Grafica al calcolatore
Altri effetti
Assegnamento finale

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

Ulteriori effetti

- Interazione con il mouse
- Collisione tra oggetti
- Skybox
- Creare una piccola texture per il tastino e applicarla
- Visualizzare un modello ulteriore nella scena, ad es. l'astronave che avevate creato alla prima lezione in laboratorio. Usare uno shader apposito
- Tutto implementato nel codice e08.cpp

Interazione col mouse

- Del tutto simile all'uso dei tasti, la gestiamo con GLFW

```
void mouse_button_callback(GLFWwindow*
window, int button, int action, int mods)
{
if (button == GLFW_MOUSE_BUTTON_LEFT &&
action == GLFW_PRESS)
{
if ((0 <= x_mousepos && x_mousepos <=
button_size) && (0 <= y_mousepos &&
y_mousepos <= button_size))
{
gui_button_press();
}}}
```

Interazione col mouse

- Nel codice usiamo l'interazione col mouse per creare un piccolo pulsante per interagire che mappiamo in un angolo
- Interagire col mondo 3D è più complesso: dovremmo rimappare in qualche modo il punto dello screen space dove clicchiamo con la scena 3D

Collisioni

- Possiamo far interagire i nostri oggetti della scena
  - Il modo più semplice è gestire le collisioni: quando gli oggetti si sovrappongono
- Ma dobbiamo calcolare le intersezioni
  - Soluzione più semplice: usare bounding box, cosa più semplice: sfere Supponiamo di voler far accadere qualcosa quando la nostra telecamera (nave spaziale) si scontra con sole o pianeti
- I pianeti sono sfere con un certo raggio
- La nostra "nave" la pensiamo come racchiusa in una sfera di raggio 0.1
- Il test di collisione diventa abbastanza semplice
  - void check_collision()

Test collisione

```
void check_collision()
{
if(glm::distance2(camera_position,
planetPosition[0]) <=
std::pow((planetSize[0] + 0.1f),2.f))
std::cout << "collision camera - sun" << std::endl;
if(glm::distance2(camera_position,
planetPosition[1]) <=
std::pow((planetSize[1] + 0.1f),2.f))
std::cout << "collision camera - planet" <<
std::endl;
if(glm::distance2(camera_position,
planetPosition[2]) <=
std::pow((planetSize[2] + 0.1f),2.f))
std::cout << "collision camera - moon" <<
std::endl;
}
```

Skybox

- Abbiamo sempre visto lo sfondo con colore uniforme
- Possiamo però usare il texture mapping per simulare un ambiente intorno
- Tecnica comune nei videogiochi “skybox”
- In fondo siete già capaci: create degli oggetti opportuni e mappate texture
  - Creaiamo una “scatola” in cui includere la regione di azione
  - Applichiamo una texture

Skybox

- Qui usa un particolare tipo di texture CUBE_MAP.
  - Quando disegniamo facciamo:

```
glDepthMask(GL_FALSE);
glUseProgram(skyProgram);
glActiveTexture(GL_TEXTURE3);
glBindTexture(GL_TEXTURE_CUBE_MAP, textureCube);
glUniform1i(glGetUniformLocation(skyProgram,
"textureSampler"), 3);
glUniformMatrix3fv(glGetUniformLocation(skyProgram,
"view_rot"), 1, GL_FALSE,
&(glm::mat3(glm::inverse(view)))[0][0]);
glBindVertexArray(vao[2]);
glDrawElements(GL_TRIANGLES, sizeof(elements),
GL_UNSIGNED_INT, 0);
glDepthMask(GL_TRUE);
```

Skybox

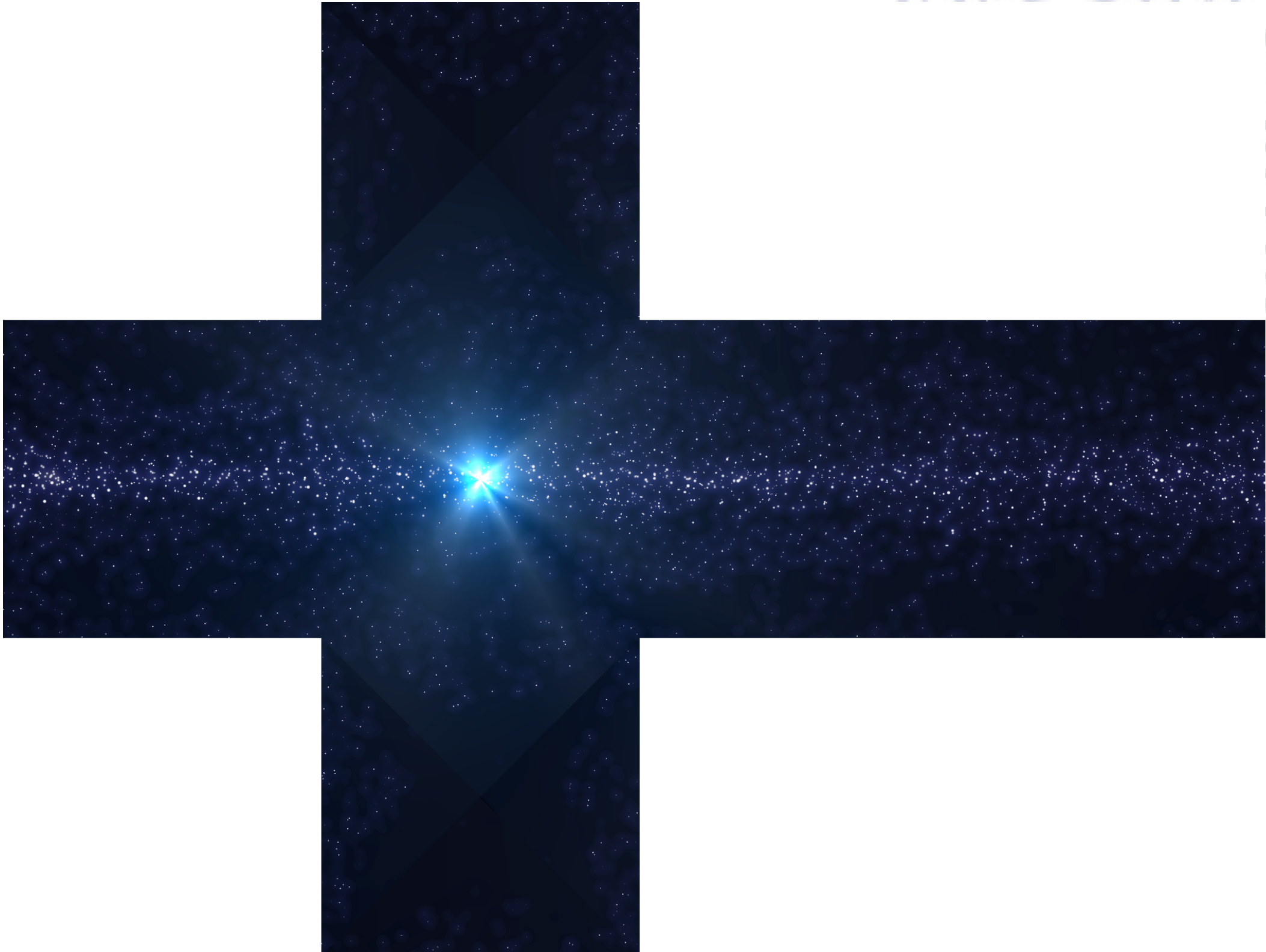
- Inizializzazione texture

```
glGenTextures(1, &textureCube); check(__LINE__);
glActiveTexture(GL_TEXTURE3); check(__LINE__);
std::array<GLenum, 6> targets =
{GL_TEXTURE_CUBE_MAP_POSITIVE_X,
GL_TEXTURE_CUBE_MAP_NEGATIVE_X,
GL_TEXTURE_CUBE_MAP_POSITIVE_Y,
GL_TEXTURE_CUBE_MAP_NEGATIVE_Y,
GL_TEXTURE_CUBE_MAP_POSITIVE_Z,
GL_TEXTURE_CUBE_MAP_NEGATIVE_Z };
glBindTexture(GL_TEXTURE_CUBE_MAP, textureCube);
check(__LINE__) ;
```

Skybox

- Vertex array

```
const GLfloat vertices[] = {
// Position
-1.0f, -1.0f,  1.0f, // 0
 1.0f, -1.0f,  1.0f, // 1
-1.0f,  1.0f,  1.0f, // 2
 1.0f,  1.0f,  1.0f, // 3
 1.0f, -1.0f, -1.0f, // 4
-1.0f, -1.0f, -1.0f, // 5
 1.0f,  1.0f, -1.0f, // 6
-1.0f,  1.0f, -1.0f  // 7
};
const GLuint elements[] = {
0, 2, 3,  0, 3, 1,
1, 3, 6,  1, 6, 4,
2, 7, 3,  2, 6, 3,
5, 7, 2,  5, 2, 0,
0, 1, 5,  5, 1, 4,
4, 6, 5,  5, 6, 7
};
```



Creare un pulsante

- Non abbiamo usato librerie per creare GUI (Graphical User Interface)
  - Ma sappiamo interagire coi disegni mediante il mouse
- Soluzione: creiamo un VAO con un quadrato
  - E usiamo uno shader apposito per renderizzare sopra la texture del pulsante

Pulsante

```
const GLfloat guiVertices[] = {
-1.0f, 1.0f-2.0f*button_size, 0.0f, 0.0f,
-1.0f+2.0f*button_size, 1.0f-2.0f*button_size,
1.0f, 0.0f,
-1.0f+2.0f*button_size, 1.0f, 1.0f, 1.0f,
-1.0f, 1.0f, 0.0f, 1.0f,
};

const GLuint guiElements[] = {
0, 1, 3,
1, 2, 3
};
```

Aggiungere altro modello

- C'è un problema nel gestire i materiali
- Il codice si aspetta di trovare i materiali nel modello. Ma se avete salvato il modello così come abbiamo fatto nella prima lezione non c'erano e il codice darebbe errore in assegnamento
  - Basta creare un file con il materiale
- Copiamo il file .mtl da un esempio visto e modifichiamo a piacere
- Nel file obj però dobbiamo aggiungere (con l'editor di testo) una riga con il nome del file. Nell'esempio
  - mtllib ship.mtl
- Inoltre aggiungiamo il nome del materiale
  - usemtl materiale

Creare modelli

- Modello creato con OpenSCAD (ricordate?)
  - Molto semplice
  - Purrtoppo non salva in obj ma potete facilmente convertire con meshlab
- Con meshlab (caricare, modificare, esportare) o meshlabserver da linea
  - meshlabserver -i file.off -o file.obj
- Poi create (copiando e modificando) il file .mtl e aggiungete le linee viste prima
- Dobbiamo usare un singolo materiale per modello con il codice attuale

Materiale

- Aggiungere nel file obj (salvato in modalità testo)

```
mtllib ship.mtl
usemtl materiale
```

Esercizio finale

1) Mettete un vostro modello di astronave, creato con openScad al posto del disco volante
2) Spostare il sistema solare in modo che inizialmente il sole sia alle coordinate -2,5,-15
3) Aggiungere la vista di un piano (un quadrato grande) che rappresenti un terreno sopra cui l'astronave vola. Questo sia renderizzato con texture mapping dell'immagine ground.png ripetuta periodicamente. Si faccia collision detection dell'astronave sul piano, interrompendo il gioco se l'astronave lo tocca.
4) Aggiungere un secondo oggetto creato da modello a vostro piacimento, posizionato inizialmente alle coordinate 2,3,-10 che rappresenti il traguardo da raggiungere. Il gioco si deve interrompere quando l'astronave collide con il traguardo, indicando la vittoria

Esercizio finale

5) Create uno shader apposito per il traguardo che ne cambi il colore in funzione del tempo, a vostro piacimento

- Correzione in presenza Giovedì 8/6 dalle 10.30
  - Verranno fatte semplici domande per verificare se sapete modificare il codice
  - Chi non potesse partecipare alla correzione/esamino dell'8, può concordare un appuntamento per la correzione in data antecedente

- Chi non può svolgere la prova, può concordare a parte un progettino sostitutivo
- Giovedì 8/6 dalle 1030 ci sarà anche il recupero del primo esercizio. Sarà assegnato un breve esercizio da svolgere in aula con consegna entro le 13:00