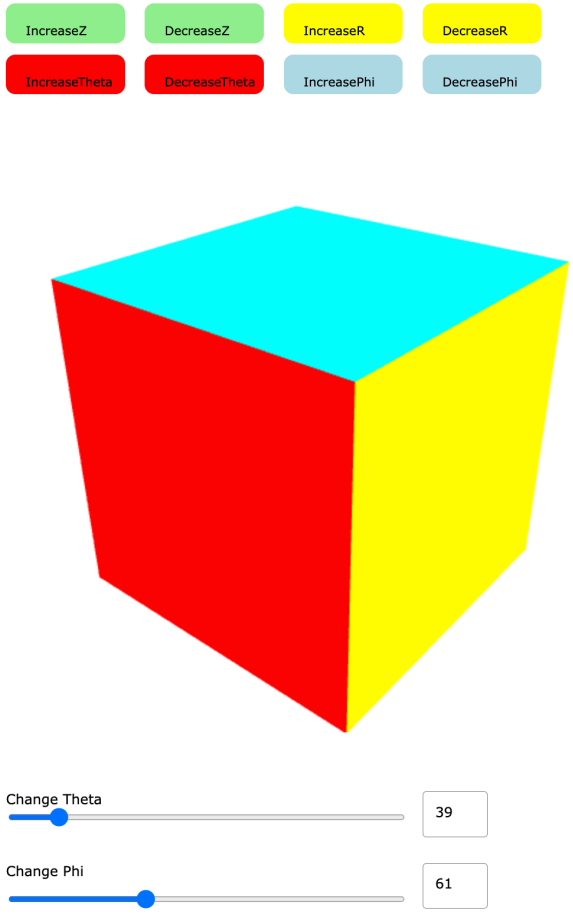
WebGL e Funzioni Utili



Libreria m4.js

Vediamo di usare una libreria di funzioni utili per definire le matrici VM e P che dovremo passare al vertex shader e in genere per gestire operazioni fra vettori e matrici (libreria m4.js).

$$VM = B^{-1} \quad \text{con} \quad B = \begin{pmatrix} \mathbf{xe} & \mathbf{ye} & \mathbf{ze} & D\sin\varphi\cos\theta \\ & & & D\sin\varphi\sin\theta \\ & & & D\cos\varphi \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$P = \begin{pmatrix} f/ar & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & \alpha & \beta \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Libreria m4.js

Per definire la matrice di vista VM solitamente si fa uso di una funzione chiamata `lookAt` con i seguenti parametri:

```
function lookAt( eye, at, up ){
…
}
```

`eye`: camera position
`at` : target position
`up` : View-Up vector

```
// Compute the camera's matrix using look at.
var cameraMatrix = m4.lookAt(cameraPos, cameraTar, up);
// Make a view matrix from the camera matrix.
var viewMatrix = m4.inverse(cameraMatrix);
```

Libreria m4.js

Per definire la matrice di proiezione P solitamente si fa uso di una funzione chiamata `perspective` con i seguenti parametri:

```
function perspective( fovy, aspect, near, far ){
  …
}
```

`fovy` : field of view in y axis in radians
`aspect` : aspect ratio of viewport (width / height)
`near` :near Z clipping plane
`far` :far Z clipping plane

```
// Compute the projection matrix
var projectionMatrix m4.perspective(fieldOfViewRadians,
                aspect, zNear, zFar);
```

m4.js e Typed Array

La libreria m4.js restituisce per default array tipizzati e per la precisione Float32Array.

Gli array tipizzati sono stati progettati dal comitato degli standard WebGL, per motivi di prestazioni.

Gli array Javascript sono generici e possono contenere oggetti, altri array ecc., e gli elementi non sono necessariamente sequenziali in memoria, come invece sono nel linguaggio C.

WebGL richiede che i buffer siano sequenziali in memoria, perché è così che l'API C si aspetta che siano.

m4.js e Typed Array

Nota: alcune funzioni come array.push non funzionano bene su array tipizzati. E' per questo che si usa la libreria m4.js che lavora e restituisce, su richiesta, anche array non tipizzati.

```
var t1 = m4.subtractVectors(vertices[b], vertices[a]);
var t2 = m4.subtractVectors(vertices[c], vertices[b]);
//restituisce un typed array Float32Array
normal = m4.cross(t1, t2);  //restituisce un array tipizzato
```

```
var t1 = m4.subtractVectors(vertices[b], vertices[a]);
var t2 = m4.subtractVectors(vertices[c], vertices[b]);
//restituisce un JavaScript array
var normal=[ ];
normal = m4.cross(t1, t2, normal); //restituisce un array
                                   //non tipizzato
```

m4.js functions

copy,
lookAt,
addVectors,
subtractVectors,
distance,
distanceSq,
normalize,
compose,
cross,
decompose,
mvec4,
dot,
identity,
transpose,
length,
orthographic,
frustum,

perspective,
translation,
translate,
xRotation,
yRotation,
zRotation,
xRotate,
yRotate,
zRotate,
axisRotation,
axisRotate,
scaling,
scale,
flatten,
multiply,
inverse,
transformVector,

transformPoint,
transformDirection,
transformNormal,
setDefaultType

Libreria webgl-utils.js

Useremo, per comodità anche altre semplici librerie, con il solo scopo di ridurre il numero di linee di codice da scrivere; per esempio useremo la libreria webgl-utils.js che mette a disposizione alcune funzioni utili per compilare, linkare e ottenere uno shader program.

```
function createProgramFromScripts(gl, shaderScriptIds, … ) {
…
}
```

```
 shaderScriptIds :Array of ids of the script tags for the
     shaders. The first is assumed to be the vertex shader,
     the second the fragment shader.
```

```
// setup GLSL program
var program = webglUtils.createProgramFromScripts(gl,
        ["3d-vertex-shader", "3d-fragment-shader"]);
```

Libreria webgl-utils.js

La libreria webgl-utils.js propone anche altre funzioni utili:

```
createAugmentedTypedArray,
createAttribsFromArrays,
createBuffersFromArrays,
createBufferInfoFromArrays,
createAttributeSetters,
createProgram,
createProgramFromScripts,
createProgramFromSources,
createProgramInfo,
createUniformSetters,
createVAOAndSetAttributes,
createVAOFromBufferInfo,
drawBufferInfo,
drawObjectList,
glEnumToString,
getExtensionWithKnownPrefixes,
resizeCanvasToDisplaySize,
setAttributes,
setBuffersAndAttributes,
setUniforms
```

Libreria ui_components.js

Si tratta di una semplice libreria di funzioni "fatta in casa" per gestire una GUI (Graphics User Interface). Implementa i widget più usuali come bottone, slider-bar, radio button, check box, input, label, menù, ecc. Si avvale del file ui_style.css per definire lo stile dei widget.

C'è una piccola guida all'utilizzo, si chiama ui.pdf (vedi pagina web del corso).

Per usarla basta includere nel file html:

```
<link href="resources/ui_style.css" rel="stylesheet" type="text/css">
<script type="text/javascript" src="resources/jquery-3.6.0.js"></script>
<script type="text/javascript" src="resources/ui_components.js"></script>
```

La libreria utilizza jQuery (jquery-3.6.0.js)

Libreria ui_components.js

La facilità d'uso consiste nel fatto che la GUI desiderata si realizza in due semplici passaggi:

1.richiamare le opportune funzioni della libreria per definire e posizionare sulla pagina web i widget;

2.per ogni widget definire la callback function che deve essere eseguita quando l'utente agisce sul quel widget.

La libreria contiene una function (detector()), che se richiamata riconosce se si sta lavorando su un dispositivo desktop o mobile e in quest'ultimo caso abilità gli eventi relativi (touch).

Nella cartella c'è il file ui_components.js, ui_style.css e alcuni esempi che le usano; sono tutti quelli il cui nome finisce con: _UI.html e .js

Nota: su chrome è possibile simulare un dispositivo mobile.

Libreria dat.gui.js

Un paio di programmatori di Google hanno creato una libreria chiamata dat.GUI, che permette di creare molto facilmente una semplice interfaccia utente che può modificare le variabili utilizzate nel codice. Per la documentazione si veda:

https://github.com/dataarts/dat.gui

E' la libreria utilizzata da three.js per mostrare e debuggare gli esempi.

Nella cartella c'è il file dat.gui.js e alcuni esempi che la usano; sono tutti quelli il cui nome finisce con: _GUI.html e .js

Esempi 1/3

Vediamo tre esempi (codici presenti nella cartella HTML5_webgl_2) che visualizzano un cubo colorato e utilizzano le librerie webgl-utils.js, m4.js, ui_components.js e dat.gui.js.

In tutti questi codici i programmi shader vengono scritti come script definendo type = "not-javascript".

Vediamo il primo:

cube_interact_shaderscript.html e .js

Questo codice è la modifica del già visto cube_interact.html e .js (se ricordate, all'oggetto cubo vengono applicate delle rotazioni intorno agli assi definite muovendo il mouse); qui per definire le rotazioni, le matrici di vista e di proiezione si usano le funzioni della libreria m4.js.
Invece per tutte le operzioni necessarie per creare il programma shader si usa la libreria webgl-utils.js.

Esempi 2/3

Vediamo il secondo:

cube_perspective_UI.html e .js

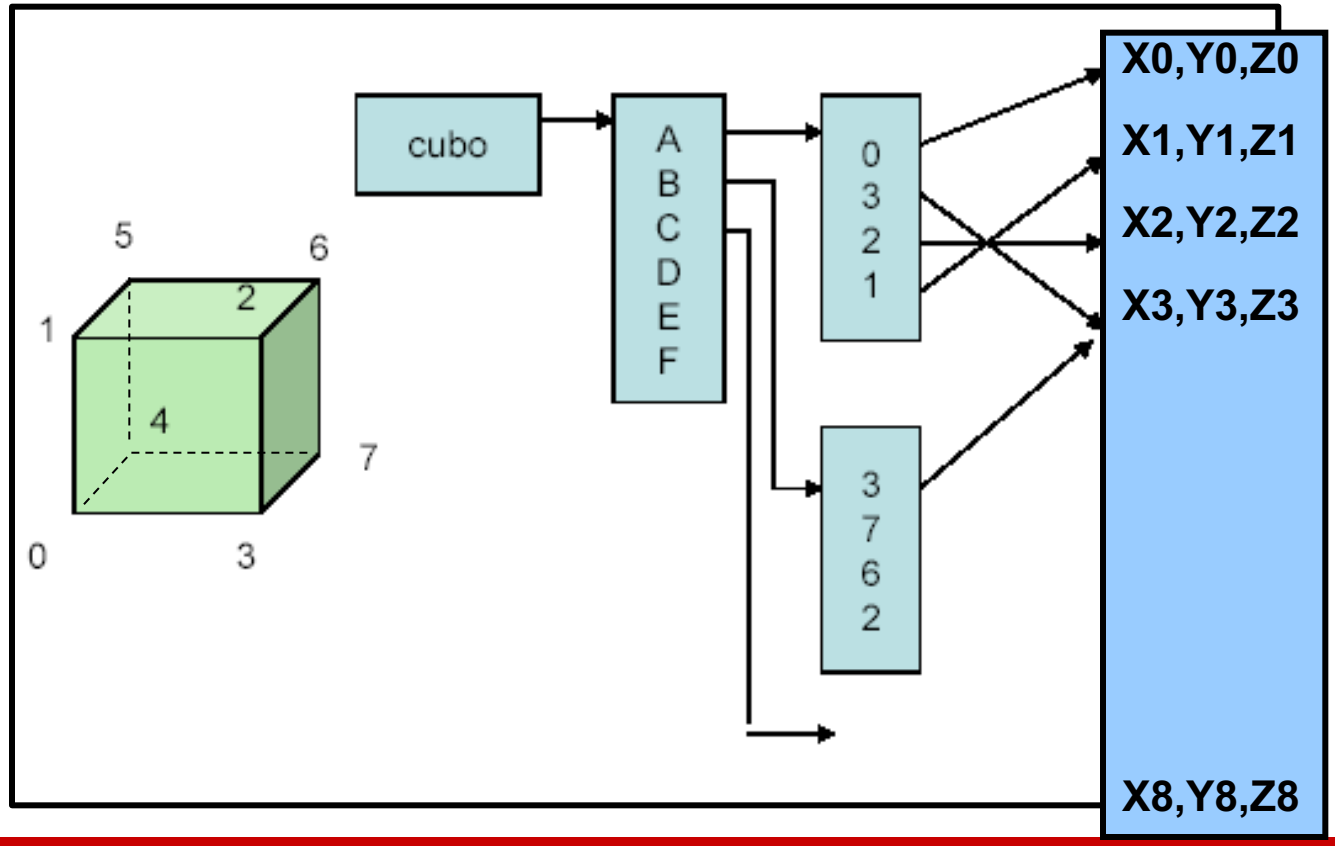
In questo codice l'oggetto cubo rimane fermo, mentre con la camera gli si gira intorno; i parametri della camera vengono definiti richiamando la function m4.lookAt della libreria m4.js

Inoltre, viene definita una GUI utilizzando la libreria ui_components.js e ui_style.css

Definiamo un Cubo 1/5

Ancora, osserviamo la modalità con cui si definisce la geometria cubo e i colori per ogni faccia.

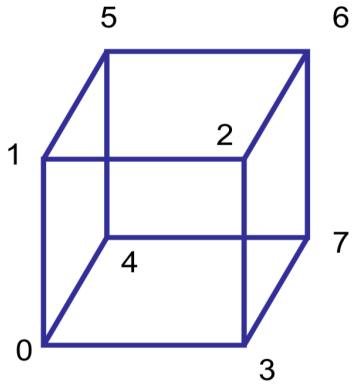
Oggetto mesh cubo (6 facce e 8 vertici) con un colore per ogni faccia:



Definiamo un Cubo 2/5

Definiamo gli 8 vertici del cubo in coordinate omogenee:

```
var vertices = [ [ 1.0, -1.0, -1.0, 1.0,], [ 1.0, -1.0, 1.0, 1.0,],
                 [ 1.0,  1.0,  1.0, 1.0,], [ 1.0,  1.0, -1.0, 1.0,],
                 [-1.0, -1.0, -1.0, 1.0,], [-1.0, -1.0, 1.0, 1.0,],
                 [-1.0,  1.0,   1.0, 1.0,],[-1.0, 1.0, -1.0, 1.0,] ];
```



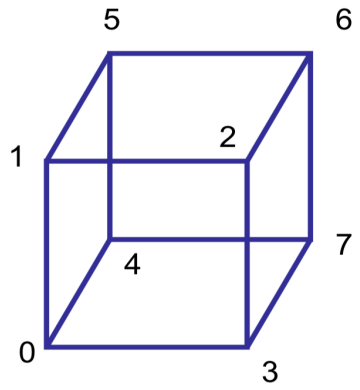
e definiamo i colori per vertice:

```
var vertexColors = [  [0.0, 0.0, 0.0, 1.0,],  // black
                      [1.0, 0.0, 0.0, 1.0,],  // red
                      [1.0, 1.0, 0.0, 1.0,],  // yellow
                      [0.0, 1.0, 0.0, 1.0,],  // green
                      [0.0, 0.0, 1.0, 1.0,],  // blue
                      [1.0, 0.0, 1.0, 1.0,],  // magenta
                      [0.0, 1.0, 1.0, 1.0,],  // cyan
                      [1.0, 1.0, 1.0, 1.0,] ]; // white
```

Definiamo un Cubo 3/5

Definiamo le facce:

```
function colorCube()
{
    quad(1, 0, 3, 2);
    quad(2, 3, 7, 6);
    quad(3, 0, 4, 7);
    quad(6, 5, 1, 2);
    quad(4, 5, 6, 7);
    quad(5, 4, 0, 1);
}
```



Definiamo un Cubo 4/5

Definiamo ogni faccia con un colore (e precisamente con il colore del primo vertice che definisce la faccia):

```
function quad(a, b, c, d) {
        pointsArray.push(vertices[a]);
        colorsArray.push(vertexColors[a]);
        pointsArray.push(vertices[b]);
        colorsArray.push(vertexColors[a]);
        pointsArray.push(vertices[c]);
        colorsArray.push(vertexColors[a]);
        pointsArray.push(vertices[a]);
        colorsArray.push(vertexColors[a]);
        pointsArray.push(vertices[c]);
        colorsArray.push(vertexColors[a]);
        pointsArray.push(vertices[d]);
        colorsArray.push(vertexColors[a]);
}
```

Definiamo un Cubo 5/5

In questo modo stiamo definendo due array rispettivamente di vertici e colori da copiare in due VBO:

```
colorCube();
pointsArray=m4.flatten(pointsArray);
colorsArray=m4.flatten(colorsArray);
```

dove m4.flatten è necessario per ottenere un array tipizzato unidimensionale, come richiede WebGL:

render function

```
var render = function(){
// Tell WebGL how to convert from clip space to pixels
  gl.clear( gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

  eye = [radius*Math.sin(phi)*Math.cos(theta),
            radius*Math.sin(phi)*Math.sin(theta), radius*Math.cos(phi)];
// Compute the camera's matrix
  var cameraMatrix = m4.lookAt(eye, at, up) ;

// Make a view matrix from the camera matrix.
   var vmMatrix = m4.inverse(cameraMatrix) ;

// Compute the projection matrix
  var pMatrix = m4.perspective(degToRad(fovy), aspect, near, far) ;

  gl.uniformMatrix4fv( modelView, false, vmMatrix );
  gl.uniformMatrix4fv( projection, false, pMatrix );
  gl.drawArrays( gl.TRIANGLES, 0, NumVertices );
}
```

Esempi 3/3

Vediamo il terzo:

cube_perspective_GUI.html e .js

La differenza fra questo codice ed il precedente sta nell'utilizzo della GUI, qui viene usata la libreria dat.gui.js.

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Giulio Casciola

Dip. di Matematica

giulio.casciola at unibo.it