# The Model Matrix

## Lecture 7

Robb T. Koether

Hampden-Sydney College

Mon, Sep 4, 2017

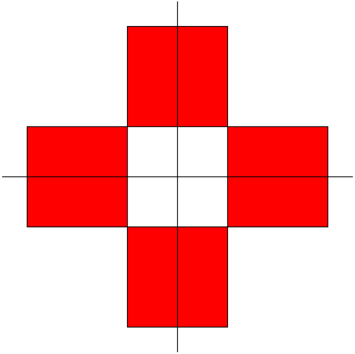
# Outline

# Outline

# The Model Coordinate System

- When we create an object, such as a square or a circle, we use the coordinate system and position that are most convenient.
  - To create a square, we might place the lower-left corner at $(0, 0)$ and let the side be 1.
  - To create a circle, we would place the center at $(0, 0)$ and let the radius be 1.
- That coordinate system is call the model coordinate system and it is specific to each object.

# The Model Coordinate System



- For example, suppose that we want to draw four squares as shown.

# The Model Coordinate System

- Should we construct 4 separate squares in four separate buffers?

# The Model Coordinate System

- Should we construct 4 separate squares in four separate buffers?
- Or should we construct one square and draw it 4 times, in 4 different locations?

# The Model Coordinate System

- Should we construct 4 separate squares in four separate buffers?
- Or should we construct one square and draw it 4 times, in 4 different locations?
- How do we change the location (in world coordinates) of an object?

# Outline

# The Model Matrix

- The model matrix is a matrix that represents a geometric transformation that will move or modify (i.e., transform) an object from its model coordinates to world coordinates.
- The model matrix consists of any combination of
  - Translations – slide in a given direction.
  - Rotations – rotate about a given axis.
  - Scalings – stretch or shrink by a given factor.
  - Or any other transformation that can be represented by a matrix.

## The Model Matrix

```
mat4 model = ··· // Create the model matrix
GLuint model_loc = glGetUniformLocation(program, "model");
glUniformMatrix4fv(model_loc, 1, GL_FALSE, model);
```

- The model matrix, like the projection matrix, must be passed to the vertex shader.
- The vertex shader will apply it, along with the projection matrix, to the vertex.

## The Vertex Shader

```
#version 450 core

uniform mat4 model;
uniform mat4 proj;

out vec4 color;

layout (location = 0) in vec2 vPosition;
layout (location = 1) in vec3 vColor;

void main()
{
    gl_Position = proj*model*vec4(vPosition, 0.0f, 1.0f);
    color = vec4(vColor, 1.0f);
}
```

# Outline

# Translations

## Translations

```
mat4 translate(float dx, float dy, float dz);
```

- The `translate()` function will return a translation matrix.
- The $x$, $y$, and $z$ coordinates will be shifted by the amounts `dx`, `dy`, and `dz`, respectively.
- See `vmath.h` for details.

# Translations

$$\mathbf{T} = \begin{pmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

# Outline

# Rotations

## Rotations

```
mat4 rotate(float angle, float ax, float ay, float az);
```

- The `rotate()` function will return a rotation matrix.
- The object will be rotated through the given angle and about an axis through the origin and the given point (ax, ay, az).
- The direction of rotation is determined by the right-hand rule: point your right thumb in the direction from the origin to the point and curl your fingers.
- See `vmath.h` for details.

# Rotations About the $z$-Axis

$$\mathbf{R_z} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

# Outline

# Scalings

## Scalings

```
mat4 scale(float sx, float sy, float sz);
```

- The `scale()` function will return a scaling matrix.
- The object will be stretched or shrunk by factors `sx`, `sy`, and `sz` in the $x$, $y$, and $z$ directions, respectively.
- If one of the values is $-1$ and the other two are 1, then the scaling will be a reflection.
- None of `sx`, `sy`, and `sz` should ever be `0`.
- See `vmath.h` for details.

# Scalings

$$\mathbf{S} = \begin{pmatrix} sx & 0 & 0 & 0 \\ 0 & sy & 0 & 0 \\ 0 & 0 & sz & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

# Outline

# Sequences of Transformations

- In most cases, an object will go through a sequence of transformations.
- All sequences of transformations can be consolidated down to
  - A scaling, followed by
  - A rotation, followed by
  - A translation.
- This is the most intuitive sequence.

# Sequences of Transformations

- A translation followed by a rotation can be rewritten as a rotation followed by a translation.
- A translation followed by a scaling can be rewritten as a scaling followed by a translation.
- A rotation followed by a scaling can be rewritten as a scaling followed by a rotation.

# Sequences of Transformations

- Furthermore, the product of two translations is again a translation.
- The product of two rotations is again a rotation.
- The product of two scalings is again a scaling.
- Thus, any sequence can be rewritten as one scaling, then one rotation, then one translation.

# Outline

# Other Rotations

- What if we want to rotate about a point $(x_0, y_0)$ that is not the origin?
- We can translate $(x_0, y_0)$ to the origin, rotate, then translate the origin back to $(x_0, y_0)$.

# Other Rotations

## Other Rotations

```
model = translate(-x_0, -y_0, 0.0f)*model;
model = rotate(angle, 0.0f, 0.0f, 1.0f)*model;
model = translate(x_0, y_0, 0.0f)*model;
```

## Other Rotations

```
model = translate(x_0, y_0, 0.0f)
          *rotate(angle, 0.0f, 0.0f, 1.0f)
            *translate(-x_0, -y_0, 0.0f)*model;
```

# Other Scalings

- What if we want to scale about a fixed point $(x_0, y_0)$ that is not the origin?
- We can translate $(x_0, y_0)$ to the origin, scale, then translate the origin back to $(x_0, y_0)$.

# Other Scalings

## Other Scalings

```
model = translate(-x_0, -y_0, 0.0f)*model;
model = scale(s_x, s_y, s_z)*model;
model = translate(x_0, y_0, 0.0f)*model;
```

## Other Scalings

```
model = translate(x_0, y_0, 0.0f)
          *scale(s_x, s_y, s_z)
            *translate(-x_0, -y_0, 0.0f)*model;
```

# Outline

# Assignment

## Assignment

- Assignment 6.
- Read pp. 207 - 210, Homogeneous Coordinates.
- Read pp. 210 - 217, Linear Transformations and Matrices.