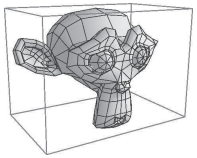
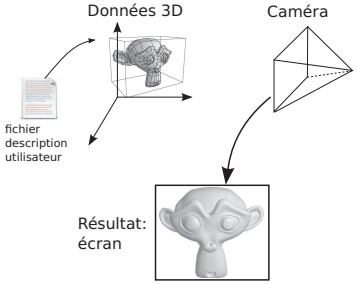
Synthèse d'images



OpenGL

Affichage 3D



Ordre de grandeurs:

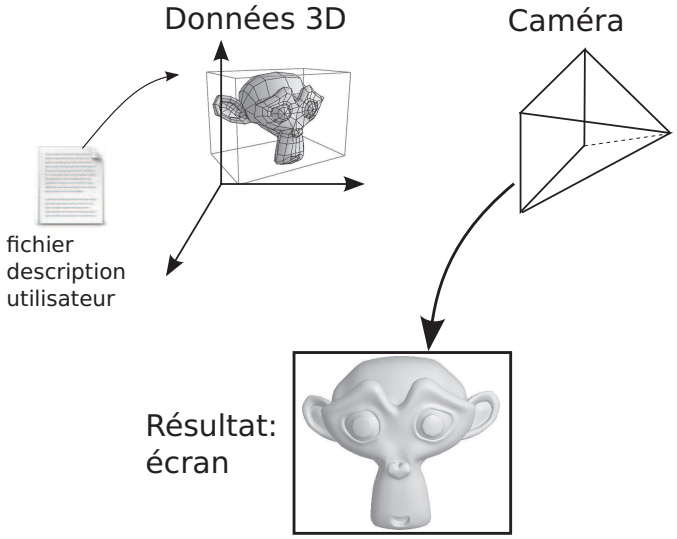
Objets 3D:
100 - 10^5 triangles

1 Triangle ~3000 pixels

25 images par secondes

Affichage 3D = calcul intensif

Affichage 3D



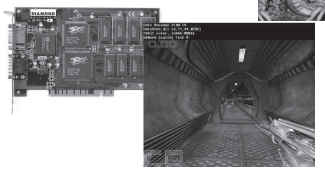
Historique

1ere 3D (années 70-80)

Supercalculateurs

80-90 *Architecture dédiée (3D "cablée")*

Fin 90 *Cartes vidéo 3D*



Historique

Début 2000 Apparition des cartes graphiques

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

Geforce 256

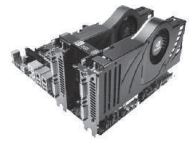
morrowind

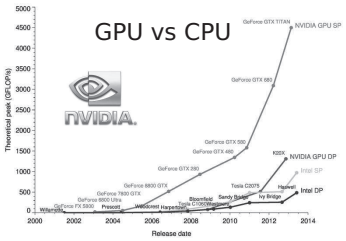
Historique

2013

Language de programmation parallèle.

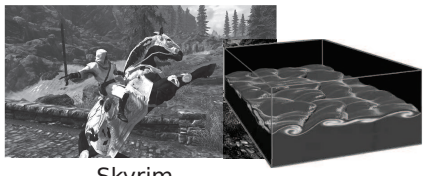
GPGPU





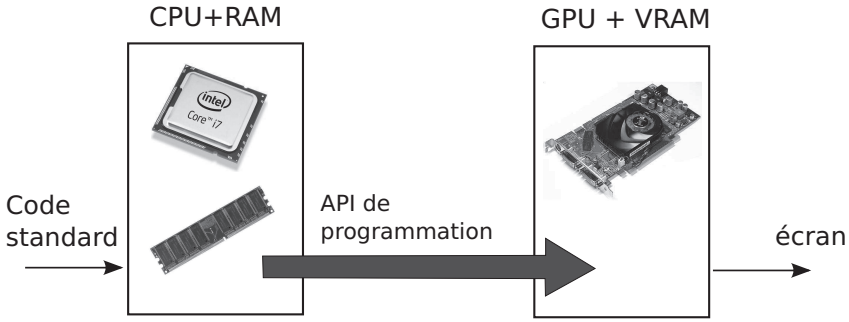
http://michaelgalloy.com/

Crytek Engine



Skyrim

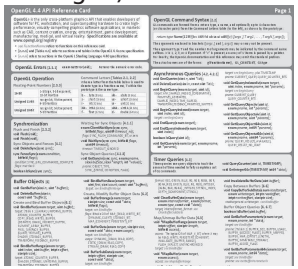
Communication avec GPU



2 API principales:
- OpenGL
- Direct3D

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

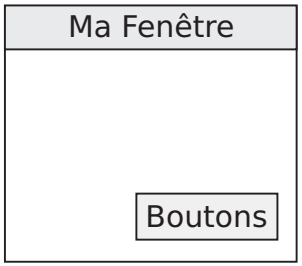
  - Version revendeur: NVIDIA, ATI, Intel
  - Mesa3D (implémentation logicielle libre)
  - OpenGL ES pour Android / iPhone
  - Consoles: Wii, PS3, DS

- Bibliothèque de fonctions C
  Nombreux wrappers

Sources: David Odin

Programmation 3D

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

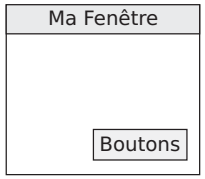


Etape 2:
Démarrer un contexte
OpenGL dans cette fenêtre

Etape 3:
Appeler le code OpenGL
dans une boucle permanente

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 & évènements)
- Qt (gestionnaire générique, très complet, lourd)
- ...

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**

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

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);

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

    return 0;
}
```

récupération touches clavier

Fonction à appeler

Callback GLUT

Autres évènements possibles:

glut**ReshapeFunc**	(redimentionnement de fenêtre)
glut**KeyboardFunc**	(appui clavier)
glut**SpecialFunc**	(touches spéciales du clavier: flêches)
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

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

Appels OpenGL

Principe général

Initialisation

1x

- Définition des données

```
float T[500];
...
T[5]=7.5;
```

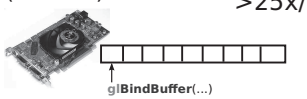
- Envoi des données sur GPU

```
glBufferData(...)
```

Boucle d'affichage

- Pointer vers les données à afficher (sur GPU)

>25x/s



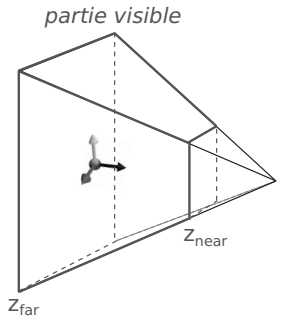
- Demande d'affichage

```
glDrawElements(...)
```

Rendu par projection

Pgm minimal, caméra

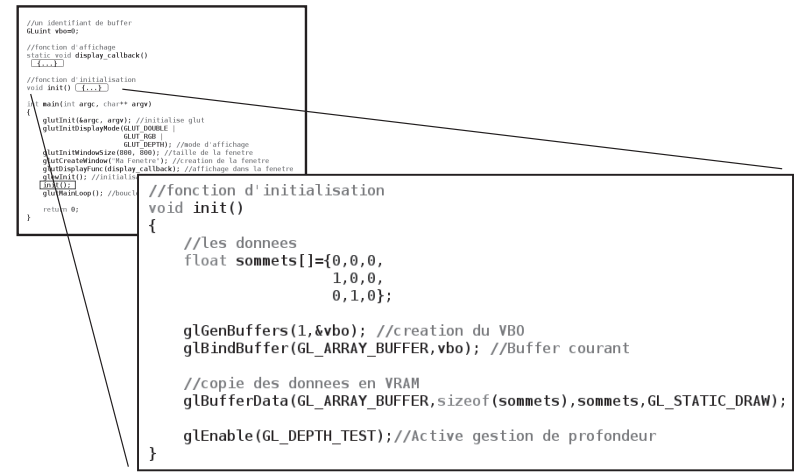
```
//fonction d'affichage
static void display_callback()

{
    //matrices de projections
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(30,1.0,0.1,20.0);

    //matrices du monde
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glTranslatef(0.0f,0.0f,-4.0f);

    //couleur du fond
    glClearColor(0.5, 0.6, 0.9, 1.0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    glutSwapBuffers();
}

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;
}
```



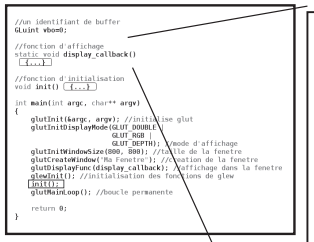
Pgm minimal, triangle

```
//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;
}
```


Pgm minimal, triangle



Pgm minimal, triangle



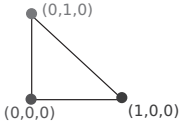
```
//un identifiant de buffer
GLuint vbo=0;

//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;
}
```

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

{
    //matrices de projections
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(30,1.0,0.1,20.0);

    //matrices du monde
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glTranslatef(0.0f,0.0f,-4.0f);

    //couleur du fond
    glClearColor(0.5, 0.6, 0.9, 1.0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    //affichage
    glEnableClientState(GL_VERTEX_ARRAY); //active donnees
    glBindBuffer(GL_ARRAY_BUFFER,vbo);    //buffer courant
    glVertexPointer(3, GL_FLOAT, 0, 0);   //buffer de donnees
    glDrawArrays(GL_TRIANGLES, 0, 3);     //demande d'affichage

    glutSwapBuffers();
}
```

Buffer courant:
vbo

0	0	0	1	0	0	0	1	0

glDrawArrays() Affiche



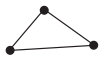
Modes affichages

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

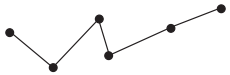
```
glPointSize(5);
glDrawArrays(GL_POINTS,0,N);
```

```
glDrawArrays(GL_LINES,0,N);
```

```
glDrawArrays(GL_LINES_LOOP,0,N);
```



```
glDrawArrays(GL_LINES_STRIP,0,N);
```



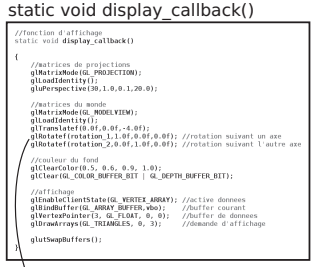
Interaction, rotation

int main()

```
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
    glutSpecialFunc(special_callback);
    glewInit(); //initialisation des fonctions de glew

    init();
    glutMainLoop(); //boucle permanente

    return 0;
}
```

glutSpecialFun(special_callback);

static void display_callback()



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

{
    //matrices de projections
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(30,1.0,0.1,20.0);

    //matrices du monde
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glTranslatef(0.0f,0.0f,-4.0f);
    glRotatef(rotation_1,1.0f,0.0f,0.0f); //rotation suivant un axe
    glRotatef(rotation_2,0.0f,1.0f,0.0f); //rotation suivant l'autre axe

    //couleur du fond
    glClearColor(0.5, 0.6, 0.9, 1.0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    //affichage
    glEnableClientState(GL_VERTEX_ARRAY); //active donnees
    glBindBuffer(GL_ARRAY_BUFFER,vbo);    //buffer courant
    glVertexPointer(3, GL_FLOAT, 0, 0);   //buffer de donnees
    glDrawArrays(GL_TRIANGLES, 0, 3);     //demande d'affichage

    glutSwapBuffers();
}
```

glRotatef(rotation_1, 1,0,0);
glRotatef(rotation_2, 0,1,0);

Variables globales

```
//donnees de rotation
int rotation_1=0;
int rotation_2=0;

static void special_callback(int key, int x_mouse,int y_mouse)
{
    switch (key)
    {
    case GLUT_KEY_UP:
        rotation_1+=3; //rotation avec la touche du haut
        break;
    case GLUT_KEY_DOWN:
        rotation_1-=3; //rotation avec la touche du bas
        break;
    case GLUT_KEY_LEFT:
        rotation_2+=3; //rotation avec la touche de gauche
        break;
    case GLUT_KEY_RIGHT:
        rotation_2-=3; //rotation avec la touche de droite
        break;
    }

    //reactualisation de l'affichage
    glutPostRedisplay();
}
```

Rem. fonctions OpenGL

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

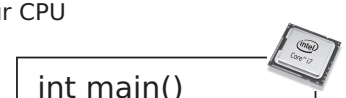
gl[NOM]f(...);
gl[NOM]i(...);
gl[NOM]u(...);
gl[NOM]d(...);
gl[NOM]vf(...);

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

ex.
glTranslatef(...)
glUniform1f(...)

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

Shaders en pratique

Créez 2 fichiers *textes*:

```
#version 120

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

shader.vert

```
#version 120

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

shader.frag

Chargez les fichiers dans le programme principal

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

Shaders en pratique

```
#version 120

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

shader.vert

Ressemble
à du C

Applique transformation
projective & monde

Variable déjà connue
position de sortie des sommets

Shaders en pratique

```
#version 120

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

shader.frag

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

Variable de sortie:
couleur du pixel

Contrôle
de la couleur

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();

Principe interpolation

Passage d'arguments varying

```
#version 120

varying vec4 position3d;

//Un Vertex Shader minimaliste
void main (void)
{
    //Projection du sommet
    gl_Position = ftransform();
    position3d=gl_Vertex;
}
```

interpolation

```
#version 120

varying vec4 position3d;

//Un Fragment Shader minimaliste
void main (void)
{
    //couleur du fragment
    gl_FragColor = position3d;
}
```

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()

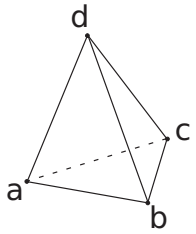
```
uniform float valeur;
varying vec4 position3d;

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

Passage d'arguments uniform

Utilisation: - Interaction clavier, souris
- Temps

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

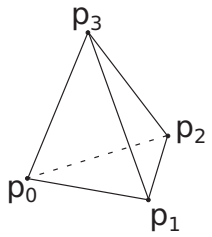
Maillages




```
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

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

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);
```

Entrelacement de données

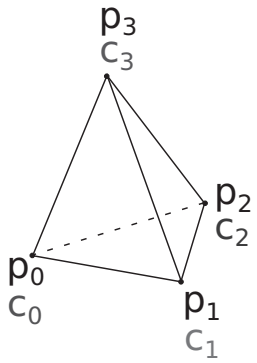
On souhaite envoyer au GPU:

4 Sommets

- géométrie (x,y,z)
- couleur (r,g,b)

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

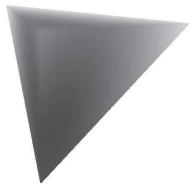
3 floats 3 floats



Entrelacement de données

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,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);
```



Display

```
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);
```


Entrelacement de données

Init

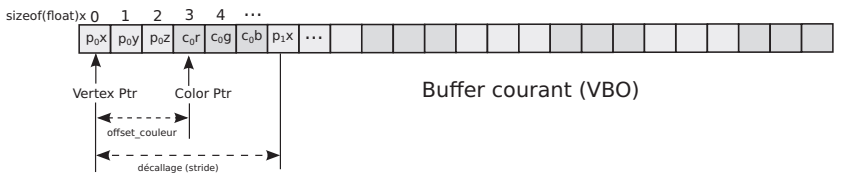
```
vec3 p0(0,0,0); vec3 c0(1,0,0);
vec3 p1(1,0,0); vec3 c1(0,1,0);
vec3 p2(0,1,0); vec3 c2(0,0,1);
vec3 p3(0,0,1); vec3 c3(0,1,1);
vec3 vertex[]={p0,c0 , p1,c1 , p2,c2 , p3,c3};

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);
```


Display

```
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);
```

position (float)

couleur (float)

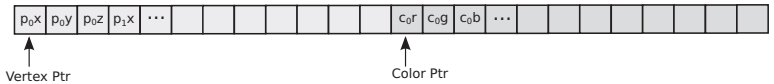


Autre possibilité

```
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



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,&vboi);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER,vboi);
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);
```

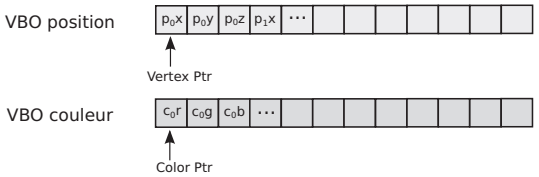

Autre possibilité

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

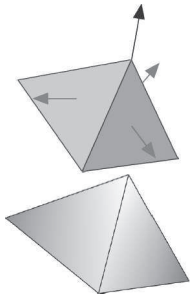
Display

```
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);
```



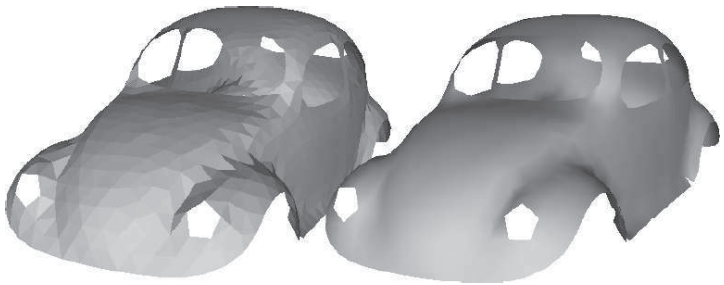
Illumination

Normale par sommet/triangle

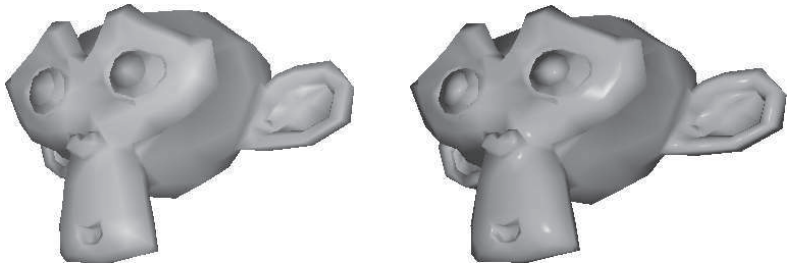


$$\mathbf{n}_k = \frac{\sum_{i \in \mathcal{V}(k)} \mathbf{n}_i}{\left\| \sum_{i \in \mathcal{V}(k)} \mathbf{n}_i \right\|}$$

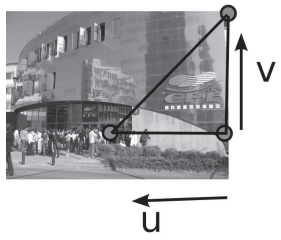
k : indice sommet
i : indice face
$\mathcal{V}(k)$: faces voisines du sommet k

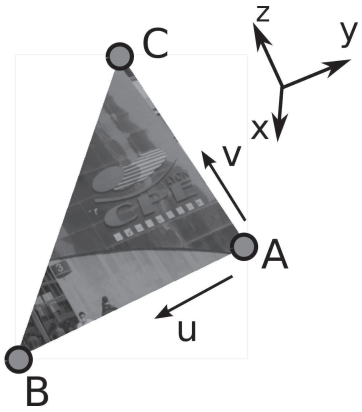


Illumination Phong/Gouraud



Textures





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