ΔΗΜΟΚΡΙΤΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΡΑΚΗΣ

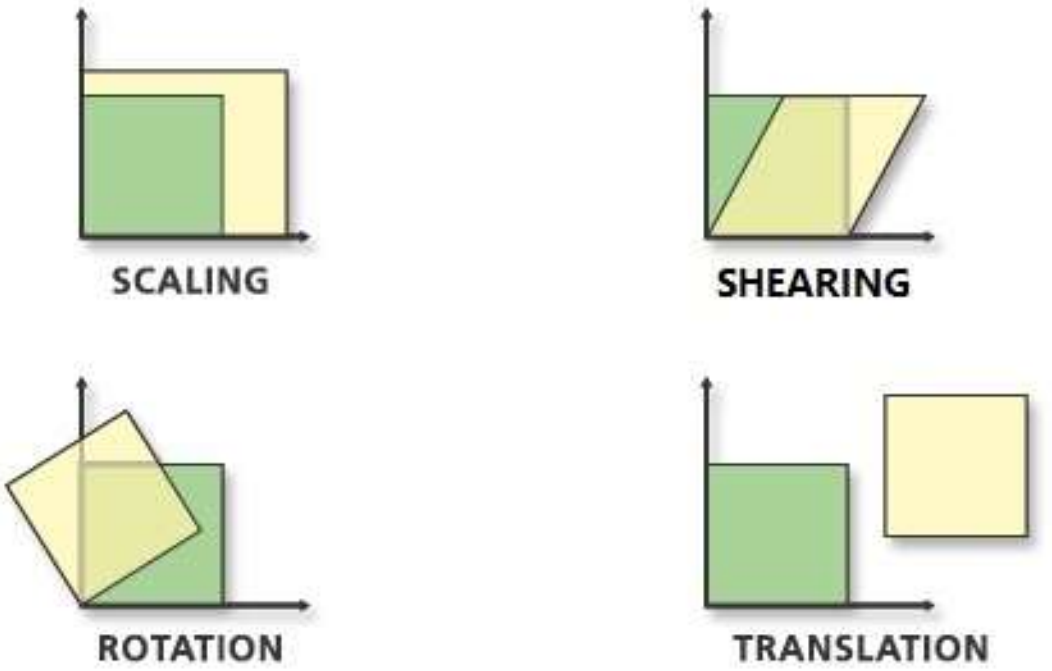
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Εισαγωγή στα Γραφικά Υπολογιστών

Εργαστήριο 3

Μετασχηματισμοί

- Μεταφορά
- Περιστροφή
- Κλιμάκωση
- Στρέβλωση



Ομογενείς συντεταγμένες

- Επιτρέπουν την εφαρμογή όλων των μετασχηματισμών με πολλαπλασιασμό πινάκων

- Κορυφές (x, y, z, w) → 3Δ συντεταγμένες (x/w, y/w, z/w)

- w = 1 για κορυφές (vertices)
- w = 0 για διανύσματα (vectors)

Μετασχηματισμοί

- Βιβλιοθήκη GLM: OpenGL Mathematics

- Προσθέστε τον φάκελο “glm” στο working directory (φάκελος του Project).

```
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtc/type_ptr.hpp>
```

Μετασχηματισμοί

- Ας δοκιμάσουμε να μετασχηματίσουμε μία κορυφή με συντεταγμένες (1, 0, 0), με μεταφορά κατά (1, 1, 0):

```
glm::vec4 vec(1.0f, 0.0f, 0.0f, 1.0f);
glm::mat4 trans;
trans = glm::translate(trans, glm::vec3(1.0f, 1.0f, 0.0f));
vec = trans * vec;
std::cout << "(" << vec.x << ", " << vec.y << ", " << vec.z << ")" << std::endl;
```

Μετασχηματισμοί

- Κλιμάκωση κατά παράγοντα 0.5 και περιστροφή γύρω από τον z:

```
glm::mat4 trans;
trans = glm::rotate(trans, glm::radians(90.0f), glm::vec3(0.0, 0.0, 1.0));
trans = glm::scale(trans, glm::vec3(0.5, 0.5, 0.5));
```

- Οι μετασχηματισμοί εφαρμόζονται πάντα με την αντίστροφη σειρά

Μετασχηματισμοί

- Δημιουργία μιας uniform μεταβλητής στον vertex shader:

```
#version 330 core
layout (location = 0) in vec3 in_position;
uniform mat4 transform;

void main()
{
   gl_Position = transform * vec4(in_position, 1.0);
}
```

Μετασχηματισμοί

- Τέλος, μεταφέρουμε τον πίνακα μετασχηματισμού από το πρόγραμμα στον vertex shader, μέσα στο rendering loop:

```
GLuint transformLoc = glGetUniformLocation(myShader.ID, "transform");
glUniformMatrix4fv(transformLoc, 1, GL_FALSE, glm::value_ptr(trans));
```

Άσκηση

- Περιστροφή του τριγώνου στο χρόνο.

Μετασχηματισμοί

- Η σειρά των μετασχηματισμών έχει σημασία:

```
glm::mat4 trans;
trans = glm::translate(trans, glm::vec3(0.5f, 0.0f, 0.0f));
trans = glm::rotate(trans, (float)glfwGetTime(), glm::vec3(0.0f, 0.0f, 1.0f));
trans = glm::scale(trans, glm::vec3(0.5, 0.5, 0.5));

//translation first or rotation first??
```

Άσκηση

- Σχεδιάστε δύο τρίγωνα σε δύο διαφορετικές θέσεις χρησιμοποιώντας μετασχηματισμούς.

- Το δεύτερο τρίγωνο να αλλάζει μέγεθος με το χρόνο (να μικραίνει και να μεγαλώνει εναλλάξ, χρησιμοποιήστε τη συνάρτηση sin())

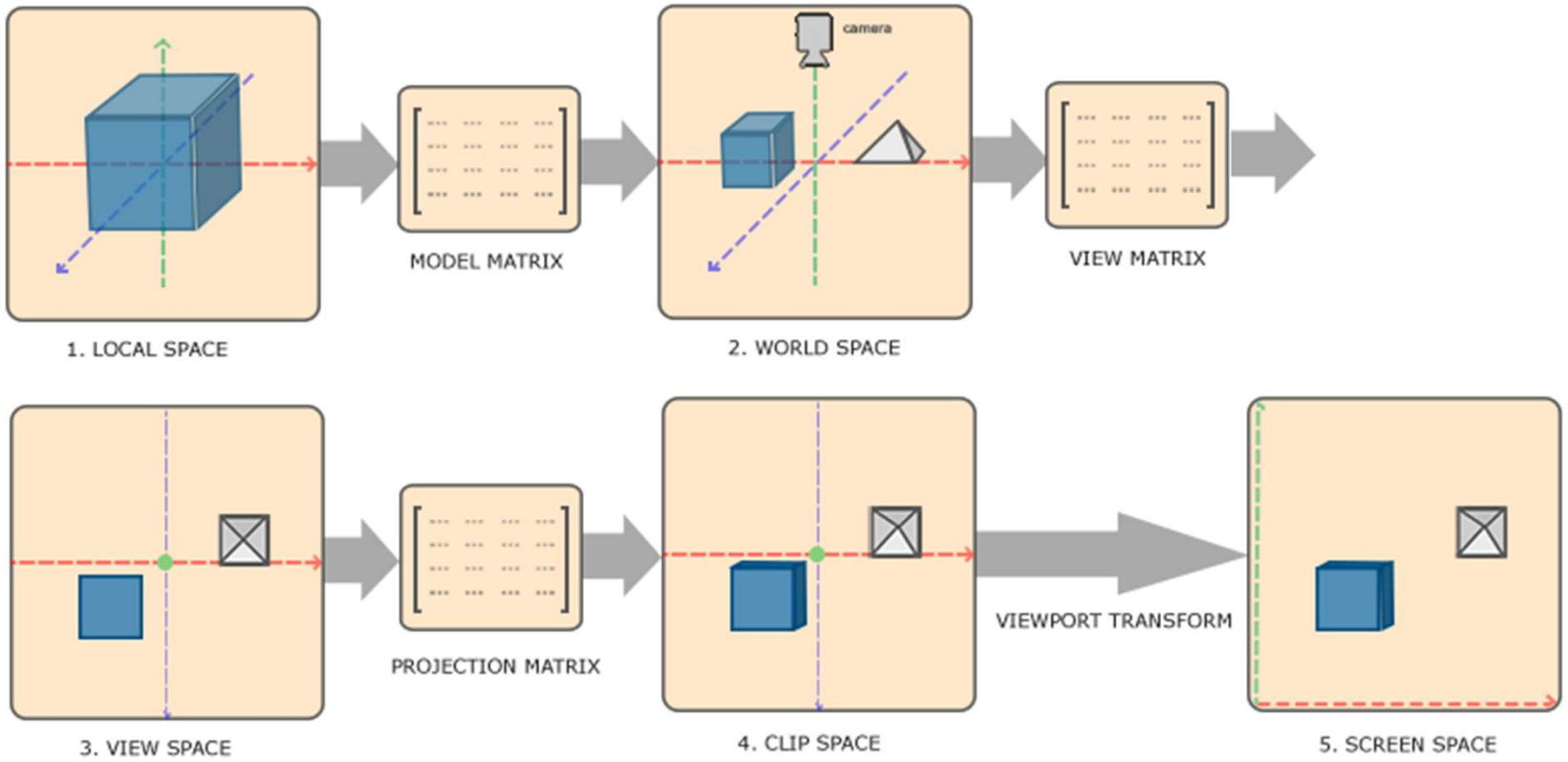
Συστήματα συντεταγμένων

- Μέχρι στιγμής: Normalized Device Coordinates [-1.0, 1.0]

- Συνήθως ορίζουμε τις συντεταγμένες των αντικειμένων σε ένα εύρος που καθορίζουμε μόνοι μας και τις μετασχηματίζουμε σε NDC στον vertex shader

- 5 σημαντικά συστήματα συντεταγμένων:
  - Τοπικό σ.σ. (Local space ή Object space)
  - Σ.σ. κόσμου (World space)
  - Σ.σ. θέασης (View space ή Eye space ή Camera space)
  - Σ.σ. αποκοπής (Clip space)
  - Σ.σ. της οθόνης (Screen space)

Συστήματα συντεταγμένων



Συστήματα συντεταγμένων

- **Πίνακας μοντέλου (Model Matrix)**: μετασχηματισμός κάθε αντικειμένου για την τοποθέτησή του στον κόσμο

- **Πίνακας θέασης (View Matrix)**: μετασχηματισμός ώστε να φαίνεται ο κόσμος από την άποψη της κάμερας

- **Πίνακας προβολής (Projection Matrix)**: μετασχηματισμός των συντεταγμένων ενός εύρους (ορισμένο από μας) σε NDC [-1.0, 1.0], συντεταγμένες εκτός του εύρους αυτού αποκόβονται (clipped)

Συστήματα συντεταγμένων

- Το στερεό που περιέχει τις συντεταγμένες που θα εμφανιστούν στην οθόνη ονομάζεται **στερεό θέασης (viewing frustum)**

- Αφού έχουν μετασχηματιστεί όλες οι κορυφές στο σ.σ. αποκοπής (clip space), **προοπτική διαίρεση (perspective division)**: διαίρεση των x, y, z με την ομογενή συντεταγμένη w:

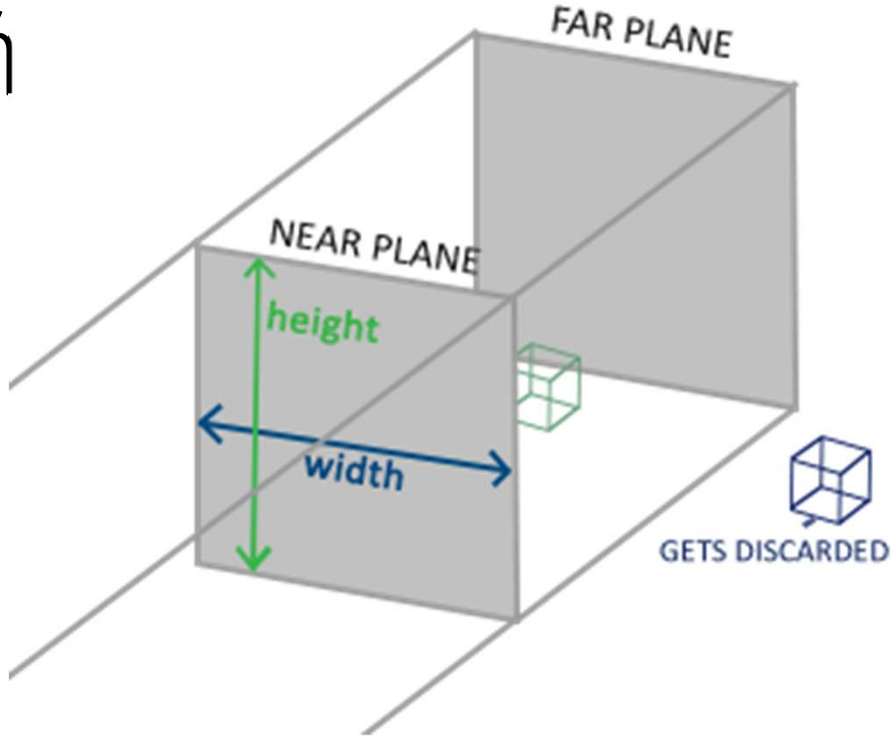
$$V_{out} = \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} x/w \\ y/w \\ z/w \end{pmatrix}$$

- Η προοπτική διαίρεση μετασχηματίζει τις συντεταγμένες από το 4Δ σ.σ. αποκοπής σε 3Δ συντεταγμένες NDC

Ορθογραφική προβολή

- Στερεό θέασης: ορθογώνιο παραλληλεπίπεδο

- Συντεταγμένη w: σταθερή, ανεξάρτητη της απόστασης από την κάμερα

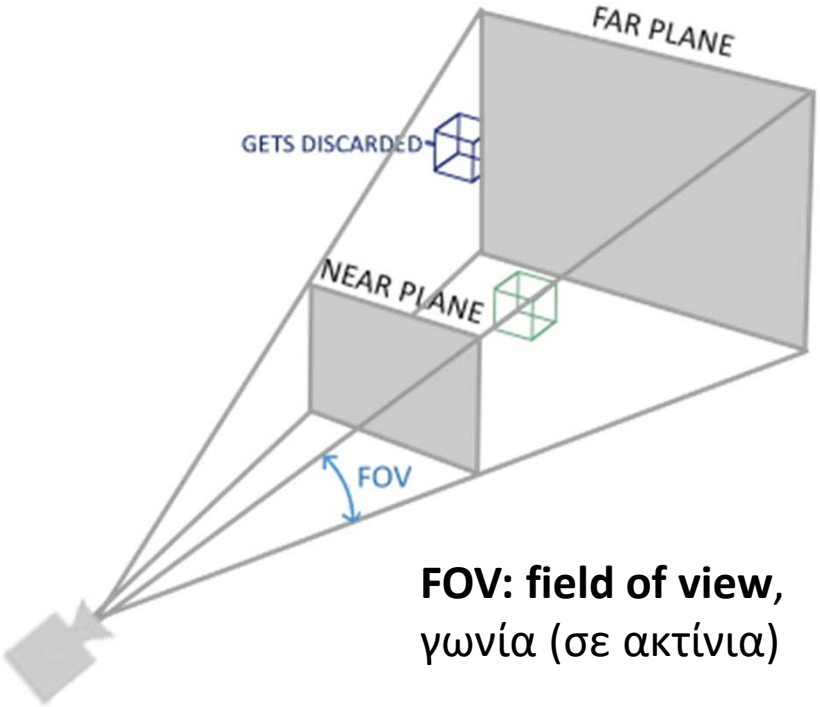


glm::ortho(GLfloat left, GLfloat right, GLfloat bottom, GLfloat top, GLfloat near, GLfloat far)

Προοπτική προβολή

- Στερεό θέασης: Κόλουρη πυραμίδα

- Συντεταγμένη w: αυξάνεται όσο αυξάνεται η απόσταση από την κάμερα



FOV: field of view, γωνία (σε ακτίνια)

glm::perspective(GLfloat FOV, GLfloat aspectRatio, GLfloat near, GLfloat far)

Συστήματα συντεταγμένων

- Συνολικά: πίνακας MVP (Model – View – Projection):

$$V_{clip} = M_{projection} \cdot M_{view} \cdot M_{model} \cdot V_{local}$$

```
uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;

void main()
{
   gl_Position = projection * view * model * vec4(in_position, 1.0);
}
```

Συστήματα συντεταγμένων

- Ένα ορθογώνιο:

```
GLfloat vertices[] =
{    //position      //color
     // first triangle
     0.5f, 0.5f, 0.0f, 1.0f, 1.0f, 0.0f,
     0.5f, -0.5f, 0.0f, 1.0f, 0.0f, 0.0f,
      -0.5f, 0.5f, 0.0f, 0.0f, 1.0f, 0.0f,
     // second triangle
     0.5f, -0.5f, 0.0f, 1.0f, 0.0f, 0.0f,
     -0.5f, -0.5f, 0.0f, 0.0f, 0.0f, 1.0f,
     -0.5f, 0.5f, 0.0f, 0.0f, 1.0f, 0.0f
};
```

Συστήματα συντεταγμένων

- Περιστροφή του μοντέλου γύρω από τον άξονα των X:

- Πίνακας μοντέλου:

```
glm::mat4 model = glm::mat4(1.0f);
model = glm::rotate(model, glm::radians(-75.0f), glm::vec3(1.0f, 0.0f, 0.0f));
```

Συστήματα συντεταγμένων

- Αντί να μετακινήσουμε την κάμερα, το ίδιο αποτέλεσμα προκύπτει αν μετακινήσουμε όλα τα αντικείμενα στην αντίθετη κατεύθυνση

- Πίνακας θέασης:

```
glm::mat4 view = glm::mat4(1.0f);
view = glm::translate(view, glm::vec3(0.0f, 0.0f, -3.0f));
//the reverse direction of where we want to move
```

Συστήματα συντεταγμένων

- Πίνακας προβολής:

```
glm::mat4 projection;
//perspective
projection = glm::perspective(glm::radians(45.0f), screenWidth / screenHeight, 0.1f, 100.0f);

//default: orthographic
//projection = glm::ortho(-1.0f, 1.0f, -1.0f, 1.0f, -1.0f, 1.0f);
```

3Δ δεδομένα

- Ένας κύβος με διαφορετικό χρώμα σε κάθε πλευρά
- 36 κορυφές
- Γιατί να μην χρησιμοποιήσουμε ένα EBO?

```
GLfloat vertices[] = {
    //position          //color
   -0.5f, -0.5f, -0.5f,  1.0f, 0.0f, 0.0,
    0.5f, -0.5f, -0.5f,  1.0f, 0.0f, 0.0,
    0.5f,  0.5f, -0.5f,  1.0f, 0.0f, 0.0,
    0.5f,  0.5f, -0.5f,  1.0f, 0.0f, 0.0,
   -0.5f,  0.5f, -0.5f,  1.0f, 0.0f, 0.0,
   -0.5f, -0.5f, -0.5f,  1.0f, 0.0f, 0.0,

   -0.5f, -0.5f,  0.5f,  0.0f, 1.0f, 0.0,
    0.5f, -0.5f,  0.5f,  0.0f, 1.0f, 0.0,
    0.5f,  0.5f,  0.5f,  0.0f, 1.0f, 0.0,
    0.5f,  0.5f,  0.5f,  0.0f, 1.0f, 0.0,
   -0.5f,  0.5f,  0.5f,  0.0f, 1.0f, 0.0,
   -0.5f, -0.5f,  0.5f,  0.0f, 1.0f, 0.0,

   -0.5f,  0.5f,  0.5f,  0.0f, 0.0f, 1.0f,
   -0.5f,  0.5f, -0.5f,  0.0f, 0.0f, 1.0f,
   -0.5f, -0.5f, -0.5f,  0.0f, 0.0f, 1.0f,
   -0.5f, -0.5f, -0.5f,  0.0f, 0.0f, 1.0f,
   -0.5f, -0.5f,  0.5f,  0.0f, 0.0f, 1.0f,
   -0.5f,  0.5f,  0.5f,  0.0f, 0.0f, 1.0f,

    0.5f,  0.5f,  0.5f,  0.0f, 0.5f, 0.0f,
    0.5f,  0.5f, -0.5f,  0.0f, 0.5f, 0.0f,
    0.5f, -0.5f, -0.5f,  0.0f, 0.5f, 0.0f,
    0.5f, -0.5f, -0.5f,  0.0f, 0.5f, 0.0f,
    0.5f, -0.5f,  0.5f,  0.0f, 0.5f, 0.0f,
    0.5f,  0.5f,  0.5f,  0.0f, 0.5f, 0.0f,

   -0.5f, -0.5f, -0.5f,  0.0f, 0.0f, 0.5f,
    0.5f, -0.5f, -0.5f,  0.0f, 0.0f, 0.5f,
    0.5f, -0.5f,  0.5f,  0.0f, 0.0f, 0.5f,
    0.5f, -0.5f,  0.5f,  0.0f, 0.0f, 0.5f,
   -0.5f, -0.5f,  0.5f,  0.0f, 0.0f, 0.5f,
   -0.5f, -0.5f, -0.5f,  0.0f, 0.0f, 0.5f,

   -0.5f,  0.5f, -0.5f,  0.5f, 0.0f, 0.0f,
    0.5f,  0.5f, -0.5f,  0.5f, 0.0f, 0.0f,
    0.5f,  0.5f,  0.5f,  0.5f, 0.0f, 0.0f,
    0.5f,  0.5f,  0.5f,  0.5f, 0.0f, 0.0f,
   -0.5f,  0.5f,  0.5f,  0.5f, 0.0f, 0.0f,
   -0.5f,  0.5f, -0.5f,  0.5f, 0.0f, 0.0f
};
```

Depth Testing

- Η OpenGL αποθηκεύει την τιμή βάθους για κάθε θέση pixel στον Z buffer (ή depth buffer)

- Κάθε fragment έχει την θέση του, τιμή χρώματός του και τιμή βάθους του

- Κάθε fragment συγκρίνει την τιμή βάθους του με την τρέχουσα τιμή του Z buffer στη θέση αυτή:
  - Μικρότερη, τιμή χρώματος→Color buffer και τιμή βάθους→Z buffer
  - Μεγαλύτερη, το fragment απορρίπτεται (discarded)

```
glEnable(GL_DEPTH_TEST);
```

OpenGL Buffers

- Color buffer:

```
glClear(GL_COLOR_BUFFER_BIT);
```

- Z buffer:

```
glClear(GL_DEPTH_BUFFER_BIT);
```

- Μαζί:

```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

Άσκηση

- Περιστροφή του κύβου στο χρόνο γύρω από τυχαίο άξονα

Περισσότερα αντικείμενα

```
glm::vec3 cubePositions[] = {
  glm::vec3( 0.0f,  0.0f,  0.0f),
  glm::vec3( 2.0f,  5.0f, -15.0f),
  glm::vec3(-1.5f, -2.2f, -2.5f),
  glm::vec3(-3.8f, -2.0f, -12.3f),
  glm::vec3( 2.4f, -0.4f, -3.5f),
  glm::vec3(-1.7f,  3.0f, -7.5f),
  glm::vec3( 1.3f, -2.0f, -2.5f),
  glm::vec3( 1.5f,  2.0f, -2.5f),
  glm::vec3( 1.5f,  0.2f, -1.5f),
  glm::vec3(-1.3f,  1.0f, -1.5f)
};
```

Περισσότερα αντικείμενα

```
for(unsigned int i = 0; i < 10; i++)
{
  glm::mat4 model = glm::mat4(1.0f);
  model = glm::translate(model, cubePositions[i]);
  float angle = 20.0f * i;
  model = glm::rotate(model, glm::radians(angle), glm::vec3(1.0f, 0.3f, 0.5f));

  //…

  glDrawArrays(GL_TRIANGLES, 0, 36);
}
```