Grafica al calcolatore
Laboratorio – 4

Andrea Giachetti andrea.giachetti@univr.it
Fabio Marco Caputo fabiomarco.caputo@univr.it
Department of Computer Science, University of Verona, Italy

Reminder:

- Ricordare il path per le librerie GLFW
  - export LD_LIBRARY_PATH=lib/lin
- Per usare l'emulazione mesa
  - export LIBGL_ALWAYS_SOFTWARE=1
- Per far funzionare i codici su alcune macchine con scheda video intel su ubuntu
  - export MESA_GLSL_VERSION_OVERRIDE=130
  - Potrebbe essere provato in alternativa al comando precedente

Depth buffer e backface culling

- Cosa succede se si commentano le corrispondenti righe? Verificare!

- //Qui si attiva lo z-buffer!
- glEnable(GL_DEPTH_TEST); check(__LINE__);
- //qui si attiva il back face culling
- glEnable(GL_CULL_FACE); check(__LINE__);

Uso texture multiple

- glGenTextures(2, textures);
- ...
- glActiveTexture(GL_TEXTURE1);
- glBindTexture(GL_TEXTURE_2D, textures[1]);
-
-
- glUniform1i(glGetUniformLocation(shaderProgram, "textureSampler"), 0);
- glUniform1i(glGetUniformLocation(shaderProgram, "textureSampler2"), 1);

Interpolazione tra 2 valori

- //GLSL provides the mix function. This function should be used where possible:
- resultRGB = mix(colorRGB_0, colorRGB_1, alpha);

Tempo e animazione

- Per animare e muovere interattivamente gli oggetti e la telecamera occorre affidarsi al tempo macchina anche per evitare problemi dovuti alle differenti prestazioni dell'HW
  - Provate a confrontare cosa succederebbe se usaste rotazione proporzionale al numero di frame calcolati...
- #include <chrono>
- using timer = std::chrono::high_resolution_clock;
- Nel draw calcoliamo il tempo intercorso dal disegno precedente
- t = (timer::now()-start_time).count() * (float(timer::period::num)/float(timer::period::den));
- dt = (timer::now()-last_time).count() * (float(timer::period::num)/float(timer::period::den));
- glm::mat4 model = glm::rotate(glm::mat4(1.f), t, glm::vec3(0.0f, 1.0f, 0.0f)); //anima il cubo centrale, dt si usa per la navigazione

Esempio: e05.cpp

- Aggiugiamo:
  - Scambio texture
  - Animazione
    - Rotazione del cubo
  - Navigazione
    - Con i tasti spostiamo la camera virtuale
    - Descrizione...

Esercizio

- Aggiungere nella scena un terzo cubo, alle coordinate (3,3,3)
- Al premere del tasto O, il cubo inizi a spostarsi per andare verso l'origine (0,0,0) ove si fermi
- Associare a questo cubo uno shader diverso che colori mappando l'immagine cube3.png modulandola con una luminosità dipendente dal tempo

Illuminare il mondo

- Per ora abbiamo imparato a fare programmi per“vedere” oggetti 3D con una telecamera cirtuale (prospettica o ortografica)
- Ma abbiamo colorato i punti proiettati o con colori fissi o col texture mapping
- Ma nel corso abbiamo imparato a “illuminare” le scene in modo fisicamente plausibile
  - Almeno con le grossissime approssimazioni del modello di Phong
- Nel primo openGl implementazione fissa
  - Implementazione calcolata per vertice e interpolata dopo
- Ora si implementa l'illuminazione negli shader
  - Possiamo fare come vogliamo, ma dobbiamo implementarlo esplicitamente

Il modello di Phong

- Componente ambientale, diffusiva e speculare

$$I^{out} = I_a k_a + I\left(k_d(\mathbf{n}\cdot\mathbf{l}) + k_s(\mathbf{n}\cdot\mathbf{h})^n\right)$$

- Se si stanno considerando i colori, allora sia le intensità della luce che i coefficienti del materiale vanno definiti per ogni componente (r,g,b)

$$I^{r,out} = I_a^r k_a^r + I^r\left(k_d^r(\mathbf{n}\cdot\mathbf{l}) + k_s^r(\mathbf{n}\cdot\mathbf{h})^n\right)$$

$$I^{g,out} = I_a^g k_a^g + I^g\left(k_d^g(\mathbf{n}\cdot\mathbf{l}) + k_s^g(\mathbf{n}\cdot\mathbf{h})^n\right)$$

$$I^{b,out} = I_a^b k_a^b + I^b\left(k_d^b(\mathbf{n}\cdot\mathbf{l}) + k_s^b(\mathbf{n}\cdot\mathbf{h})^n\right)$$

- I coefficienti di diffusione e ambientali di solito sono uguali
  - La superficie appare del colore specificato dalla terna di coefficienti quando illuminata da luce bianca.
  - Le riflessioni speculari (highlights) invece sono di solito del colore della luce ($k^r_s = k^g_s = k^b_s = 1$)

e06

- Facciamo un passo ulteriore: non possiamo sempre scrivere i modelli a mano, ma possiamo/dobbiamo usare modelli generati a parte
  - Questi possono contenere sia le geometrie poligonali sia informazioni associate
  - Ad esempio le texture
  - Ad esempio i coefficienti dei materiali da associare
- Metteremo modelli e file associati nella cartella “asset”
- Useremo librerie esterne per gestire il caricamento

Caricare i file

- Servono funzioni che creino strutture dati come quelle che abbiamo creato a mano con vertici e indici (ma anche normali e coordinate texture) dai file
- Usiamo la libreria tinyobj
  - http://syoyo.github.io/tinyobjloader/
- Carichiamo file obj e un file associato coi parametri (.mtl)
- Le specifiche sono quelle del formato Wavefront .obj
- Potete trovarle all'indirizzo
  - http://en.wikipedia.org/wiki/Wavefront_.obj_file
- Nell'esercizio, carichiamo due modelli di sfera, con differenti risoluzioni. Il draw li disegna entrambi in posizioni diverse
  - Non usiamo per il momento la texture
  - Usiamo però lo shading con la luce (direzionale)

e06

- Nel codice si caricano due modelli (sfere di diverse risoluzione che hanno associati parametri e coordinate texture)

```
newmtl materiale  // nome materiale
Ka 0 0 0  // Ambient: notare: è spenta
Kd 0.784314 0.784314 0.784314 // diffuse
Ks 0 0 0   // specular
Ni 1      // sarebbe densità
Ns 100    // esponente speculare (shininess)
Tf 1 1 1  //
D 1         //
map_Kd sphere.png  // vorrebbe dire che
// si dovrebbe moltiplicare la kd per la
// texture
```

Note

- Quindi passiamo allo shader i parametri caricati
- In più definiamo una luce direzionale (all'infinito)

```
glUniform3fv(glGetUniformLocation(shaderProgram,
"light_direction"), 1, &light_direction[0]);
glUniform3fv(glGetUniformLocation(shaderProgram,
"light_intensity"), 1, &light_intensity[0]);
glUniform3fv(glGetUniformLocation(shaderProgram,
"view_position"), 1, &camera_position[0]);
glUniform1f(glGetUniformLocation(shaderProgram,
"shininess"), material[0].shininess);
glUniform3fv(glGetUniformLocation(shaderProgram,
"material_ambient"), 1, material[0].ambient);
glUniform3fv(glGetUniformLocation(shaderProgram,
"material_diffuse"), 1, material[0].diffuse);
glUniform3fv(glGetUniformLocation(shaderProgram,
"material_specular"), 1, material[0].specular);
```

Shading per vertici

- Lo shader fornito, carica questi valori e calcola l'illuminazione con modello di Phong sui vertici.
- Poi in rasterizzazione il colore viene interpolato sulle facce
- Vertex shader:

```
"flat out vec3 Color;"
…
void main() {"
    "   vec3 Normal = mat3(model) * normal;"
    "   gl_Position = projection*view*model * vec4(position, 1.0);"
    "   vec4 vertPos = model * vec4(position, 1.0);"
    "   vec3 Position = vec3(vertPos)/vertPos.w;"
    "   vec3 view_direction = normalize(view_position - Position);"
    "   vec3 R = reflect(-light_direction, Normal);"
    "   Color = material_ambient;"
    "   Color += material_diffuse * max(dot(light_direction,
Normal), 0.0);"
    "   Color += material_specular * pow(max(dot(R, view_direction),
0.0), shininess);"
    "}";
```

Shading

- Fragment shader

```
"flat in vec3 Color;"
  "out vec4 outColor;"
  "void main() {"
  "  outColor = vec4(Color, 1.0);"
  "}";
```

- Flat shading.
- Se togliamo le dichiarazioni “flat” cosa succede? (e06b.cpp)
  - Che si applica il Gouraud shading (interpolando sul triangolo il colore)
  - Cosa cambia?

Phong shading

- La scarsa qualità dell'highlight speculare è uno dei limiti del Gouraud shading.
- Ma noi possiamo anche implementare il phong shading, interpolando le normali
- Quindi:
  - Non calcolare l'illuminazione sul vertex shader
  - Passare le variabili (che verranno interpolate sui frammenti) al fragment shader
  - Calcolare l'illuminazione sul fragment shader (stessa equazione)
  - Provate a farlo (dopo gli altri esercizi magari)

Esercizi

- Eseguire e06
- Modificando i parametri nel file mtl per entrambi i modelli
  - Colorare l'oggetto di colore tendente al rosso variando le componenti diffusive
  - Attivare una luce ambientale di colore tendente all'azzurro. Cosa cambia?
  - Attivare la componente speculare. Cosa succede al variare di Ns? Cosa accade quando diventa alto?
- Eseguire e06b
- Utilizzare la texture caricata nel codice e modularla con la componente diffusiva e ambientale (non con la speculare)
  - Le coordinate texture sono fornite nei modelli
  - Come cambierebbe modulando anche la speculare?

Altri esercizi

- Provare ad aggiungere una seconda sorgente luminosa di colore giallo con differente direzione.
  - Rendere possibile accenderla premendo il tasto 'g'
  - Aggiungere all'accensione anche una componente ambientale di colore tendente al giallo
- Fare in modo che al premere del tasto “O” la sfera a destra orbiti intorno a quella centrale (ruotando quindi sul piano xz attorno all’asse y)
- Spostare il calcolo dello shading al fragment shader