0(0,0,0); vec3 c0(1,0,0);
vec3 p1(1,0,0); vec3 c1(0,1,0);
vec3 p2(0,1,0); vec3 c2(0,0,1);
vec3 p3(0,0,1); vec3 c3(0,1,1);
vec3 vertex[]={p0,p1,p2,p3 , c0,c1,c2,c3};
```

```
glEnableClientState(GL_VERTEX_ARRAY);
glVertexPointer(3, GL_FLOAT, 0, 0);

glEnableClientState(GL_COLOR_ARRAY);
long int offset_couleur=4*3*sizeof(float);
glColorPointer(3, GL_FLOAT, 0, (GLubyte*)offset_couleur);

glDrawElements(GL_TRIANGLES, 4*3, GL_UNSIGNED_INT, 0);
```

VBO Courant



074

Autre possibilité

```
vec3 p0(0,0,0); vec3 c0(1,0,0);
vec3 p1(1,0,0); vec3 c1(0,1,0);
vec3 p2(0,1,0); vec3 c2(0,0,1);
vec3 p3(0,0,1); vec3 c3(0,1,1);
vec3 vertex[]={p0,p1,p2,p3};
vec3 color[]={c0,c1,c2,c3};

triangle_index triangle0(0,1,2); triangle_index triangle1(0,1,3);
triangle_index triangle2(0,2,3); triangle_index triangle3(3,1,2);
triangle_index index[]={triangle0,triangle1,triangle2,triangle3};

glGenBuffers(1,&vbo_position);
glBindBuffer(GL_ARRAY_BUFFER,vbo_position);
glBufferData(GL_ARRAY_BUFFER,sizeof(vertex),vertex,GL_STATIC_DRAW);

glGenBuffers(1,&vbo_couleur);
glBindBuffer(GL_ARRAY_BUFFER,vbo_couleur);
glBufferData(GL_ARRAY_BUFFER,sizeof(color),color,GL_STATIC_DRAW);

glGenBuffers(1,&vboi);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER,vboi);
glBufferData(GL_ELEMENT_ARRAY_BUFFER,sizeof(index),index,GL_STATIC_DRAW);
```

Init

```
glEnableClientState(GL_VERTEX_ARRAY);
glBindBuffer(GL_ARRAY_BUFFER,vbo_position);
glVertexPointer(3, GL_FLOAT, 0, 0);

glEnableClientState(GL_COLOR_ARRAY);
glBindBuffer(GL_ARRAY_BUFFER,vbo_couleur);
glColorPointer(3, GL_FLOAT, 0, 0);

glDrawElements(GL_TRIANGLES, 4*3, GL_UNSIGNED_INT, 0);
```

Display

075

Autre possibilité

```

Init
vec3 p0(0,0,0); vec3 c0(1,0,0);
vec3 p1(1,0,0); vec3 c1(0,1,0);
vec3 p2(0,1,0); vec3 c2(0,0,1);
vec3 p3(0,0,1); vec3 c3(0,1,1);
vec3 vertex[] = {p0,p1,p2,p3};
vec3 color[] = {c0,c1,c2,c3};

triangle_index triangle0(0,1,2); triangle_index triangle1(0,1,3);
triangle_index triangle2(0,2,3); triangle_index triangle3(3,1,2);
triangle_index index[] = {triangle0,triangle1,triangle2,triangle3};

glGenBuffers(1,&vbo_position);
glBindBuffer(GL_ARRAY_BUFFER,vbo_position);
glBufferData(GL_ARRAY_BUFFER,sizeof(vertex),vertex,GL_STATIC_DRAW);

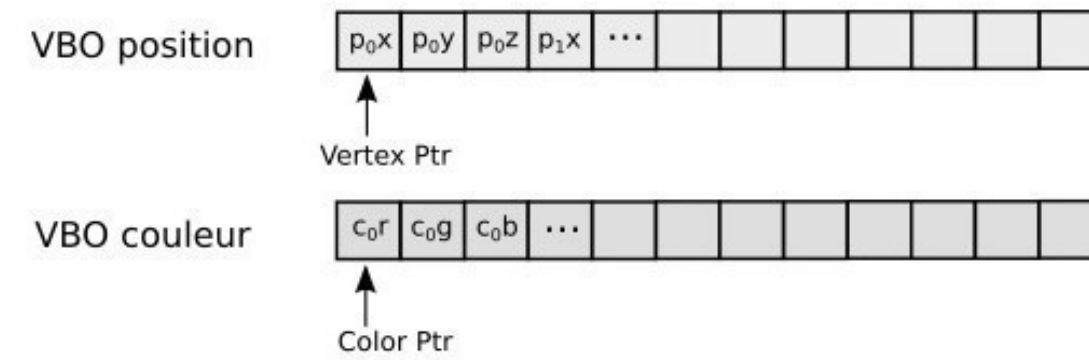
glGenBuffers(1,&vbo_couleur);
glBindBuffer(GL_ARRAY_BUFFER,vbo_couleur);
glBufferData(GL_ARRAY_BUFFER,sizeof(color),color,GL_STATIC_DRAW);

glGenBuffers(1,&vbo_i);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER,vbo_i);
glBufferData(GL_ELEMENT_ARRAY_BUFFER,sizeof(index),index,GL_STATIC_DRAW);

Display
glEnableClientState(GL_VERTEX_ARRAY);
glBindBuffer(GL_ARRAY_BUFFER,vbo_position);
glVertexPointer(3, GL_FLOAT, 0, 0);

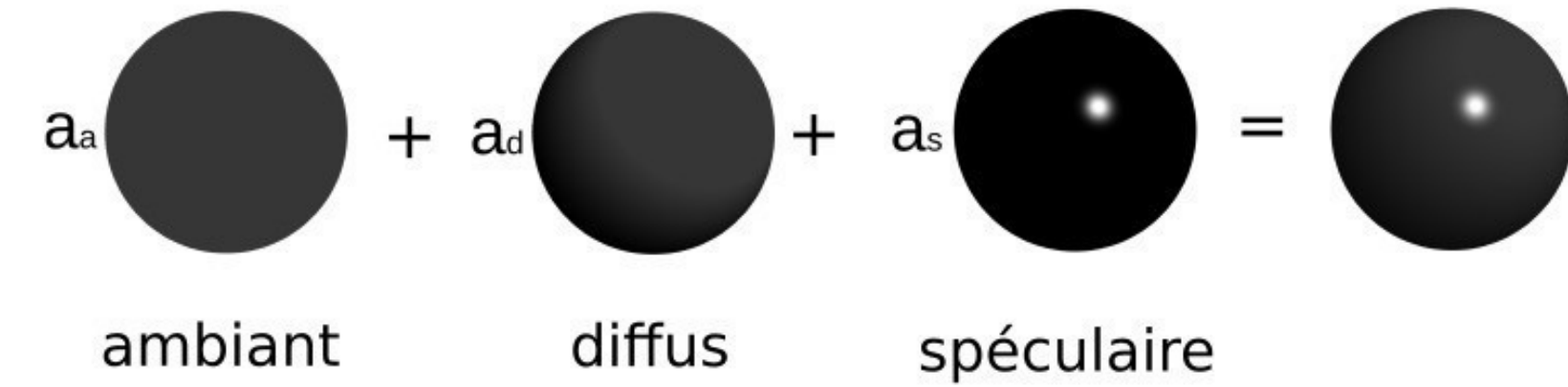
glEnableClientState(GL_COLOR_ARRAY);
glBindBuffer(GL_ARRAY_BUFFER,vbo_couleur);
glColorPointer(3, GL_FLOAT, 0, 0);

glDrawElements(GL_TRIANGLES, 4*3, GL_UNSIGNED_INT, 0);
    
```



076

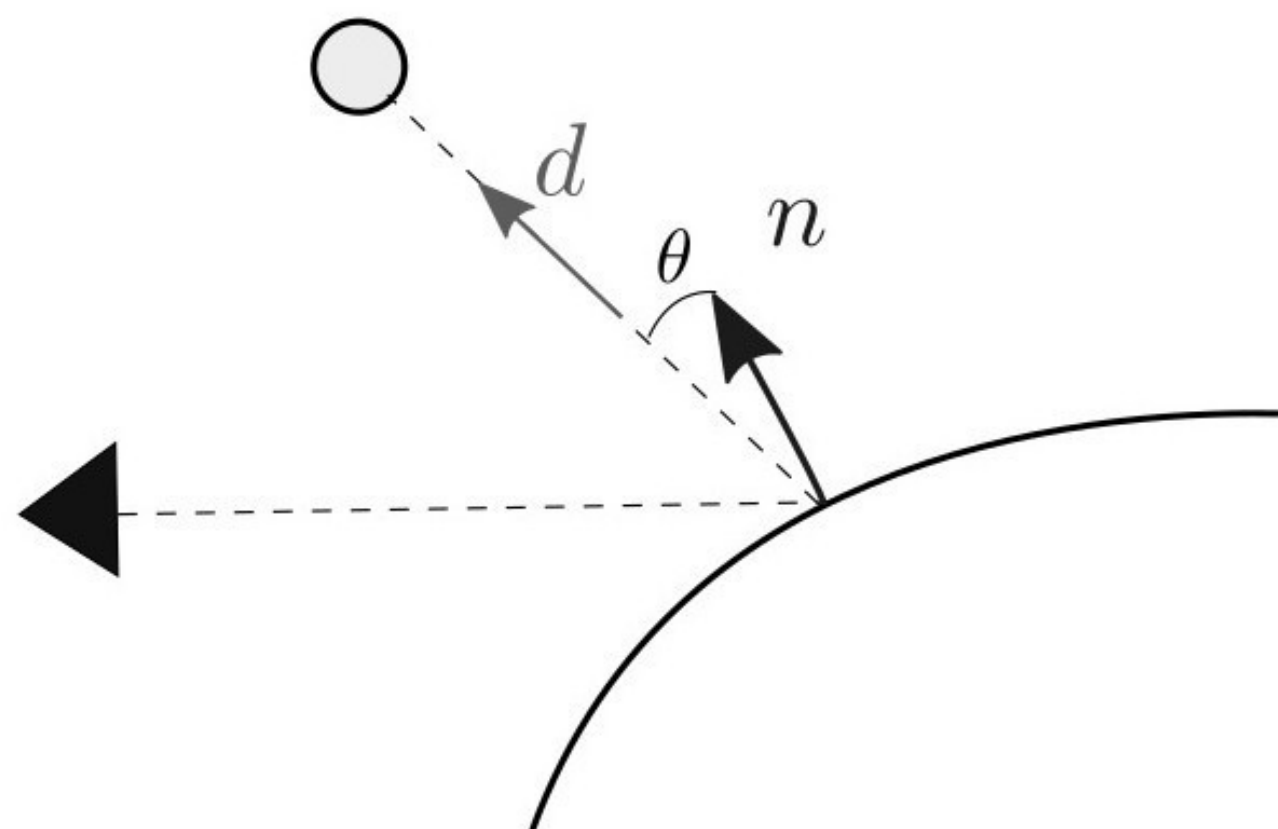
Illumination



077

Illumination

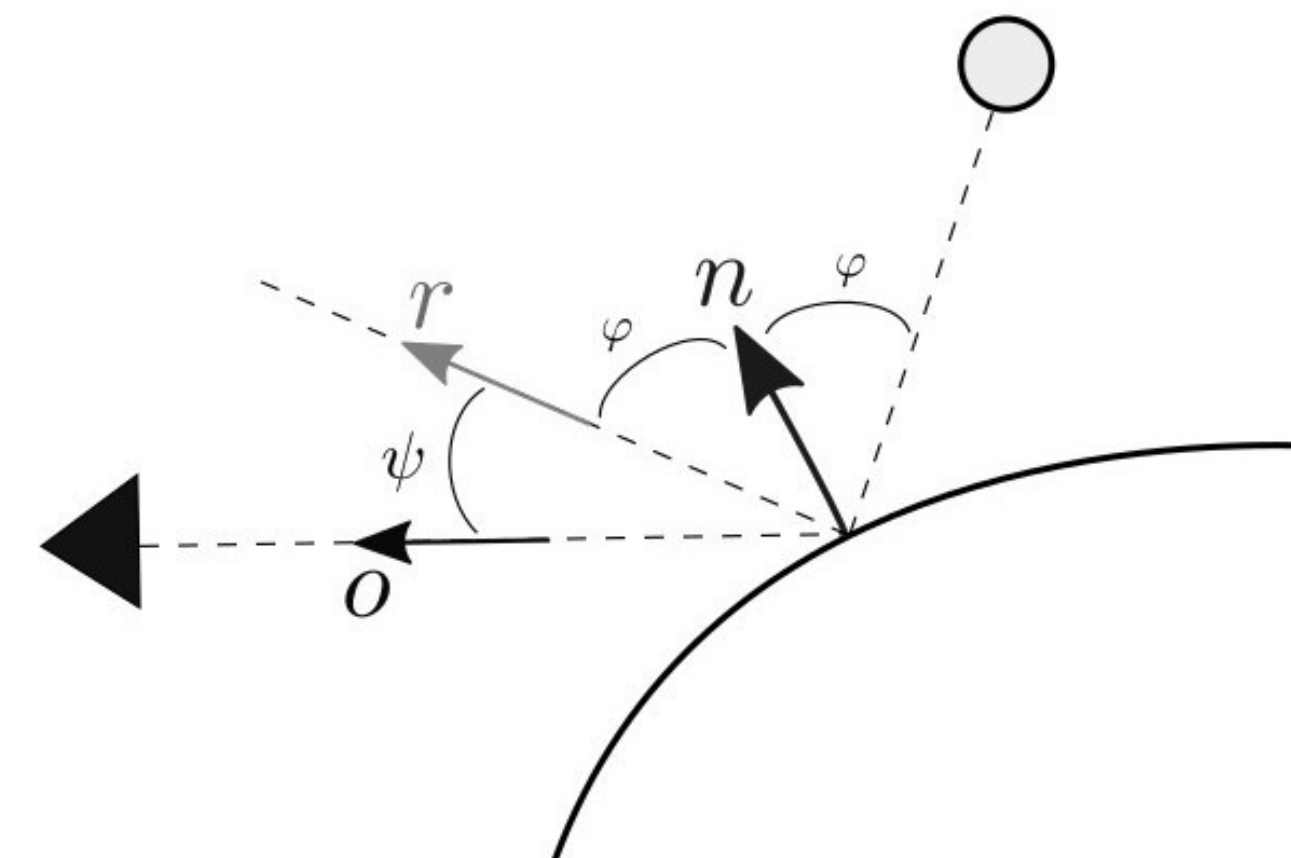
Coefficient diffus $c_d = \cos(\theta)$



078

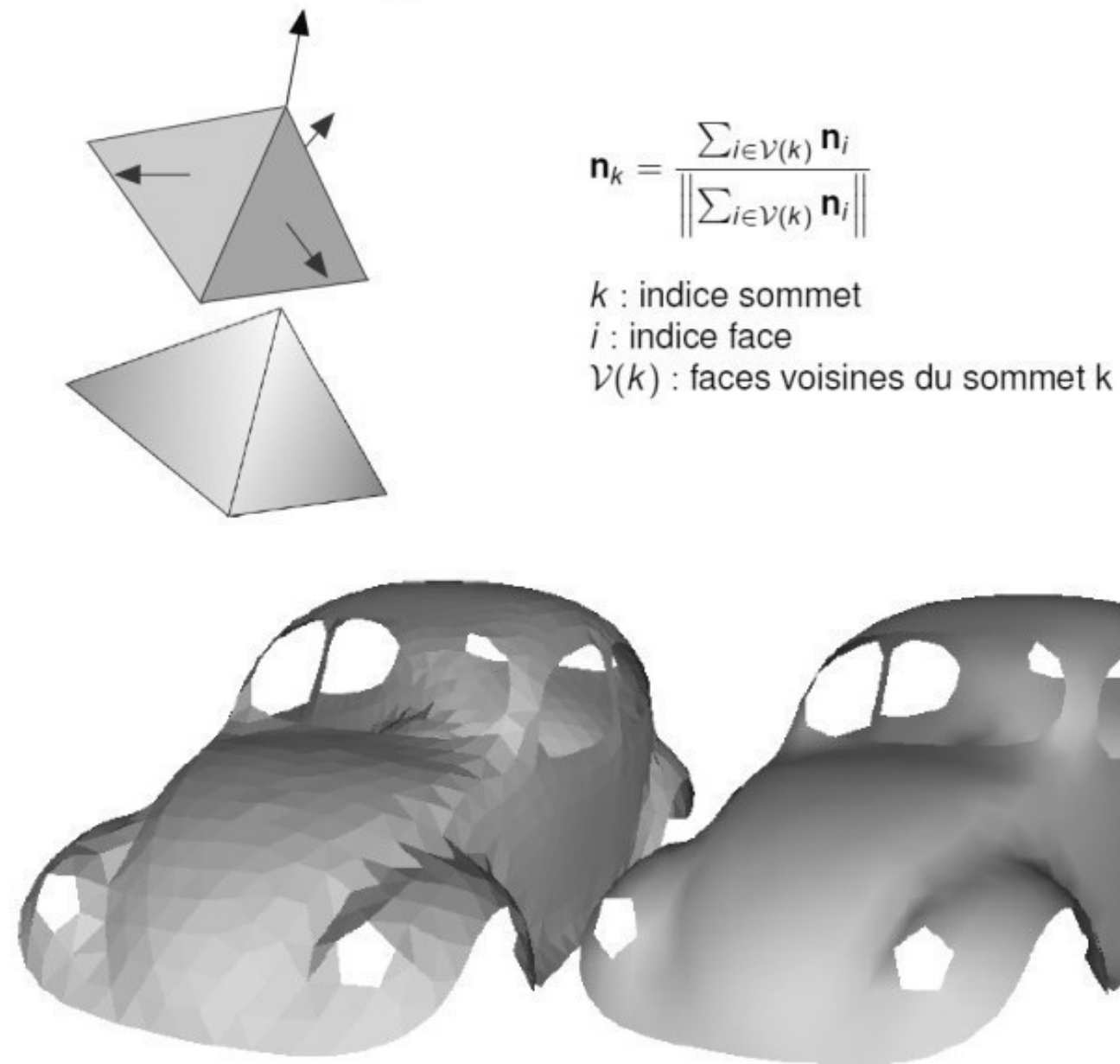
Illumination

Coefficient spéculaire $c_s = \cos(\psi)^n$



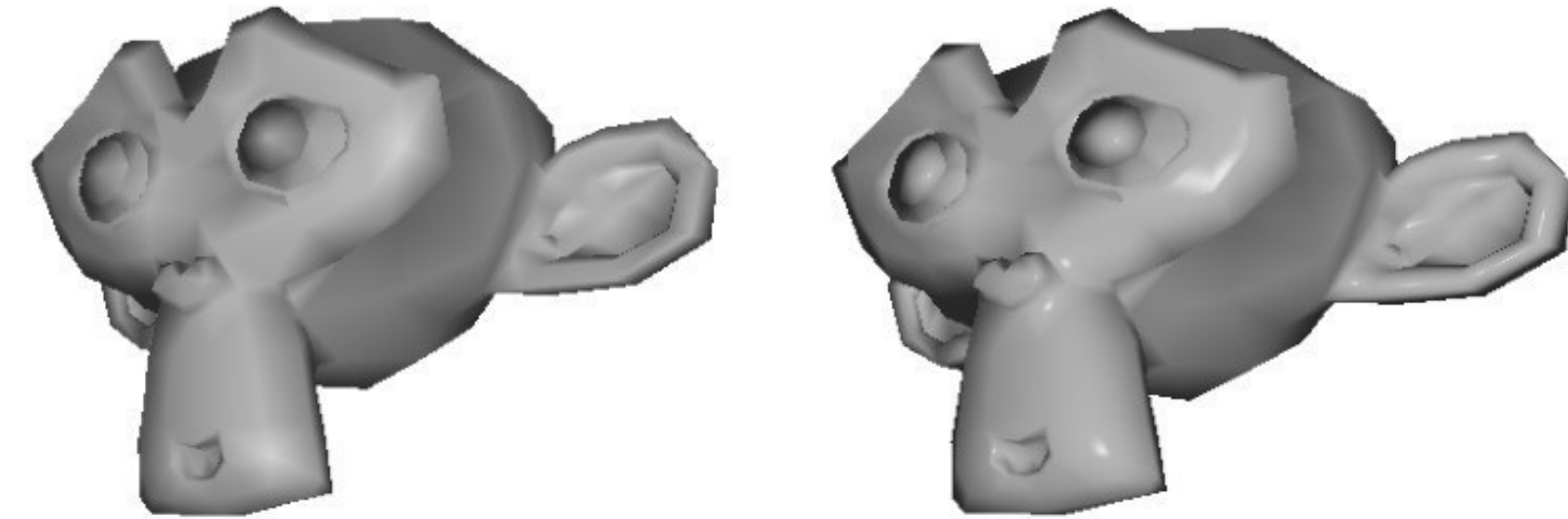
079

Normale par sommet/triangle



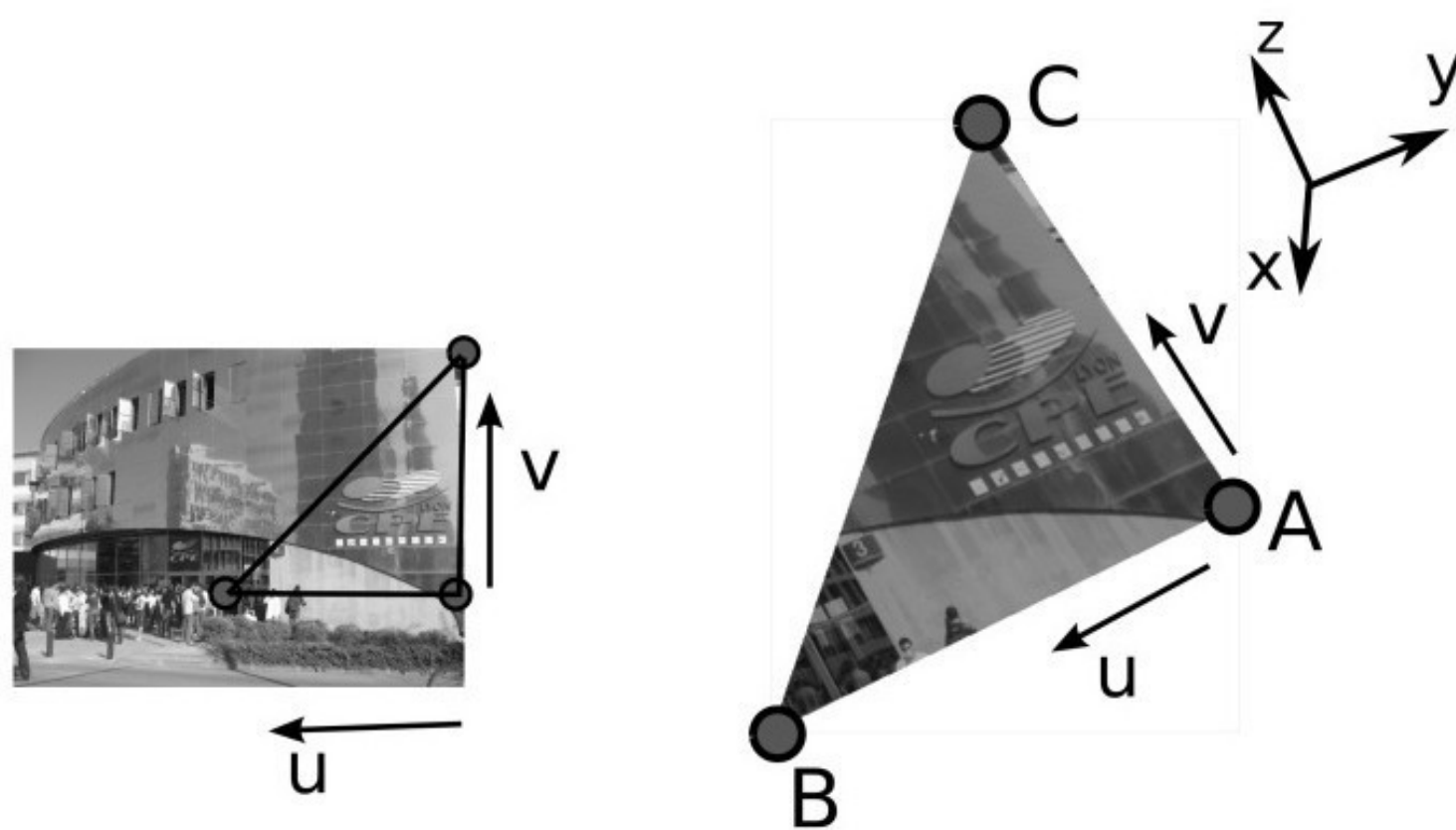
080

Illumination Phong/Gouraud



081

Textures



082

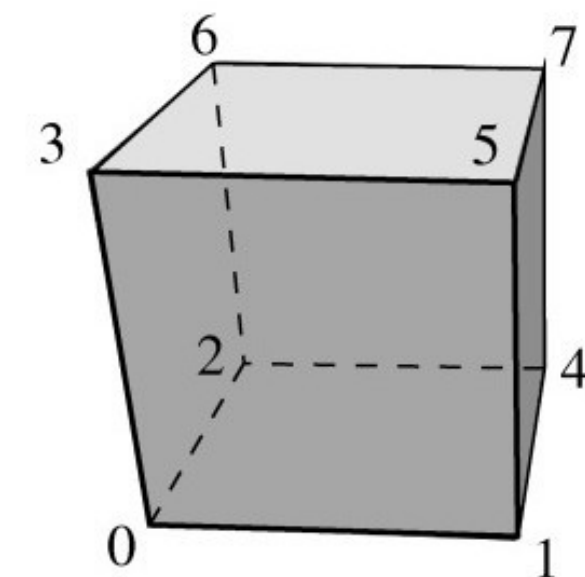
Format de fichiers

```
OFF
8 6 12
0 0 0
1 0 0
0 1 0
0 0 1
1 1 0
1 0 1
0 1 1
1 1 1
4 0 1 4 2
4 1 5 7 4
4 3 6 7 5
4 2 6 3 0
4 2 4 7 6
4 0 3 5 1
```

.off

```
v 0 0 0
v 1 0 0
v 0 1 0
v 0 0 1
v 1 1 0
v 1 0 1
v 0 1 1
v 1 1 1
f 1 2 5 3
f 2 6 8 5
f 4 7 8 6
f 3 7 4 1
f 3 5 8 7
f 1 4 6 2
```

.obj



083