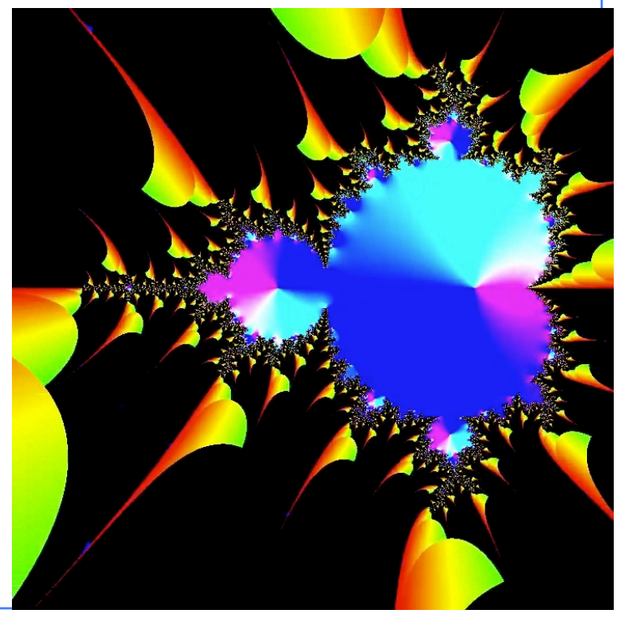
TSBK07/TSBK11

Computer Graphics

Ingemar Ragnemalm, ISY





Lecture 3

3D concepts

3D transformations

Viewing

Projection

Lecture questions

1. What is the difference between a right-hand and a left-hand system?

2. Why did the minus sign end up in the wrong place for the rotation around Y?

3. Why do we multiply R*T when placing the camera?

4. How do the homogenous coordinates help for projection?

5. What is a viewing *frustum*?

Lab 1

- Monday
- Booked 13-17, Asgård, Olympen and Egypten
- Starts with a dugga
- Lab material will be updated but works as is.
- We (hopefully 2 persons per lab) will be available for questions and demonstrations.

A little bit from last time. Remember sending matrices directly?

In practice: Hide common code

```
mat4 Rz(GLfloat a)
{
  mat4 m;
  m.m[0] = cos(a); m.m[1] = -sin(a); m.m[2] = 0;    m.m[3] = 0.0;
  m.m[4] = sin(a); m.m[5] = cos(a);  m.m[6] = 0;    m.m[7] = 0.0;
  m.m[8] = 0;      m.m[9] = 0;       m[10] = 1.0;   m.m[11] = 0.0;
  m.m[12] = 0.0;   m.m[13] = 0.0;    m.m[14] = 0.0; m.m[15] = 1.0;

  return m;
}
```

VectorUtils4 is used in the lab for this.

The mat4 structure

The type mat4 is a struct containing an array! It looks basically like this:

```
typedef struct mat4
{
    GLfloat m[16];

    (A bunch of initializers here)
} mat4;
```

You can create a mat4 directly from data like this:

```
mat4 myMatrix = mat4(1, 0, 0, 0,
                     0, 1, 0, 0,
                     0, 0, 1, 0,
                     0, 0, 0, 1);
```

and you can access any element as myMatrix.m[i]

Multiple transformation example

Vertex shader still only multiplies one matrix

Uses our simple vector/matrix library ”VectorUtils4”

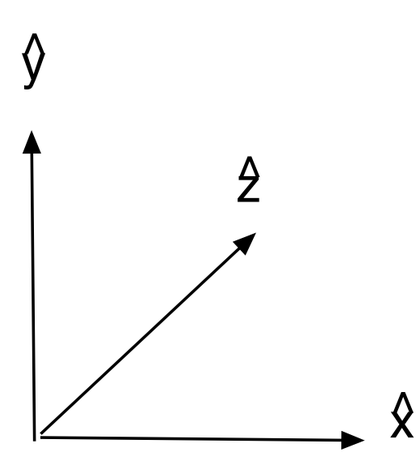
```
total = Rz(Pi/4, rot) * T(0, -1, 0, trans);

uploadMat4ToShader(program, "m", m);
```

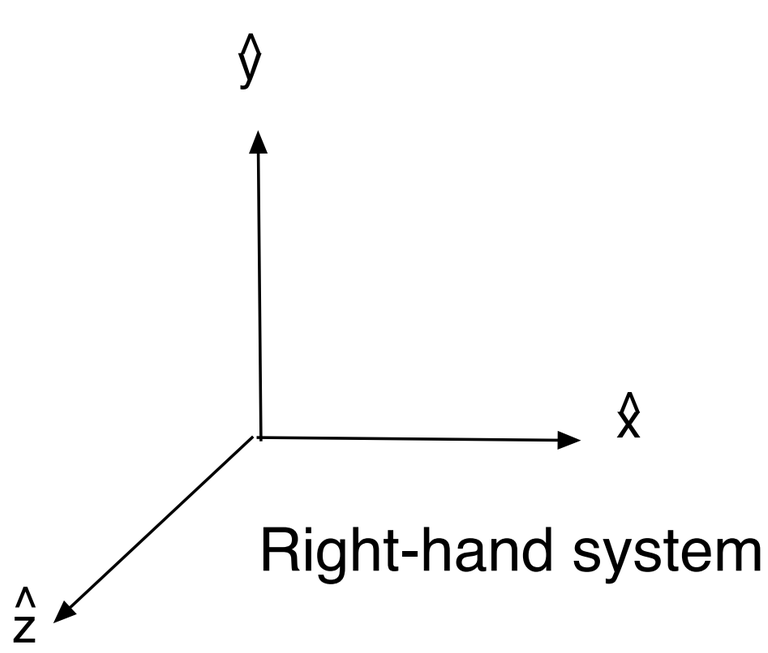
Beware that the order of transformations matter!

Spaceship demo. We used transformations to move it around.

3D coordinate system



Left-hand system



Right-hand system

3D point – (x, y, z) (Positional vector)

Directional vector – (x, y, z)

3D line $\mathbf{p} = \mathbf{p_1} + \mu * \mathbf{d}$

or $\mathbf{p} = \mathbf{p_1} + \mu * (\mathbf{p_2} - \mathbf{p_1})$

3D line segments e.g. $0 < \mu < 1$

3D plane $A*x + B*y + C*z + D = 0$

Properties of 3D planes:

n = (A,B,C) Normal vector

Most transformations are trivially expanded to 3D:

Translation: $T(t_x, t_y) = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$

Translation: $T(t_x, t_y, t_z) = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Scaling: $S(s_x, s_y) = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Scaling: $S(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Rotation

Around the x axis: $R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

$\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Around the y axis: $R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Around the z axis: $R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

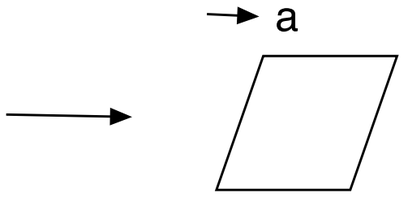
And some more

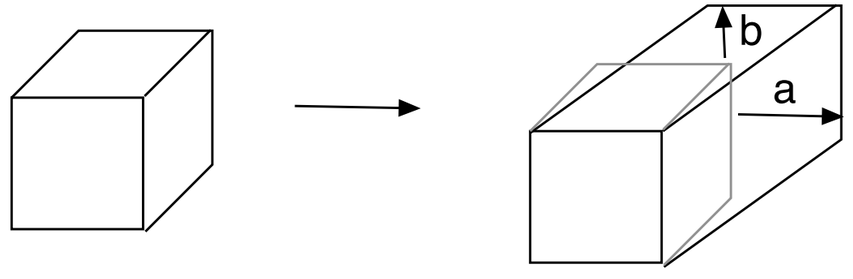
Mirroring: $M_X = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Mirroring: $M_x = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Shearing: $SH_X(a) = \begin{bmatrix} 1 & a & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Shearing: $SH_z(a, b) = \begin{bmatrix} 1 & 0 & a & 0 \\ 0 & 1 & b & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$





In 3D we need a more complex sequence of transformations for:

- 3D viewing
- Camera placement
- Projection

(Whiteboard time)

3D examples

Not extremely different

- Models given with 3D coordinates

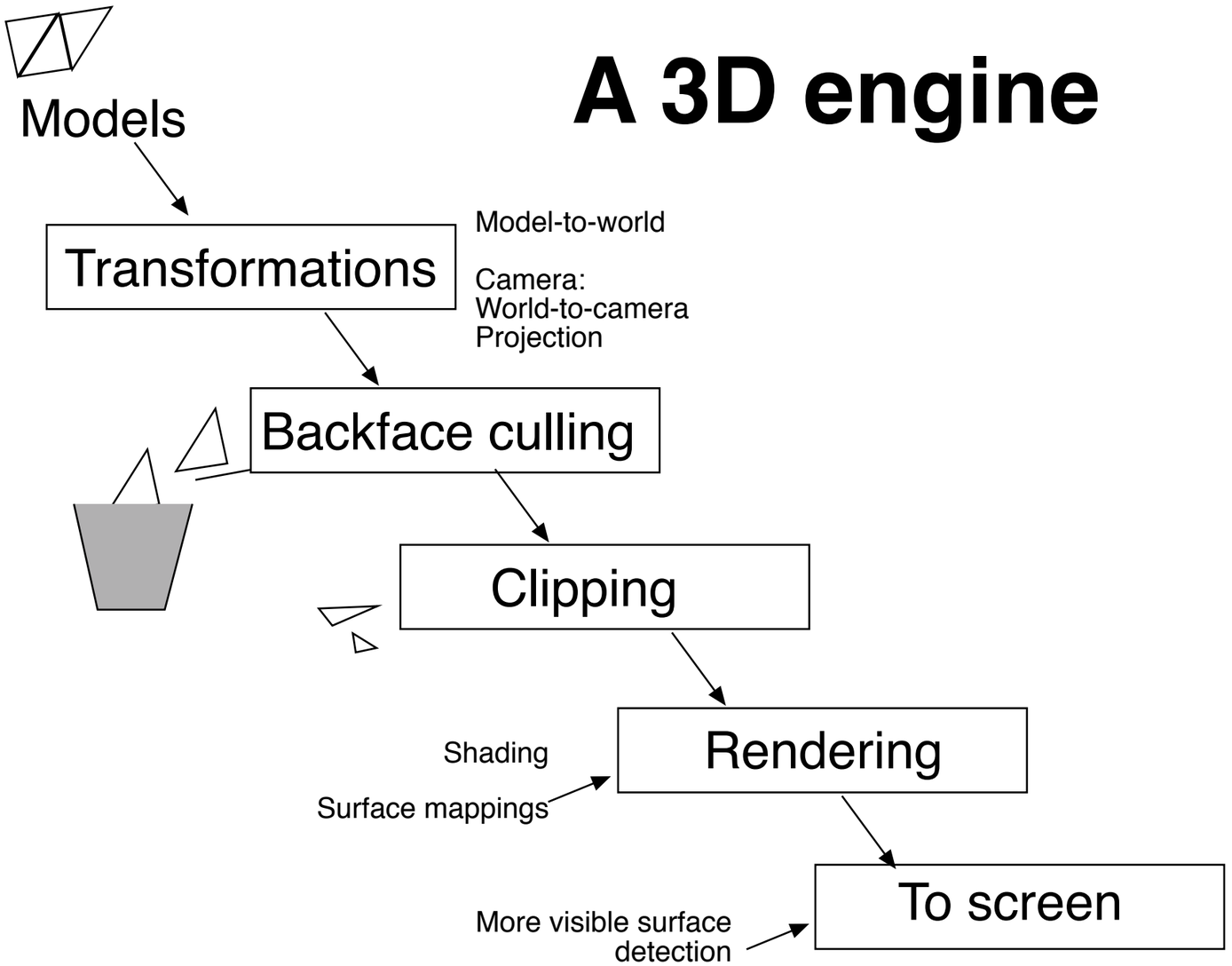
- Projection and ”look-at” matrices

Color cube demo: Solid models more complex.

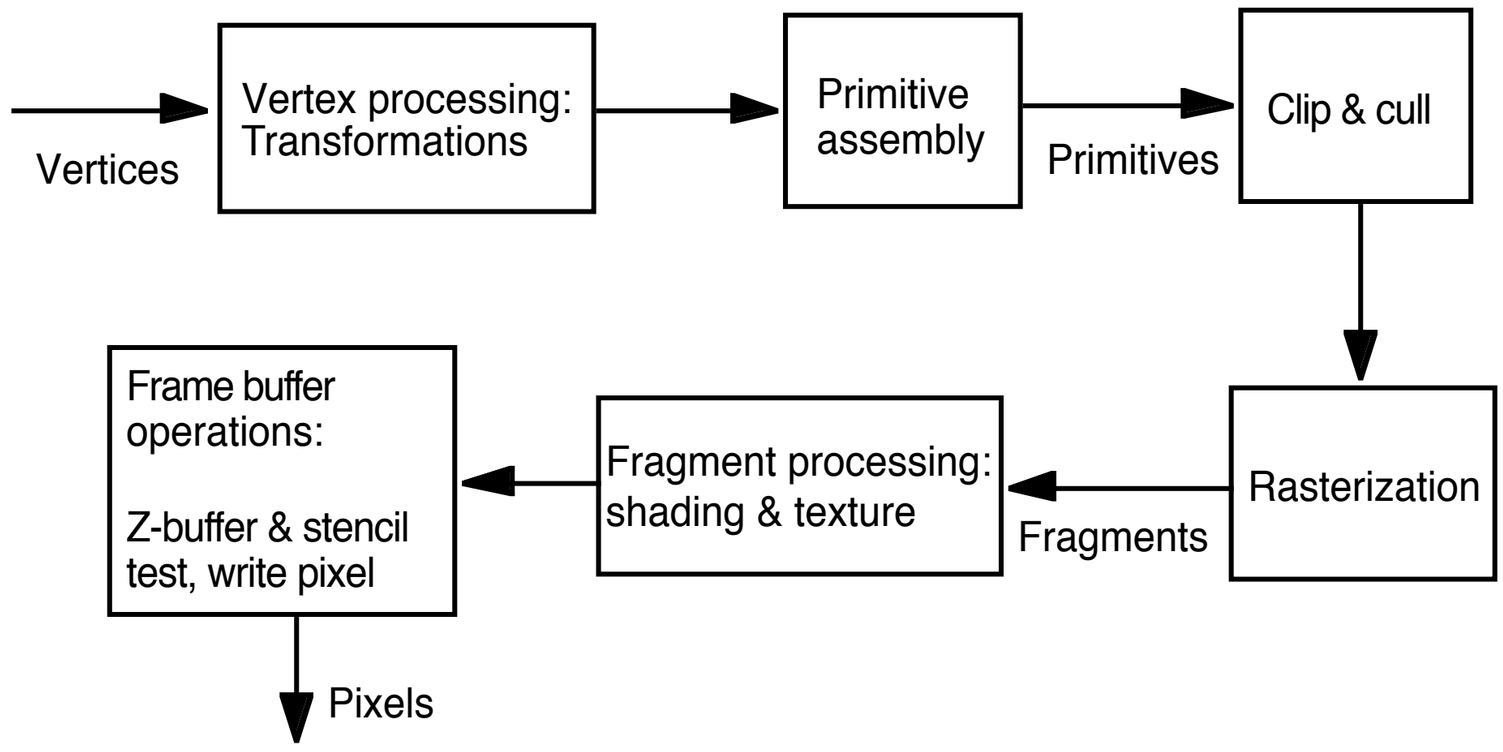
Visible surface detection vital. Try it in lab 1 + details in future lecture.

Also color data per vertex.

A 3D engine


Models
Transformations
Model-to-world
Camera:
World-to-camera
Projection
Backface culling
Clipping
Rendering
Shading
Surface mappings
To screen
More visible surface
detection

The OpenGL pipeline



Next lecture

The OpenGL pipeline

The OpenGL Shading Language

3D object representation

Illumination