Esercitazione 5 - Texture mapping

Punto 1

Il primo punto dell'esercitazione richiede di implementare il texture mapping 2D del toro con immagini lette da file in formato .bmp. Per fare ciò si è modificato il metodo putVert() come segue:

```
int numTexVer = 2; // ripetizioni verticali della texture sul toro
int numTexHor = 10; // ripetizioni orizzontali della texture sul toro

void putVert(int i, int j) {
   float phi = PI2 * j / NumPerWrap;
   float theta = PI2 * i / NumWraps;
   float sinphi = sin(phi);
   float cosphi = cos(phi);
   float sintheta = sin(theta);
   float costheta = cos(theta);
   float r = MajorRadius + MinorRadius * cosphi;

   glNormal3f(sintheta * cosphi, sinphi, costheta * cosphi);

   float s = (float) i / (float) NumWraps * (float) numTexHor;
   float t = (float) j / (float) NumPerWrap * (float) numTexVer;

   glTexCoord2f(s, t);

   glVertex3f(sintheta * r, MinorRadius * sinphi, costheta * r);
}
```

Di particolare interesse sono i parametri *s* e *t* che definiscono le coordinate da considerare sulla texture. I parametri s e t sono compresi, rispettivamente, sugli intervalli [0, numTexHor] e [0, numTexVer], fatto abbastanza inusuale in quanto di solito si considera l'intervallo [0, 1]. L'utilizzo di due ulteriori funzioni di OpenGL permette il corretto funzionamento del mapping:

```
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
```

In questo modo, quando i parametri s e t sono superiori a 1, ne viene considerata solamente la parte decimale. Ad esempio, se s = 1,32 OpenGL considera s = 0,32.

Di seguito è mostrato un esempio con NumWraps = 4, NumPerWrap = 4, numTexHor = 2, numTexVer = 2.

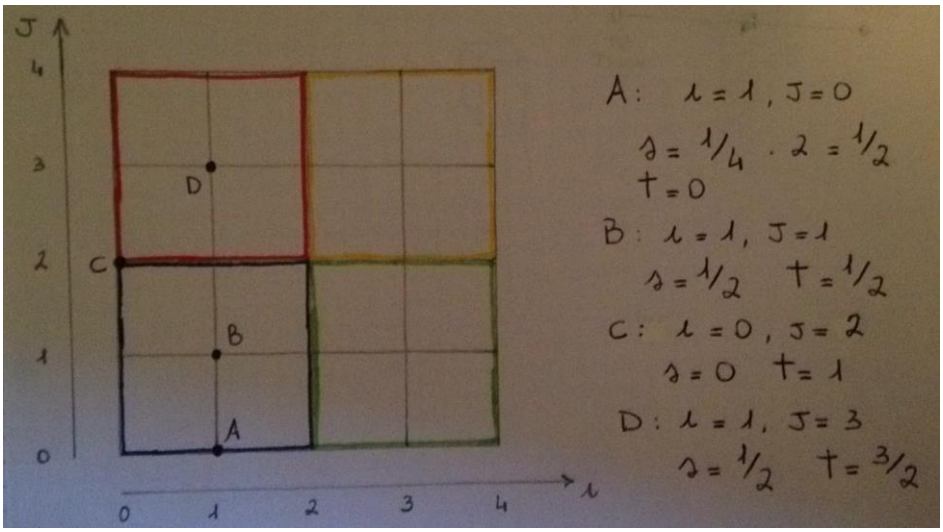

Figura 1 - Esempio di mapping

Il risultato finale applicando diverse texture:

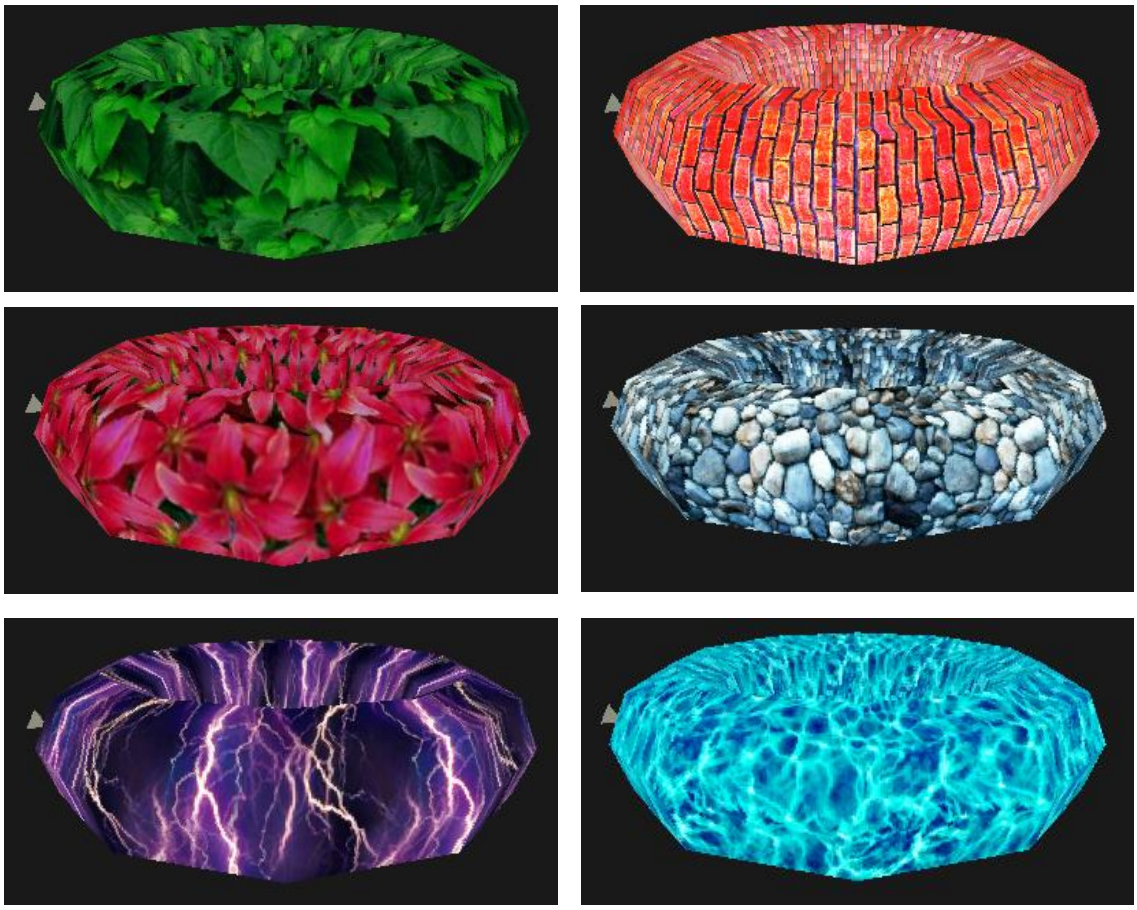

Figura 2 - Applicazione di diverse texture

Punto 2
Il secondo punto dell'esercitazione richiede di implementare l'environment mapping sferico e cubico.

L'environment mapping sferico si implementa molto facilmente in quanto è necessario disporre solamente di un'immagine con effetto "fisheye" e di abilitare alcune funzioni di OpenGL.

```
glEnable(GL_TEXTURE_2D);
glTexGeni(GL_S,GL_TEXTURE_GEN_MODE ,GL_SPHERE_MAP);
glTexGeni(GL_T,GL_TEXTURE_GEN_MODE ,GL_SPHERE_MAP);
glEnable(GL_TEXTURE_GEN_S);
glEnable(GL_TEXTURE_GEN_T);
glBindTexture(GL_TEXTURE_2D, sphereTexture);
```

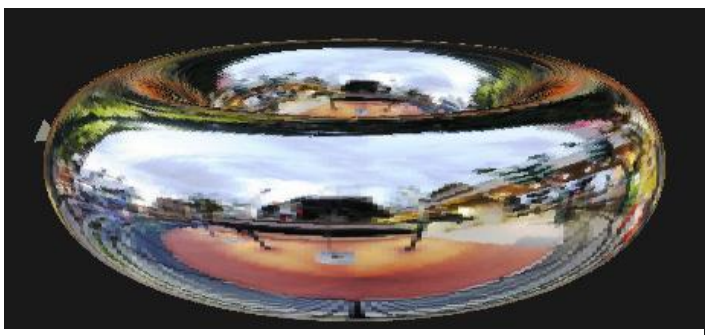

Figura 3 - Environment mapping sferico

L'environment mapping cubico necessita l'abilitazione di alcune funzioni di OpenGL similmente al caso di mapping sferico.

```
glEnable(GL_TEXTURE_CUBE_MAP);
glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_REFLECTION_MAP);
glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_REFLECTION_MAP);
glTexGeni(GL_R, GL_TEXTURE_GEN_MODE, GL_REFLECTION_MAP);
glEnable(GL_TEXTURE_GEN_S);
glEnable(GL_TEXTURE_GEN_T);
glEnable(GL_TEXTURE_GEN_R);
glBindTexture(GL_TEXTURE_CUBE_MAP, cubeTexture);
```

Inoltre, è necessario effettuare il mapping delle sei immagini del cubo secondo lo schema mostrato qui a sotto.

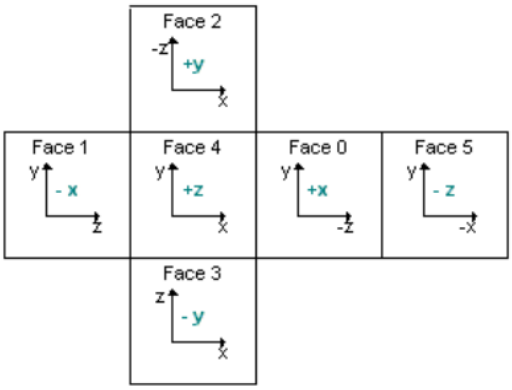

Figura 4 - Mapping delle sei immagini del cubo

A livello implementativo:

```
glGenTextures(1, &cubeTexture);
glBindTexture(GL_TEXTURE_CUBE_MAP, cubeTexture);
glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_REFLECTION_MAP);
glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_REFLECTION_MAP);
glTexGeni(GL_R, GL_TEXTURE_GEN_MODE, GL_REFLECTION_MAP);

RgbImage img1 = getTextureFromFile(cube[0]);
RgbImage img2 = getTextureFromFile(cube[1]);
RgbImage img3 = getTextureFromFile(cube[2]);
RgbImage img4 = getTextureFromFile(cube[3]);
RgbImage img5 = getTextureFromFile(cube[4]);
RgbImage img6 = getTextureFromFile(cube[5]);

glTexImage2D(GL_TEXTURE_CUBE_MAP_NEGATIVE_X, 0, GL_RGB, 256, 256, 0, GL_RGB, GL_UNSIGNED_BYTE,
ImageData(&img1));
glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_X, 0, GL_RGB, 256, 256, 0, GL_RGB, GL_UNSIGNED_BYTE,
ImageData(&img2));
glTexImage2D(GL_TEXTURE_CUBE_MAP_NEGATIVE_Y, 0, GL_RGB, 256, 256, 0, GL_RGB, GL_UNSIGNED_BYTE,
ImageData(&img3));
glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_Y, 0, GL_RGB, 256, 256, 0, GL_RGB, GL_UNSIGNED_BYTE,
ImageData(&img4));
glTexImage2D(GL_TEXTURE_CUBE_MAP_NEGATIVE_Z, 0, GL_RGB, 256, 256, 0, GL_RGB, GL_UNSIGNED_BYTE,
ImageData(&img5));
glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_Z, 0, GL_RGB, 256, 256, 0, GL_RGB, GL_UNSIGNED_BYTE,
ImageData(&img6));

glTexParameteri(GL_TEXTURE_CUBE_MAP,GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_CUBE_MAP,GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_CUBE_MAP,GL_TEXTURE_WRAP_S,GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_CUBE_MAP,GL_TEXTURE_WRAP_T,GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_CUBE_MAP,GL_TEXTURE_WRAP_R,GL_CLAMP_TO_EDGE);
```

Si noti come si sia utilizzata una funzione di libreria per leggere correttamente le immagini in formato .bmp.

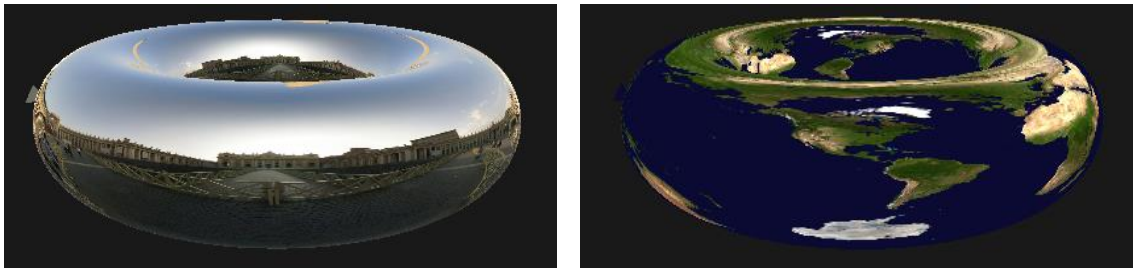

Figura 5 - Environment mapping cubico

Punto 3

Il terzo punto dell'esercitazione richiede di implementare il *procedural mapping* con algoritmo a piacere. A tal fine si è modificato il metodo esistente initCheckerTextures() (che implementava un texture mapping a scacchiera) per ottenere una texture con i colori dell'arcobaleno.

```
GLubyte image[7][1][3];
int colors[7][3] = { {255, 0, 0}, {255, 196, 0}, {255, 247, 0}, {0, 255, 0},
                     {0, 230, 255}, {0, 0, 255}, {213, 0, 255} };

for (int i = 0; i < 7; i++) {
   for (int j = 0; j < 1; j++) {
      image[i][j][0] = (GLubyte) colors[i][0];
      image[i][j][1] = (GLubyte) colors[i][1];
      image[i][j][2] = (GLubyte) colors[i][2];
   }
}
```

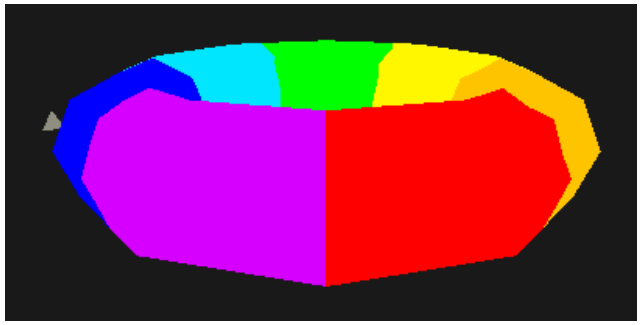

Figura 6 - Procedural mapping