Giovanni Baratta - 0000853633 - Laboratorio 3

Caricamento e visualizzazione di più di un oggetto mesh .obj con la possibilità di passare la selezione dall'uno all'altro tramite special key arrows

Per gestire più modello è stato utilizzato un vettore di elementi ognugno dei quali dispone di una propria matrice per le trasformazioni in OCS, una mesh associata ed un materiale. Tramite un indice viene memorizzato l'oggetto attualmente attivo sulla quale applicare le trasfromazioni e le modifiche. Tramite le frecce direzionali si modifca l'indice dell'oggetto selezionato.

Calcolo e memorizzazione delle normali ai vertici per i modelli mesh poligonali

Durante il caricamento della mesh vengono calcolate le normali ai vertici. Per ogni vertice vengono sommate le normali dei relativi triangoli, ed al termine viene effettuata la normalizzazione.

```
    for (int index = 0; index < mesh->indices.size(); index = index + 3)
    {
        // INDICI DEI VERTICI
        int i1 = mesh->indices.at(index);
        int i2 = mesh->indices.at(index + 1);
        int i3 = mesh->indices.at(index + 2);

        // VERTICI
        glm::vec3 v1 = mesh->vertices[i1];
        glm::vec3 v2 = mesh->vertices[i2];
        glm::vec3 v3 = mesh->vertices[i3];

        glm::vec3 lato1 = glm::vec3(v1) - glm::vec3(v2);
        glm::vec3 lato2 = glm::vec3(v1) - glm::vec3(v3);
        glm::vec3 normale = glm::normalize(glm::cross(lato1, lato2));
        mesh->normals[i1] += normale;
        mesh->normals[i2] += normale;
        mesh->normals[i3] += normale;
    }

    // rinormalizzo tutti
    for (int index = 0; index < mesh->normals.size(); index++)
        mesh->normals[index] = glm::normalize(mesh->normals[index]);
```

Navigazione interattiva in scena

Per implementare il panning, in tutti e 4 i casi, sono state trasformate le coordinate in WCS della camera e del punto di lookAt nel sistema di riferimento VCS.

```
glm::mat4 vcsToWcsMatrix() {
    glm::vec3 vup = ViewSetup.upVector;
    glm::vec3 w = glm::normalize(ViewSetup.position - ViewSetup.target);
    glm::vec3 u = glm::normalize(glm::cross(vup, w));
    glm::vec3 v = glm::cross(w, u);

    glm::mat4 vcsToWcs = glm::transpose(
        glm::mat4(
            u.x, v.x, w.x, ViewSetup.position.x,
            u.y, v.y, w.y, ViewSetup.position.y,
            u.z, v.z, w.z, ViewSetup.position.z,
            0.0f, 0.0f, 0.0f, 1)
    );

    return vcsToWcs;
}
```

Una volta ottenute le coordinate in VCS, a seconda del tipo di panning, è stato aggiunto un offset nella direzione opportuna, infine sono state convertite nuovamente in WCS le coordinate.

```
    glm::mat4 vcsToWcs = vcsToWcsMatrix();
    glm::mat4 wcsToVcs = glm::inverse(vcsToWcs);

    glm::vec4 vcsPos = wcsToVcs * ViewSetup.position;
    glm::vec4 vcsTarget = wcsToVcs * ViewSetup.target;

    glm::vec4 newPos = vcsPos + glm::vec4(+CAMERA_SHIFT, 0.0, 0.0, 0.0);
    glm::vec4 newTarget = vcsTarget + glm::vec4(+CAMERA_SHIFT, 0.0, 0.0, 0.0);

    glm::vec4 wcsNewPos = vcsToWcs * newPos;
    glm::vec4 wcsNewTarget = vcsToWcs * newTarget;

    ViewSetup.position = wcsNewPos;
    ViewSetup.target = wcsNewTarget;
```

Per realizzare il `moveForeward` e il `moveBack` si è semplicemente spostate la camera lungo l'asse w della camera.

```
    glm::vec3 w = glm::normalize(ViewSetup.position - ViewSetup.target);
    glm::vec4 newPos = ViewSetup.position - glm::vec4(w * CAMERA_SHIFT, 0.0);
    ViewSetup.position = newPos;
```

Camera motion

Per realizzare il movimento della camera lungo un percorso, è stata definita una curva di Bezièr, e tramite l'algoritmo di De Casteljau implementato nell'esercitazione 1, ad ogni aggiornamento è stata valutata la curva. In base al tempo passato dalla valutazione precedente è stato scelto un punto successivo per la valutazione. Il valore ottenuto è stato utilizzato per spostare la posizione della telecamera.

```
    if (cameraPathEnabled) {
        float newUpdate = currentTime();
        pathCovered += (newUpdate - lastUpdate) * CAMERA_SPEED / 1000;
        if (pathCovered > 1)
            pathCovered -= static_cast<int>(pathCovered);
        deCasteljau(pathCovered, deCasteljauResult);
        ViewSetup.position =
            glm::vec4(deCasteljauResult[0],deCasteljauResult[1], deCasteljauResult[2], 1.0);
        lastUpdate = newUpdate;
    }
```

La funzionalità può essere attivata tramite il menu.

Trasformazione degli oggetti in scena

Per effettuare le trasformazioni con il sistema di riferimento OCS sono state semplicemente applicate le relative trasformazioni di traslazione, rotazione e scaling alla matrice associata ad ogni oggetto. Nel caso di trasformazioni in WCS le operazioni sono state applicate prima di moltiplicare la matrice di trasformazione da OCS a WCS.

```
    glPushMatrix();
    glLoadIdentity();

    switch (TransformMode) {
    case WCS:
        glTranslatef(translation_vector.x, translation_vector.y, translation_vector.z);
        glRotatef(angle, rotation_vector.x, rotation_vector.y, rotation_vector.z);
        glScalef(scale_factor, scale_factor, scale_factor);
        glMultMatrixf(objects.at(selectedObject).model_matrix);
        break;
    case OCS:
        glLoadMatrixf(objects.at(selectedObject).model_matrix);
        glTranslatef(translation_vector.x, translation_vector.y, translation_vector.z);
        glRotatef(angle, rotation_vector.x, rotation_vector.y, rotation_vector.z);
        glScalef(scale_factor, scale_factor, scale_factor);
        break;
    }
    glGetFloatv(GL_MODELVIEW_MATRIX, objects[selectedObject].model_matrix);
    glPopMatrix();
```