Grafica Computazionale

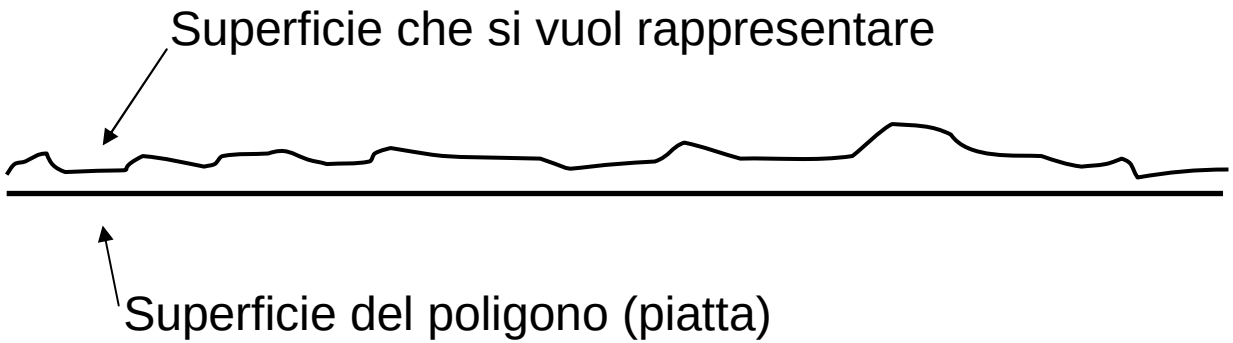
Bump mapping

Fabio Ganovelli
fabio.ganovelli@gmail.com
a.a. 2006-2007

Bump Mapping

- Tecnica per far apparire una superficie piatta come se avesse del dettaglio geometrico

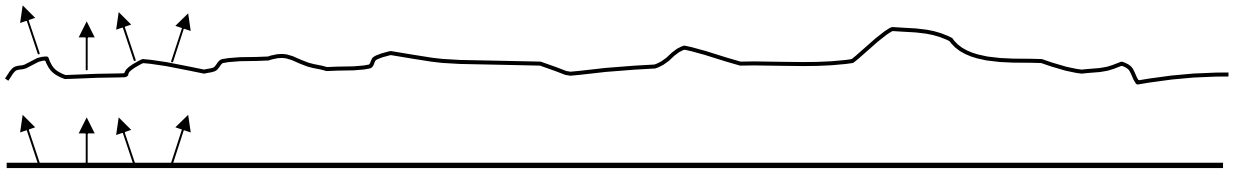
Bump mapping



Osservazione chiave: la percezione della profondità dipende largamente dall'illuminazione

Questa immagine mostra il rendering di un singolo quad

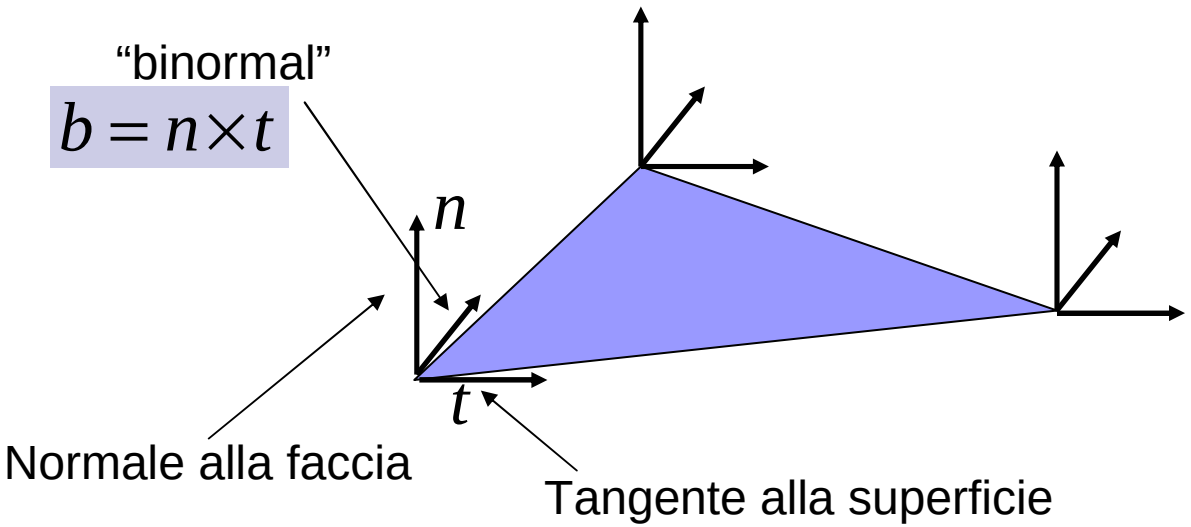
Bump mapping



- Come funziona:
  - Una texture (chiamata normal map) codifica la normale nel punto (invece del colore)
  - Il lighting viene fatto per frammento usando tale normale

Bump Mapping

- La normal map codifica le normali nello spazio texture
- Le direzioni di vista e della luce sono espresse in object space
- La texture è “spalmata” sulla superficie, cioè gli è **tangente** in ogni punto
- Per questo lo spazio in cui sono espresse le normali è chiamato tangent space
- Ci serve la trasformazione che porta le direzioni (vista, luce) da object space a tangent space



Bump Mapping

- Il bump mapping è una funzionalità presente in opengl, ma noi lo facciamo con gli shaders
- Vertex shader:
  - Prende la normale e la tangente dal contesto (quelle passate dall'utente), completa la costruzione del frame con la binormale e trasforma la direzione di vista e della luce in tangent space
- Fragment shader:
  - Legge direzioni di vista e luce interpolate e calcola il lighting

Vertex program: demo/lab.09.normal_mapping/p0.vert

```
/* bump map vertex shader */
varying vec3 lightv; /* direzione della luce in texture space coordinates */
varying vec3 viewv;  /* direzione di vista in texture space coordinates */
attribute vec3 tangento; /* vettore tangente alla superficie in object coordinates */

void main(){
	/* posizione del vertice in clip coordinates */
	gl_Position = gl_ModelViewProjectionMatrix*gl_Vertex;

	vec3 viewe = vec3(gl_ModelViewMatrix*gl_Vertex);

	gl_TexCoord[0] = gl_MultiTexCoord0;

	vec3 lighte = normalize(vec3(gl_LightSource[0].position ) - viewe);

	vec3 normal = normalize(gl_NormalMatrix * gl_Normal);
	vec3 tangent = normalize(gl_NormalMatrix * tangento);
	vec3 binormal = cross(normal,tangent);

	lightv.x = dot(tangent,lighte.xyz);
	lightv.y = dot(binormal,lighte.xyz);
	lightv.y = dot(normal,lighte.xyz);

	viewv.x = dot(tangent,viewe.xyz);
	viewv.y = dot(binormal,viewe.xyz);
	viewv.z = dot(normal,viewe.xyz);

	lightv = normalize(lightv);
	viewv = normalize(viewv);
}
```

fragment program: demo/lab.09.normal_mapping/p0.frag

```
varying vec3 lightv; /* direzione della luce in texture space coordinates */
varying vec3 viewv;  /* direzione di vista in texture space coordinates */
varying vec4 col;

uniform sampler2D texMap;
uniform sampler2D normalMap;

void main(){

vec4 Light = vec4(normalize(lightv),0.0);
vec4 normal  =  texture2D(normalMap,gl_TexCoord[0].st);
normal = 2.0*(normal - vec4(0.5,0.5,0.5,0.5));
float val =  max(dot(Light,normal),0.0);
gl_FragColor = vec4(val,val,val,1.0);
}
```

Q1: il valore viewv (la direzione di vista) non è utilizzato.
Per cosa potremmo utilizzarlo?

Q2: a cosa serve la linea in grassetto?