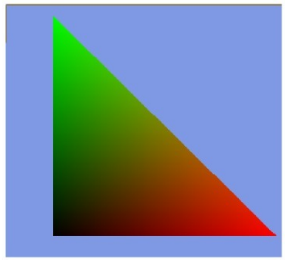
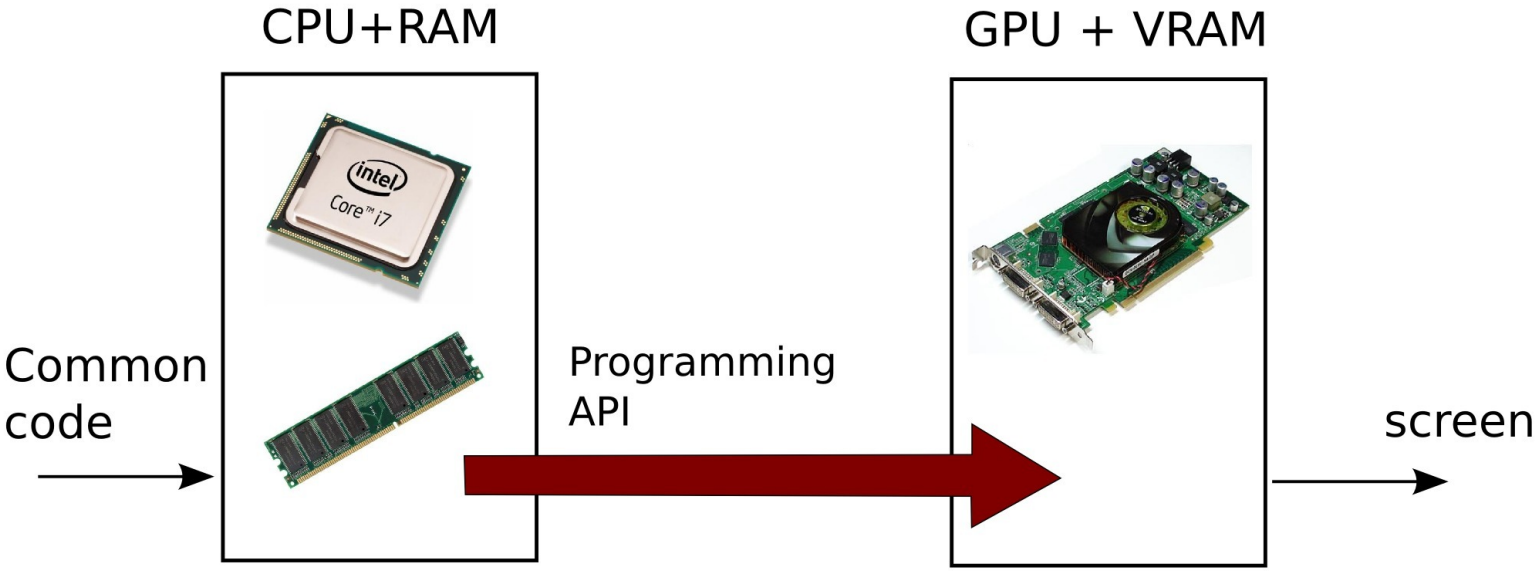
```
glUniformMatrix4fv(get_uni_loc(shader_program_id,"rotation"),
                   1,false,pointeur(rotation));
glUniform4f(get_uni_loc(shader_program_id,"translation"),
            translation_x,translation_y,translation_z,0.0f);
```

OpenGL



```
#version 120

// A simple fragment shader
void main (void)
{
     // The color of the fragment
     gl_FragColor = vec4(1.0f,0.0f,0.0f,1.0f);
}
```

Communicating with the GPU

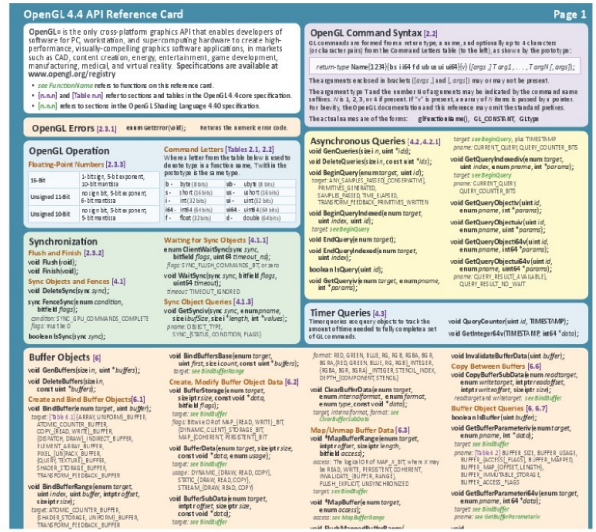


2 main API:

- **OpenGL**
- **Direct3D**

OpenGL

- A set of **specifications** as an API to communicate with the graphics card.



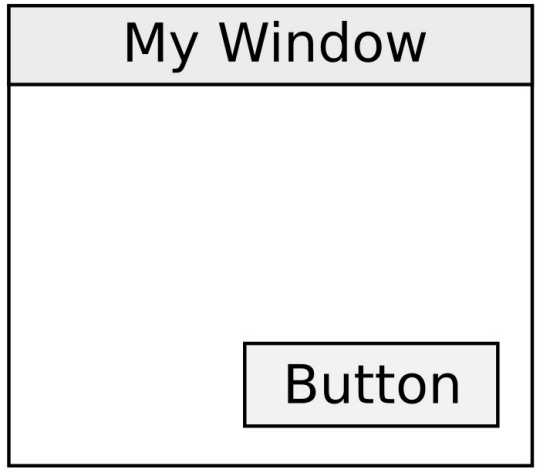
- Has multiple implementations depending on the graphics card and the drivers.
  - Proprietary: NVIDIA, ATI, Intel
  - Mesa3D (free software implementation)
  - OpenGL ES for Android / iPhone
  - Gaming Plateform: Wii, PS3, DS

- Library of C functions
  Various wrappers

[Sources: David Odin]

3D Programming

1st Step:
Draw a window
(indépendant from OpenGL)

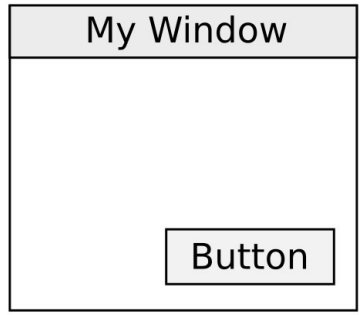


2nd Step:
Start a context
OpenGL within this window

3rd Step:
Call OpenGL calls
within a permanent loop

3D Programming

1st Step:
Draw a window
(independant from OpenGL)



We need a window manager & event manager

Different choices:

- Glut (simple, light, limited)
- SDL (specialized for small games)
- GLFW (light, recent)
- GTK (generic window & event manager)
- Qt (generic manager, very complete, heavy)
- ...

Using GLUT

GLUT - The OpenGL Utility Toolkit

A Window manager dedicated to 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;
}
```

To compile : g++ pgm.c -lGL -lGLU -lglut -lGLEW

Under Linux: Need to install the packages **freeglut, Glew**

Drawing Functions

To draw something

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

glNAME : OpenGL function

glClearColor(r,g,b,a) : Color of the background of the screen

glClear() : clear the screen (+ depth buffer)

glutSwapBuffers() : Swap the drawing/temp buffer

glut specific function

Callback GLUT

```c
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); // Initialize glut
    glutInitDisplayMode(GLUT_DOUBLE |
                        GLUT_RGB |
                        GLUT_DEPTH); // Drawing mode
    glutInitWindowSize(800, 800);     // Init window size
    glutCreateWindow("Ma Fenetre");  // Draw the window

    glutDisplayFunc(display_callback);
    glutKeyboardFunc(keyboard_callback);

    glewInit();   // Init glew functions
    glutMainLoop(); // Permanent loop

    return 0;
}
```

Get the keyboard pressed key

Function to call

Callback GLUT (mouse)

Get mouse click

```
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);
```

Callback functions = function called when an event occurs

Callback GLUT

Other possible events:

glut**ReshapeFunc**	(window resizing)
glut**KeyboardFunc**	(press a key on the keyboard)
glut**SpecialFunc**	(special keys: arrows)
glut**MouseFunc**	(mouse click)
glut**MotionFunc**	(mouse motion)
glut**TabletButtonFunc**	(use of a tablet)
glut**IdleFunc**	(when nothing is happening)
glut**TimerFunc**	(function called after a given time)

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

OpenGL Called

General principle

Initialization

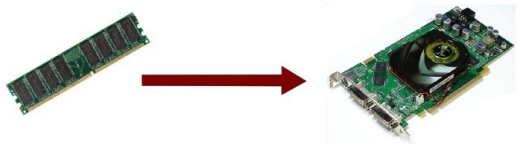
1x

- Data setup

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

- Send data to GPU

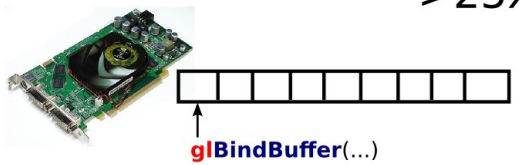
```
glBufferData(...)
```



Drawing loop

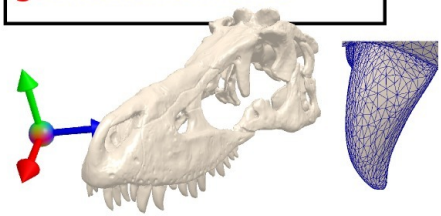
- setup pointers on data to draw (on the GPU)

>25x/s



- Drawing request

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



Minimal program
Drawing a triangle

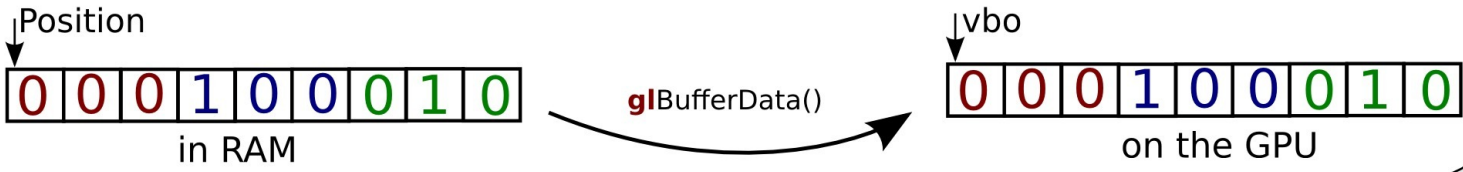
Pgm minimal, triangle

```
// Id for a buffer on the GPU
GLuint vbo=0;

// Drawing function
static void display_callback()
 {...}

// Init function
void init() {...}

int main(int argc, char** argv)
{
    glutInit(&argc, argv); // Initialize glut
    glutInitDisplayMode(GLUT_DOUBLE |
                        GLUT_RGB |
                        GLUT_DEPTH);  // Drawing mode
    glutInitWindowSize(800, 800);     // Init window size
    glutCreateWindow("Ma Fenetre");   // Draw the window
    glutDisplayFunc(display_callback);
    glewInit(); // Init glew functions
    init();
    glutMainLoop(); // Permanent loop

    return 0;
}
```

Pgm minimal, triangle

```
// Id for a buffer on the GPU
GLuint vbo=0;

// Drawing function
static void display_callback()
 {...}

// Init function
void init() {...}

int main(int argc, char** argv)
{
    glutInit(&argc, argv); // Initialize glut
    glutInitDisplayMode(GLUT_DOUBLE |
                        GLUT_RGB |
                        GLUT_DEPTH);  // Drawing mode
    glutInitWindowSize(800, 800);     // Init window size
    glutCreateWindow("Ma Fenetre");   // Draw the window
    glutDisplayFunc(display_ca
    glewInit(); // Init glew function
    init();
    glutMainLoop(); // Permanent

    return 0;
}
```

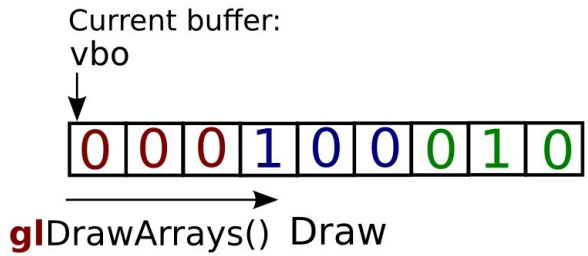
```
// Init function
void init()
{
    // Data
    float sommets[]={0,0,0,
                     1,0,0,
                     0,1,0};

    glGenBuffers(1,&vbo);  // Create ID for a VBO on the GPU
    glBindBuffer(GL_ARRAY_BUFFER,vbo);  // Set up the current buffer

    // Copy data from the RAM memory to the GPU memory
    glBufferData(GL_ARRAY_BUFFER,sizeof(sommets),sommets,GL_STATIC_DRAW);

    glEnable(GL_DEPTH_TEST); // Activate the management of the depth
}
```

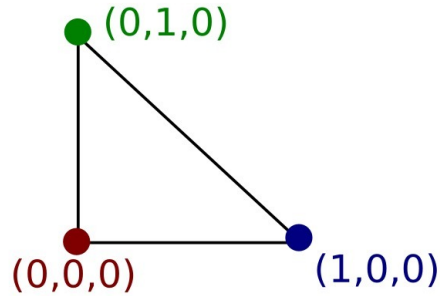


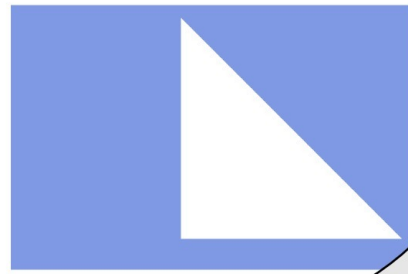
Pgm minimal, triangle

```
// Id for a buffer on the GPU
GLuint vbo=0;

// Drawing function
static void display_callback()
 {...}

// Init function
void init() {...}

int main(int argc, char** argv)
{
    glutInit(&argc, argv); // Initialize glut
    glutInitDisplayMode(GLUT_DOUBLE |
                        GLUT_RGB |
                        GLUT_DEPTH);  // Drawing mode
    glutInitWindowSize(800, 800);     // Init window size
    glutCreateWindow("Ma Fenetre");   // Draw the window
    glutDisplayFunc(display_callback);
    glewInit(); // Init glew functions
    init();
    glutMainLoop(); // Permanent loop

    return 0;
}
```

```
// Drawing function
static void display_callback()

{
    // Background color
    glClearColor(0.5, 0.6, 0.9, 1.0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    // Drawing
    glEnableClientState(GL_VERTEX_ARRAY);  // Activate data management
    glBindBuffer(GL_ARRAY_BUFFER,vbo);     // Current buffer
    glVertexPointer(3, GL_FLOAT, 0, 0);    // Set up data buffer
    glDrawArrays(GL_TRIANGLES, 0, 3);      // Request drawing

    glutSwapBuffers();
}
```



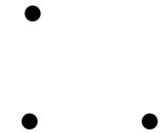




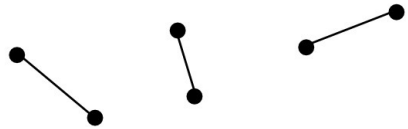
Drawing modes

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

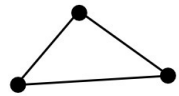
```
glPointSize(5);
glDrawArrays(GL_POINTS,0,N);
```



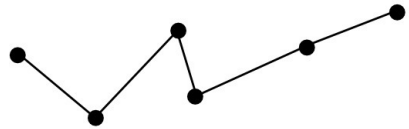
```
glDrawArrays(GL_LINES,0,N);
```



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



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



Introduction to Shaders

Shaders

Programming in C, C++, etc
on the CPU


```
int main()
{ ...

  Create data
  Send data on GPU
  Request drawing
  Event management
```

On the GPU

2 programs

Vertex Shader

Called for every vertices

Project position

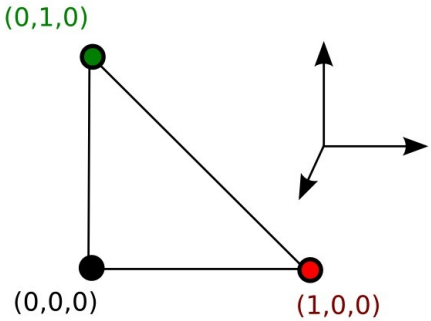
Fragment Shader

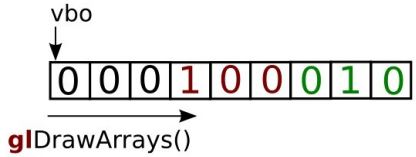
Called for every colored pixel

Color of the pixel

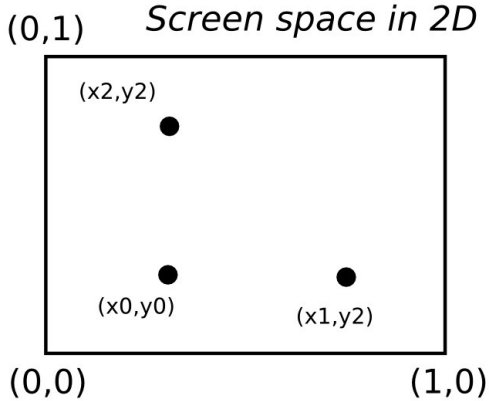
Goal of the shaders

3D Data





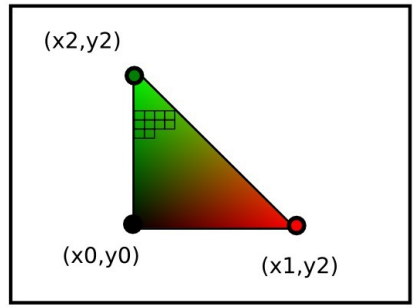
Vertex Shader



Input:
3D Coordinate

Output:
2D coordinate in screen space

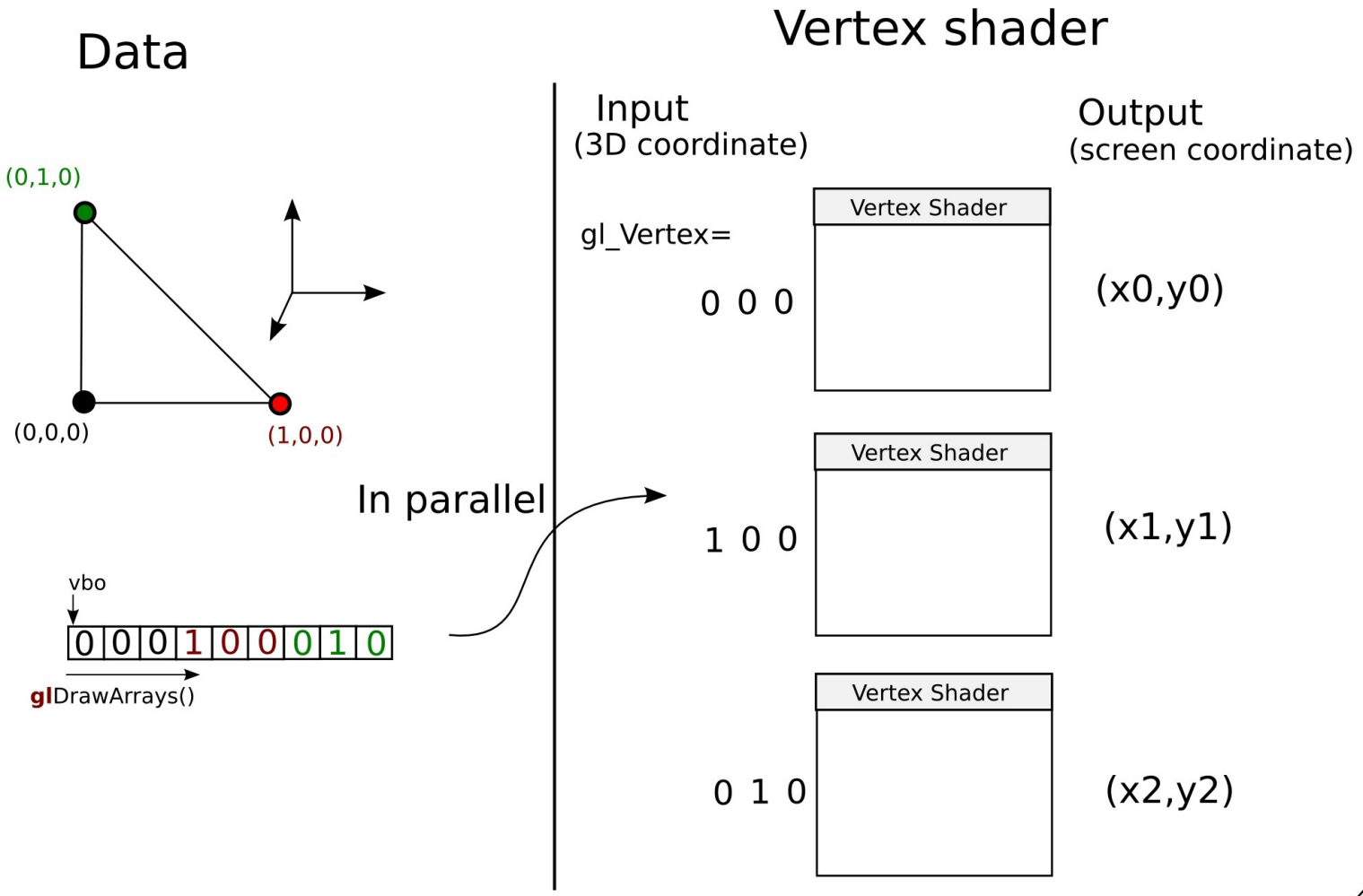
Fragment Shader



Input:
coordinate of the current fragment
+ linear interpolation on the data

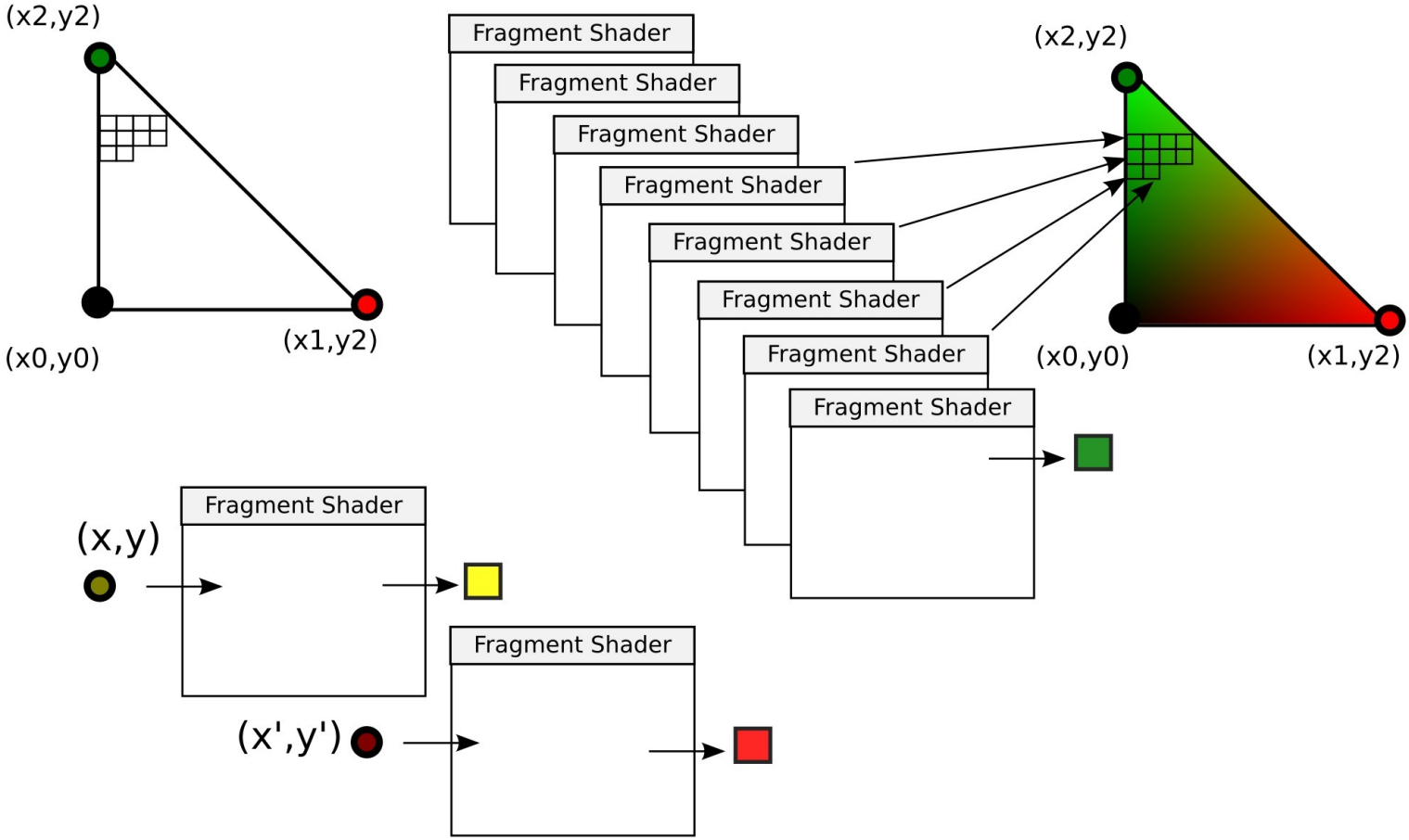
Output:
color of the fragment

Parallelism



Parallelism

Fragment shader



Shaders in practice

Create 2 text files

```
#version 120

// A simple vertex shader
void main (void)
{
     // Vertex coordinate
     gl_Position = gl_Vertex;
}
```

shader.vert

```
#version 120

// A simple fragment shader
void main (void)
{
     // The color of the fragment
     gl_FragColor = vec4(1.0f,0.0f,0.0f,1.0f);
}
```

shader.frag

Load the files in the main program

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

Shaders in practice

```
#version 120

// A simple fragment shader
void main (void)
{
      // Vertex coordinate
     gl_Position = gl_Vertex;
}
```

shader.vert

Looks like C

Variable already known
output vertex coordinate in screen space

Variabl already known
input vertex coordinates in 3D space

Shaders in practice

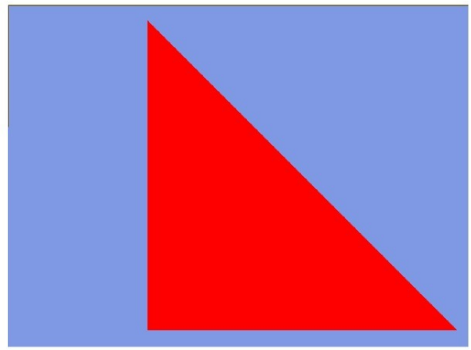
```
#version 120

// A simple fragment shader
void main (void)
{
    // The color of the fragment
    gl_FragColor = vec4(1.0f,0.0f,0.0f,1.0f);
}
```

shader.frag

This is not C
Looks like C++
=> This is **GLSL**

Output variable:
color of the pixel

Color



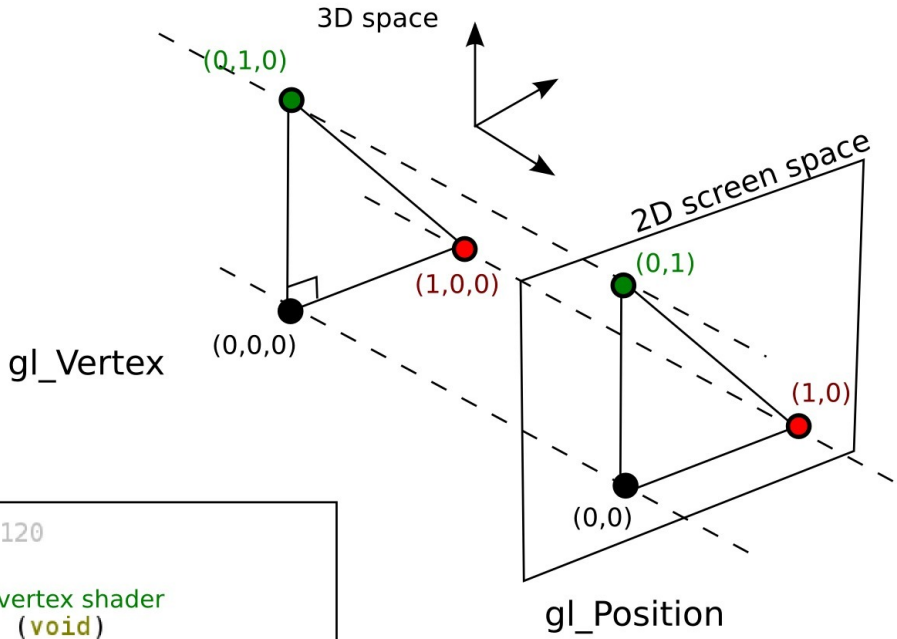
GLSL

Language similar to C and C++ but simpler.
Provides existing structures to perform 3D geometry computations.

```
int
float

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

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

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

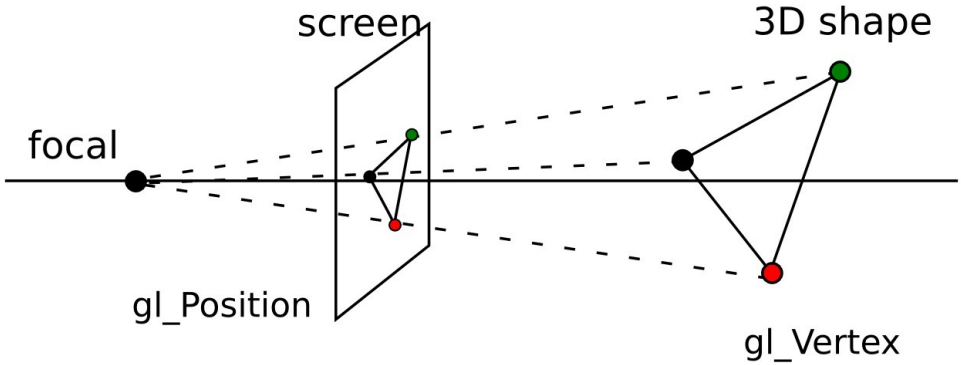
Projection

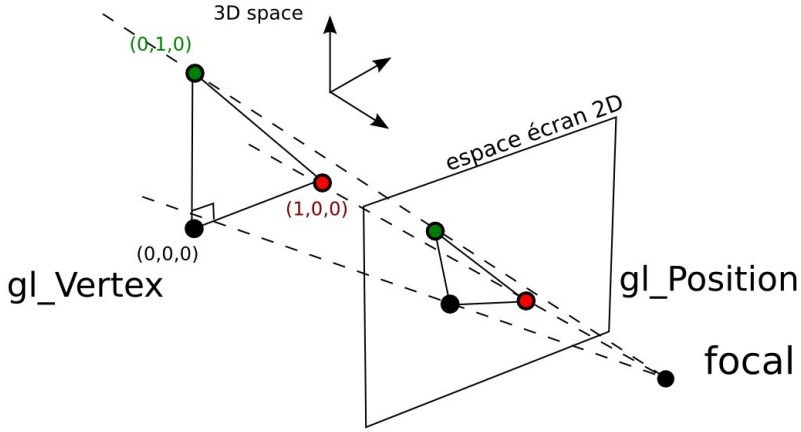



```
#version 120

// A simple vertex shader
void main (void)
{
     // Vertex coordinate
     gl_Position = gl_Vertex;
}
```

Orthogonal projection

Projection



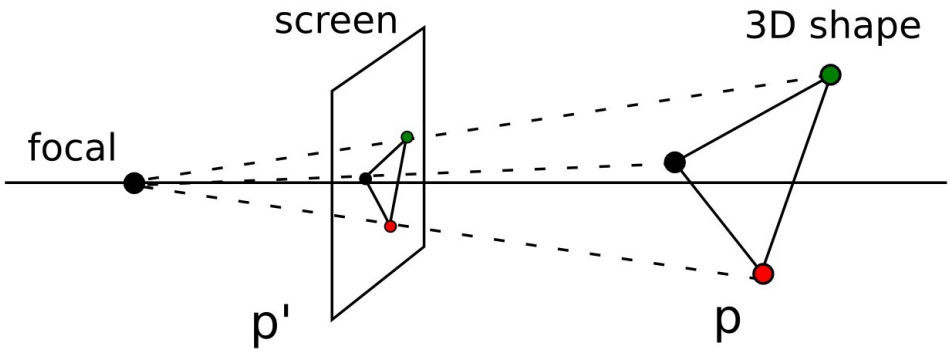


gl_Position=perspective_projection(gl_Vertex)

Perspective projection

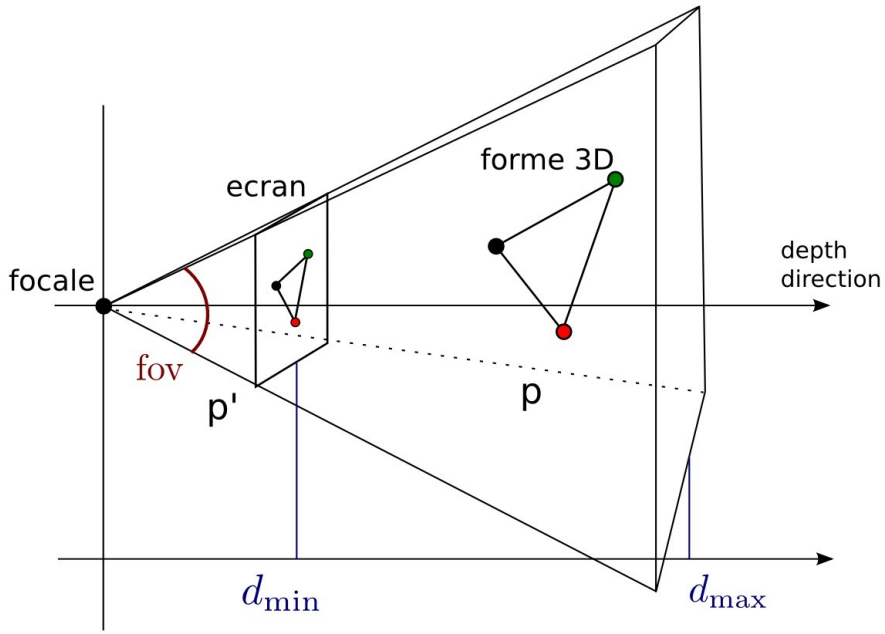
gl_Position=perspective_projection(gl_Vertex)

projection_perspective : Projection matrix 4x4



$$p' = M\ p$$

Perspective projection



a: aspect ratio y/x

$$p' = M\, p$$

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

$$f = \mathrm{cotan}(\mathrm{fov}/2)$$

$$f_x = f/a$$

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

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

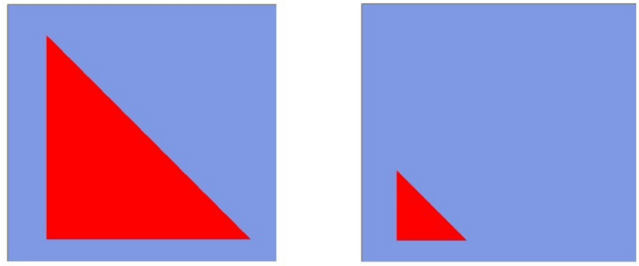
Perspective projection

Exemple of 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));


void main (void)
{

    vec4 p=projection*gl_Vertex;

    // position in screen space
    gl_Position = p;
}
```

Warning:
Column vectors

fov=60 degrees
a=1
d_min=0.5
d_max=10



z=2 z=6

Passing uniform arguments

We can pass parameters from CPU to GPU

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;



void main (void)
{
     // fragment color
     gl_FragColor = vec4(valeur,valeur,0,1);
}
```

uniform:
parameter have the same value for all the shaders.

Passing uniform arguments

Usage: - Keyboard interaction, mouse
- Time

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

Rem. OpenGL Functions

The OpenGL API is in C : 1 name per argument type

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

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

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

Interaction

Can translate an object in the shader

Glut

Keyboard event

```
If arrow 'up':
  Then add translation
```

Drawing function

Send data translation to the graphics card

```
glUniform()
```

CPU

Vertex Shader

Translate vertex

```
uniform translation;

gl_Vertex+translation;
```

GPU

Interaction

Translating an object in the shader

```
static void special_callback(int key, int,int)
{
    float dL=0.01f;
    switch (key)
    {
    case GLUT_KEY_UP:
        translation_y+=dL;
        break;
    case GLUT_KEY_DOWN:
        translation_y-=dL;
        break;
    case GLUT_KEY_LEFT:
        translation_x-=dL;
        break;
    case GLUT_KEY_RIGHT:
        translation_x+=dL;
        break;
    }

    // Request drawing
    glutPostRedisplay();
}
```

Keyboard interaction

Vertex Shader

```
#version 120

uniform vec4 translation;


void main (void)
{

    vec4 p=gl_Vertex+translation;
    gl_Position = p;
}
```

```
glUniform4f(get_uni_loc(shader_program_id,"translation"),
            translation_x,
            translation_y,
            translation_z,0.0f);
glDrawArrays(GL_TRIANGLES, 0, 3);
```

Drawing

Interaction

Object can also be rotating in the shader

```
glUniformMatrix4fv(get_uni_loc(shader_program_id,"rotation"),
                   1,false,pointeur(rotation));
glUniform4f(get_uni_loc(shader_program_id,"translation"),
            translation_x,translation_y,translation_z,0.0f);
```

```
uniform vec4 translation;
uniform mat4 rotation;



void main (void)
{
     // rotation followed by translation
     vec4 p=rotation*gl_Vertex+translation;


     gl_Position = p;
}
```

Interaction

Different object can receive different rotation/translation

Principle:

Send unifor data (translation1/rotation1) from the main pgm
Request drawing of object 1

Send unifor data (translation2/rotation2) from the main pgm
Request drawing of object 2

...

Interaction

A translation/rotation applied to one object and not to the others
=> Displacement of the object

A translation/rotation applied to every objects
=> Displacement of the camera

There is no differences between the motion of a 'camera' and moving every objects of the scene.

p'=rotation*p+translation

Vertex shader + deformation

```
#version 120

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


void main (void)
{

    vec4 p=rotation*gl_Vertex+translation;
    p=projection*p;

    // position in the screen space
    gl_Position = p;
}
```

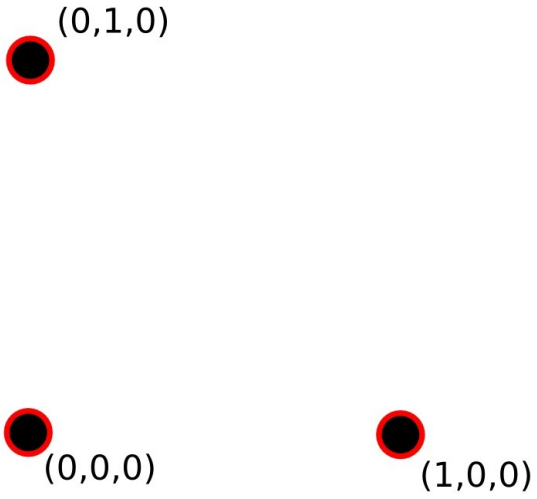
Received from the main program

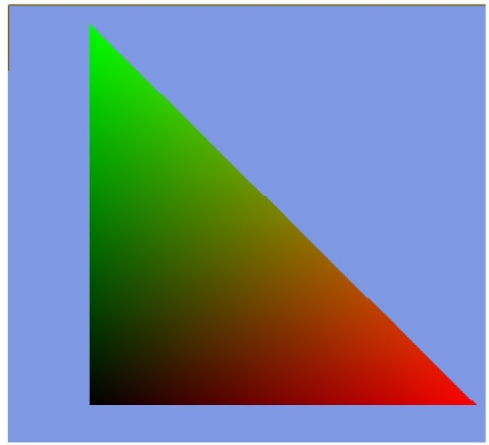
Varying arguments (OpenGL 2)

```
#version 120

varying vec4 position3d;


void main (void)
{

    gl_Position = ftransform();
    position3d=gl_Vertex;
}
```

interpolation

```
#version 120

varying vec4 position3d;


void main (void)
{

    gl_FragColor = position3d;
}
```





Varying arguments

```
#version 120

varying vec4 position3d;


void main (void)
{

    gl_Position = ftransform();
    position3d=gl_Vertex;
}
```

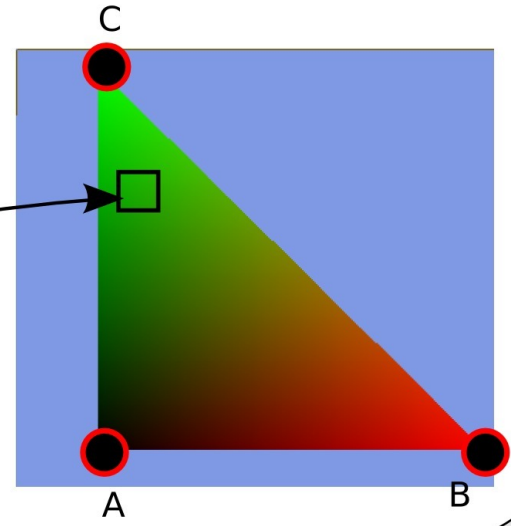
interpolation

```
#version 120

varying vec4 position3d;


void main (void)
{

    gl_FragColor = position3d;
}
```

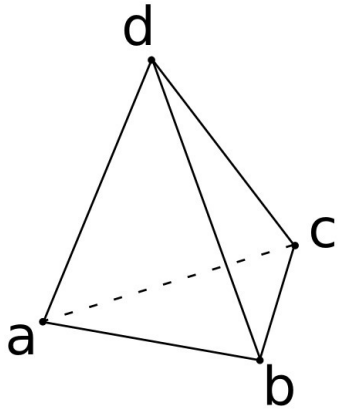
Barycentric interpolation (linear)
automatically computed by the graphics card:

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

for all the fragments



Mesh

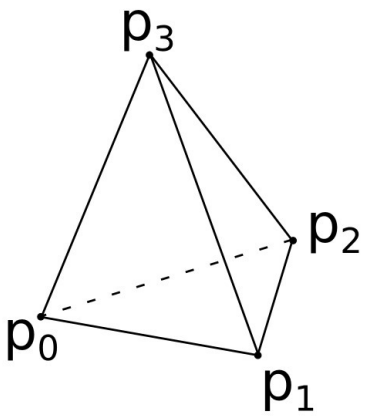



```
float vertex[]=
{xa,ya,za , xb,yb,zb , xd,yd,zd,
 xb,yb,zb , xc,yc,zc , xd,yd,zd,
 xc,yc,zc , xd,yd,zd , xa,ya,za,
 xa,ya,za , xb,yb,zb , xc,yc,zc};
```

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

Repetition:
- waste of memory
- complex to change one vertex

Mesh: indexed connectivity



Separate:
3D geometry / connectivity

```
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 of buffers on the GPU

Geometry buffer

p_0x p_0y p_0z p_1x ...

float

Index buffer

0 1 3 1 2 3 0 ...

unsigned int

+ Coordinates are written only 1 times

Indexed format

Initialization

```
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);
```

Drawing

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

Data interleaving

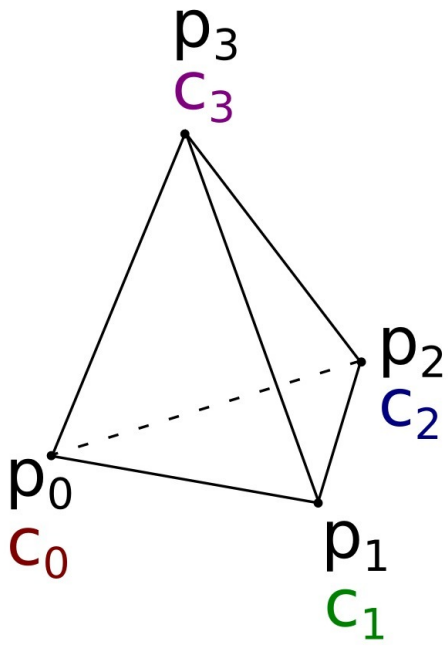
We want to send to the GPU:

4 Vertices

- geometry (x,y,z)
- color (r,g,b)

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

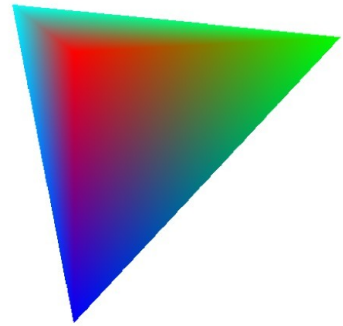
3 floats 3 floats



Data interleaving

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);
```

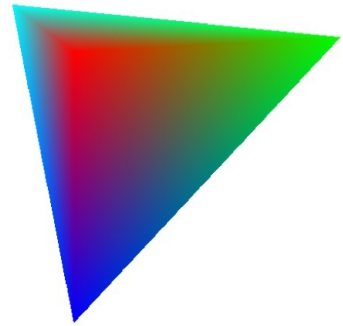
Data interleaving

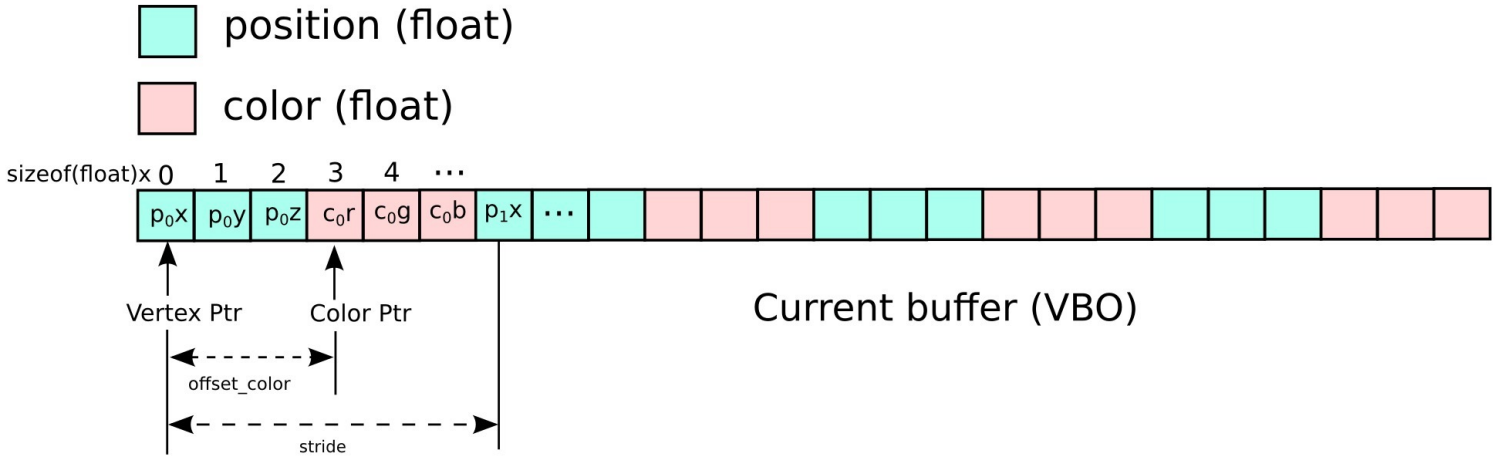
Init

```
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);
```



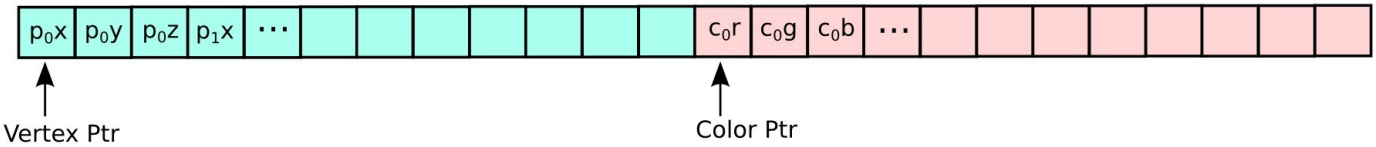


Other possibility

```
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);
```

Current VBO



Other possibility

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);
```

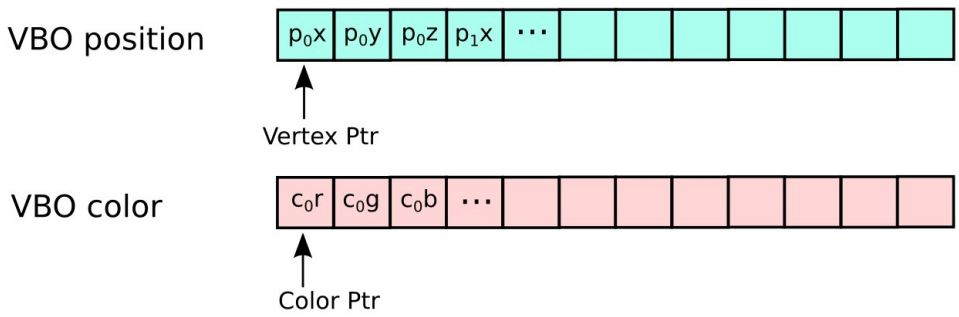
Other possibility

Init

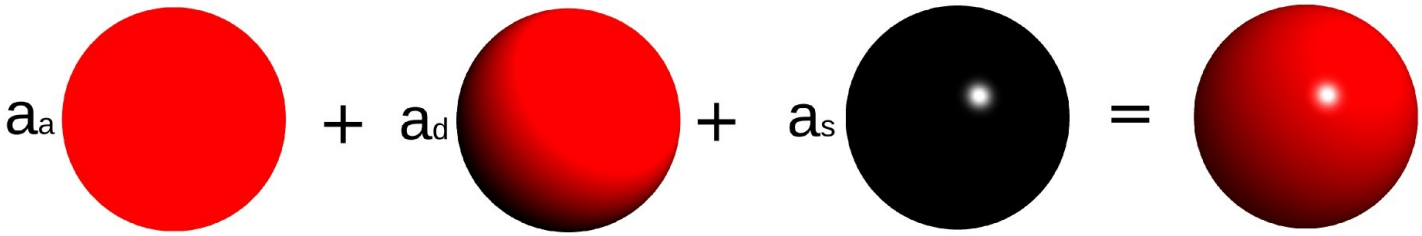
```
vec3 p0(0,0,0); vec3 c0(1,0,0);
vec3 p1(1,0,0); vec3 c1(0,1,0);
vec3 p2(0,1,0); vec3 c2(0,0,1);
vec3 p3(0,0,1); vec3 c3(0,1,1);
vec3 vertex[]={p0,p1,p2,p3};
vec3 color[] ={c0,c1,c2,c3};

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);
```



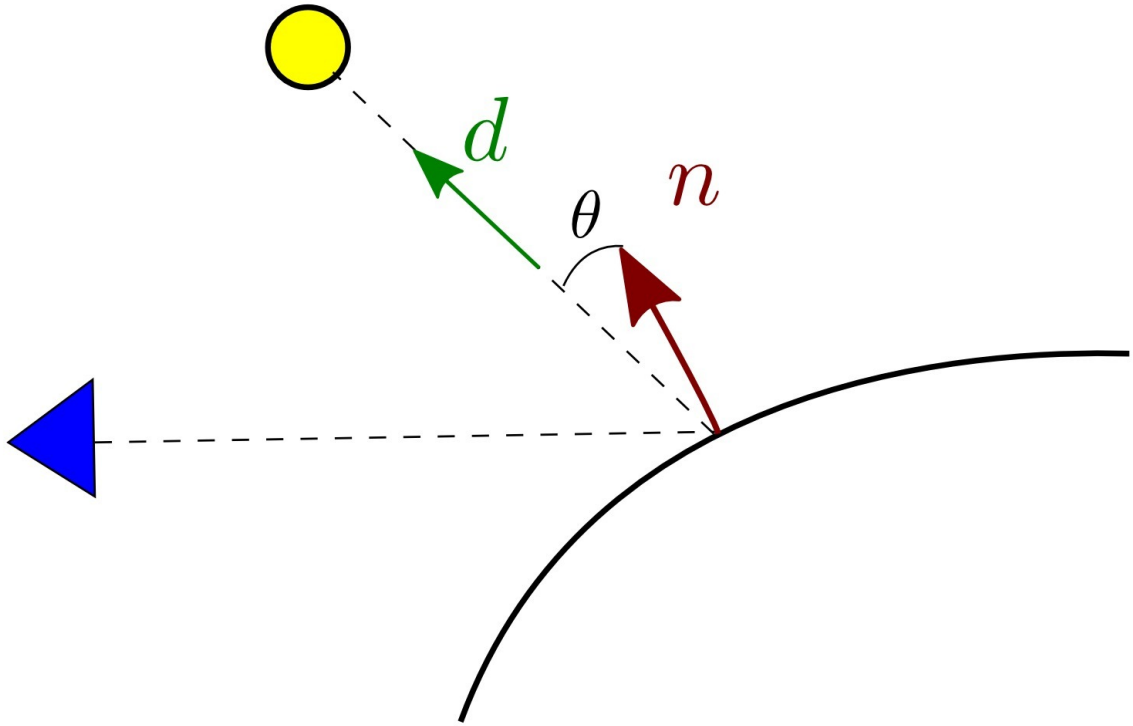
Shading



ambiant diffuse specular

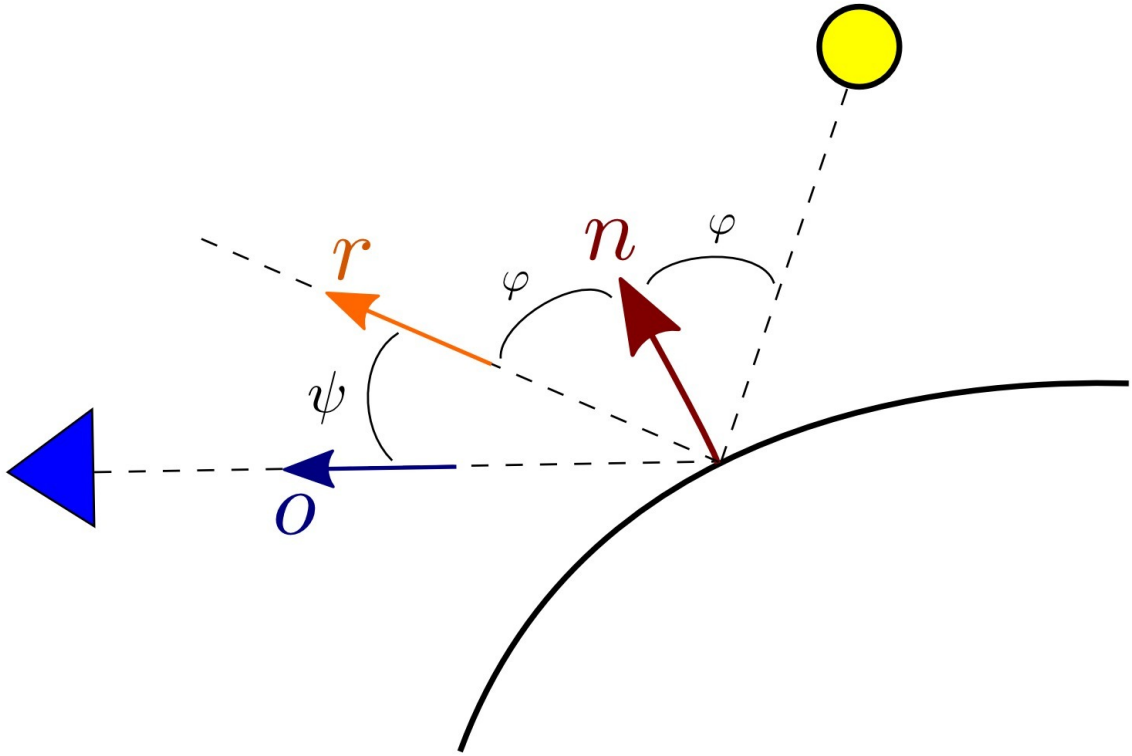
Shading

Diffuse coefficient $c_d = \cos(\theta)$

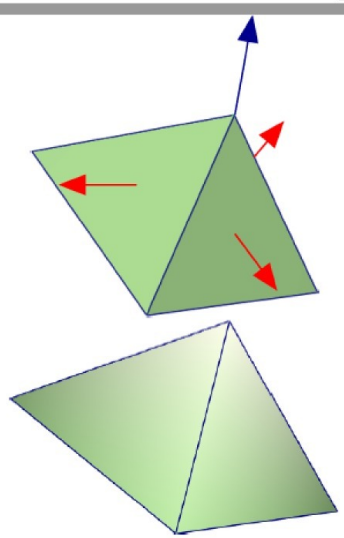


Shading

Specular coefficient $c_s = \cos(\psi)^n$

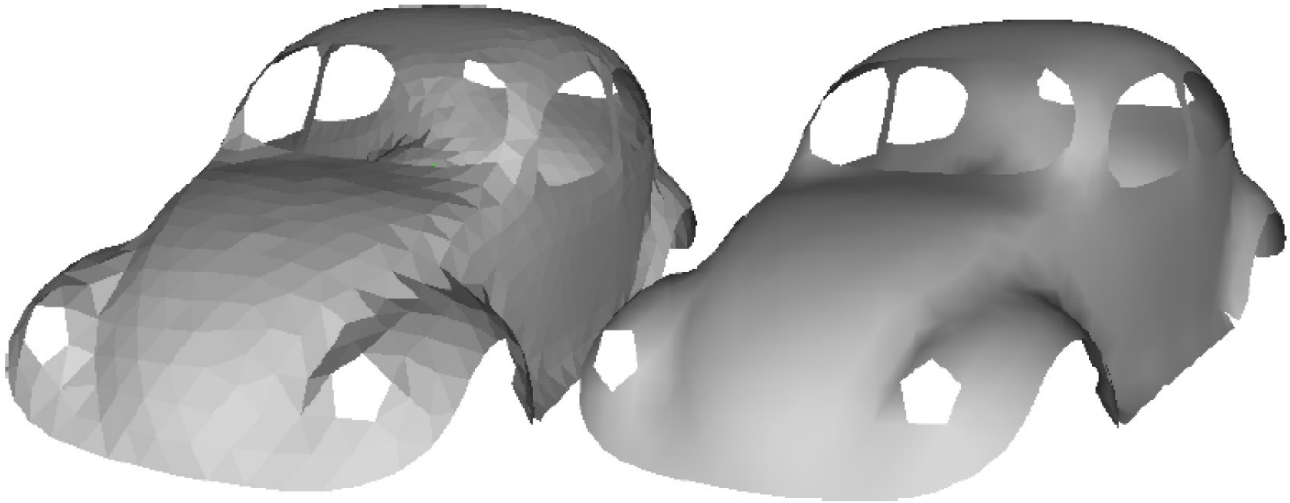


Normals per vertex/triangle

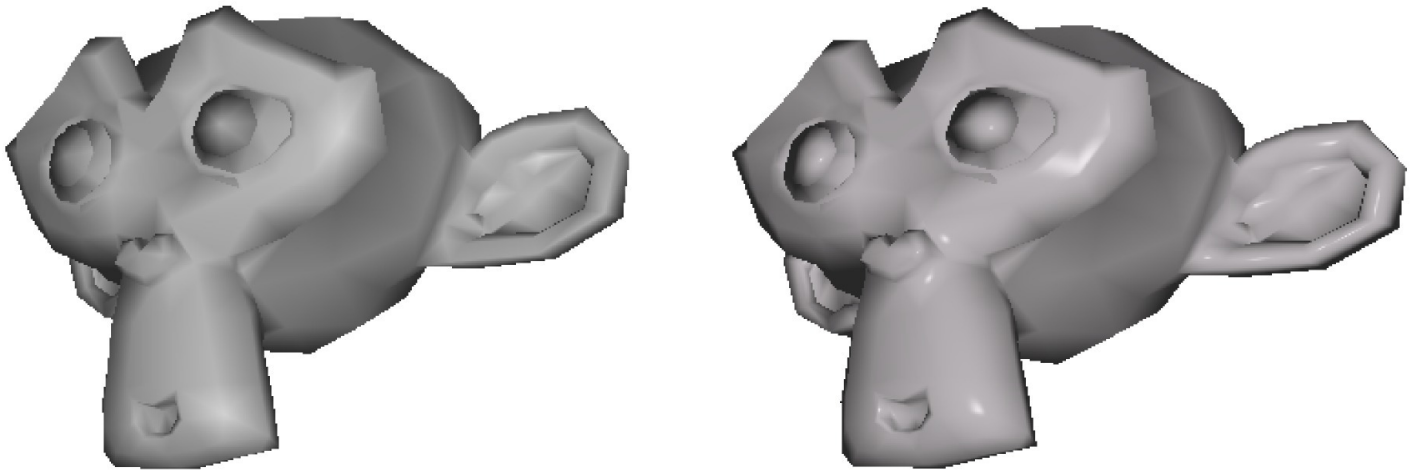


$$\mathbf{n}_k = \frac{\sum_{i \in \mathcal{V}(k)} \mathbf{n}_i}{\left\| \sum_{i \in \mathcal{V}(k)} \mathbf{n}_i \right\|}$$

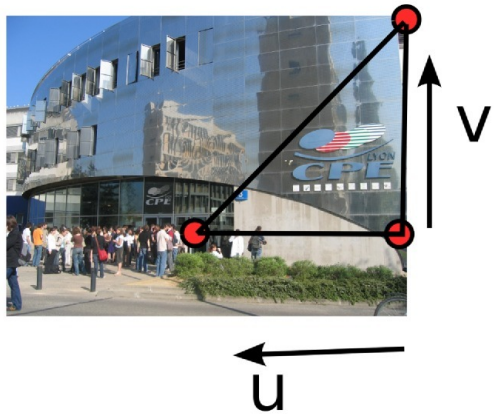
k : index of vertex
i : index of triangle
$\mathcal{V}(k)$: neighboring triangles of vertex k

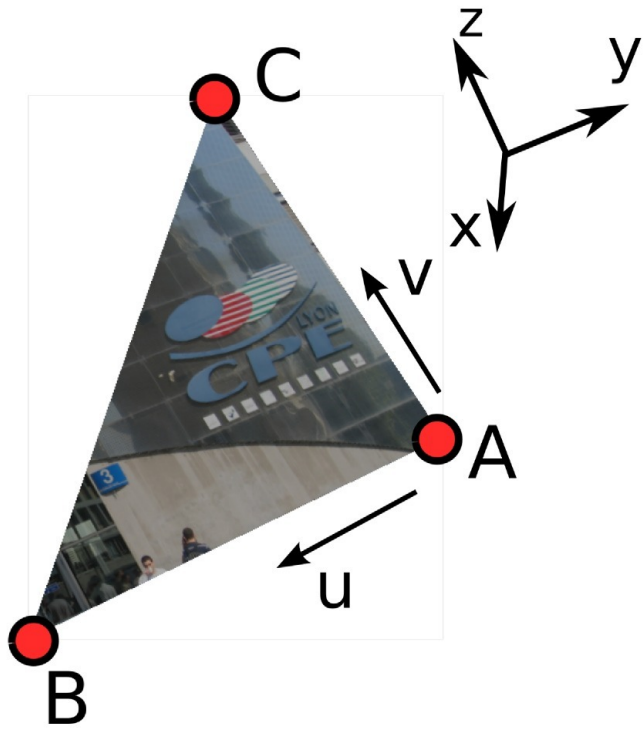


Shading Phong/Gouraud



Textures





File format

```
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