

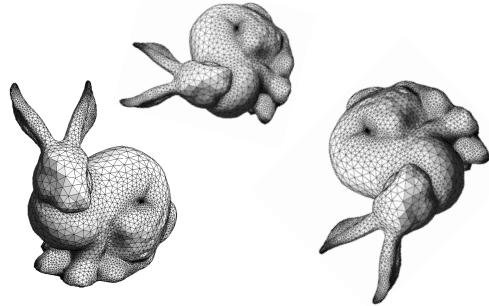
## Transformations

Jernej Barbic  
University of Southern California

Vector Spaces  
Euclidean Spaces  
Frames  
Homogeneous Coordinates  
Transformation Matrices  
[Angel, Ch. 3]

1

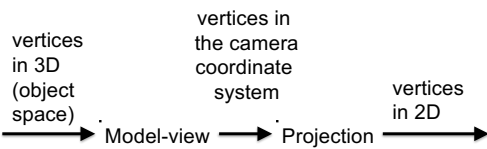
## OpenGL Transformations



2

## OpenGL Transformation Matrices

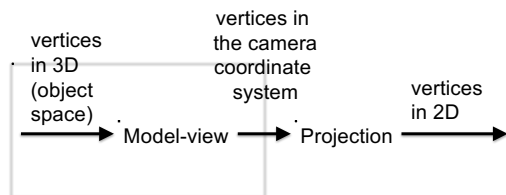
- Model-view matrix (4x4 matrix)
- Projection matrix (4x4 matrix)



3

## 4x4 Model-view Matrix (this lecture)

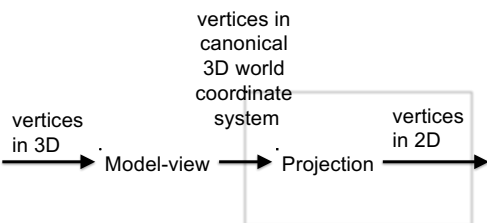
- Translate, rotate, scale objects
- Position the camera



4

## 4x4 Projection Matrix (next lecture)

- Project from 3D to 2D



5

## OpenGL Transformation Matrices



- Manipulated separately in OpenGL
- Core profile: set them directly
- Compatibility profile: must set matrix mode

```
glMatrixMode (GL_MODELVIEW);  
glMatrixMode (GL_PROJECTION);
```

6

## Setting the Modelview Matrix: Core Profile

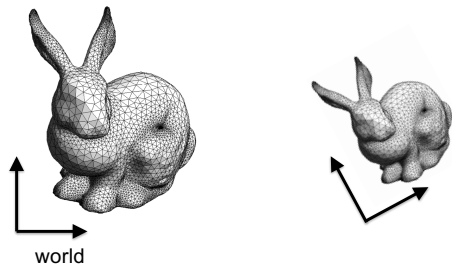
- Set identity:  
`openGLMatrix->SetMatrixMode(OpenGLMatrix::ModelView);`  
`openGLMatrix->LoadIdentity();`
- Use our openGLMatrix library functions:  
`openGLMatrix->Translate(dx, dy, dz);`  
`openGLMatrix->Rotate(angle, vx, vy, vz);`  
`openGLMatrix->Scale(sx, sy, sz);`
- Upload m to the GPU:  
`float m[16]; // column-major`  
`openGLMatrix->GetMatrix(m);`  
`GLboolean isRowMajor = GL_FALSE;`  
`glUniformMatrix4fv(h_modelViewMatrix, 1, isRowMajor, m);`  
*// note: h\_modelViewMatrix is a handle of the shader modelview matrix variable (will discuss in Shaders lecture)*

## Setting the Modelview Matrix: Compatibility Profile

- Load or post-multiply  
`glMatrixMode(GL_MODELVIEW);`  
`glLoadIdentity(); // very common usage`  
`float m[16] = { ... };`  
`glLoadMatrixf(m); // rare, advanced`  
`glMultMatrixf(m); // rare, advanced`
- Use library functions  
`glTranslatef(dx, dy, dz);`  
`glRotatef(angle, vx, vy, vz);`  
`glScalef(sx, sy, sz);`

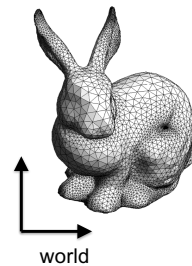
8

## Translated, rotated, scaled object



9

## The rendering coordinate system

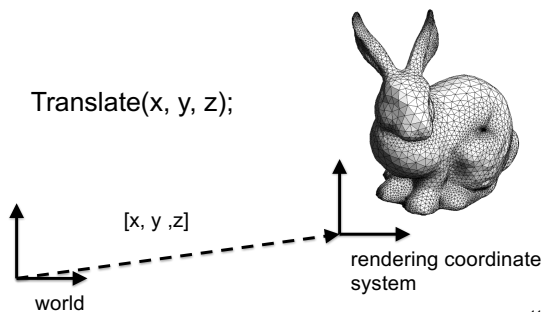


Initially (after `LoadIdentity()`) :  
 rendering coordinate system =  
 world coordinate system

10

## The rendering coordinate system

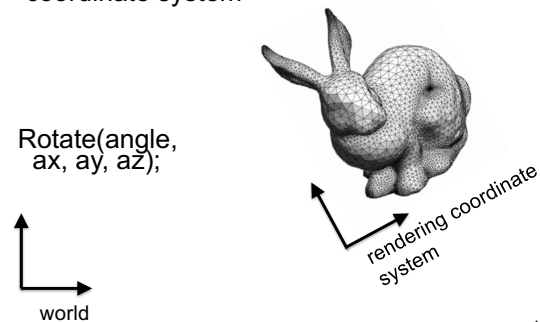
`Translate(x, y, z);`



11

## The rendering coordinate system

`Rotate(angle, ax, ay, az);`



12

### The rendering coordinate system

Scale(sx, sy, sz);

world

rendering coordinate system

13

### OpenGL pseudo-code

```

MatrixMode(ModelView);
LoadIdentity();
Translate(x, y, z);
Rotate(angle, ax, ay, az);
Scale(sx, sy, sz);
glUniformMatrix4fv(...);
renderBunny();

```

world

rendering coordinate system

14

### Rendering more objects

How to obtain this frame?

world

15

### Rendering more objects

Solution 1:  
Find Translate(...), Rotate(...), Scale(...)

How to obtain this frame?

world

16

### Rendering more objects

How to obtain this frame?

Solution 2:  
LoadIdentity();  
Find Translate(...), Rotate(...), Scale(...)

world

17

### 3D Math Review

18

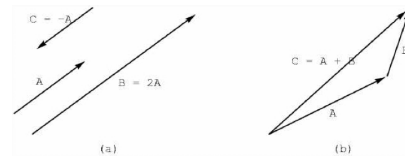
## Scalars

- Scalars  $\alpha, \beta, \gamma$  from a *scalar field*
- Operations  $\alpha + \beta, \alpha \cdot \beta, 0, 1, -\alpha, ()^{-1}$
- “Expected” laws apply
- Examples: rationals or reals with addition and multiplication

19

## Vectors

- Vectors  $u, v, w$  from a *vector space*
- Vector addition  $u + v$ , subtraction  $u - v$
- Zero vector  $\mathbf{0}$
- Scalar multiplication  $\alpha v$



20

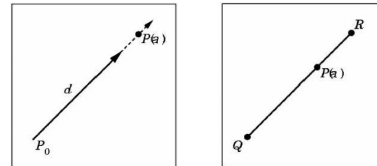
## Euclidean Space

- Vector space over real numbers
- Three-dimensional in computer graphics
- Dot product:  $\alpha = u \cdot v = u_1 v_1 + u_2 v_2 + u_3 v_3$
- $\mathbf{0} \cdot \mathbf{0} = 0$
- $u, v$  are *orthogonal* if  $u \cdot v = 0$
- $|v|^2 = v \cdot v$  defines  $|v|$ , the *length* of  $v$

21

## Lines and Line Segments

- Parametric form of line:  $P(\alpha) = P_0 + \alpha d$

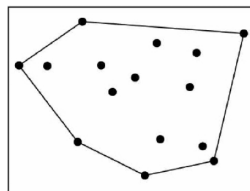


- Line segment between  $Q$  and  $R$ :  
 $P(\alpha) = (1-\alpha)Q + \alpha R$  for  $0 \leq \alpha \leq 1$

22

## Convex Hull

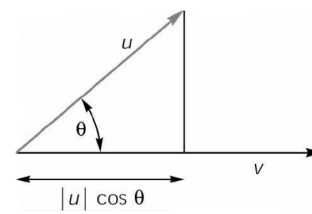
- Convex hull defined by  
 $P = \alpha_1 P_1 + \dots + \alpha_n P_n$   
for  $\alpha_1 + \dots + \alpha_n = 1$   
and  $0 \leq \alpha_i \leq 1, i = 1, \dots, n$



23

## Projection

- Dot product projects one vector onto another vector  
 $u \cdot v = u_1 v_1 + u_2 v_2 + u_3 v_3 = |u| |v| \cos(\theta)$   
 $\text{pr}_v u = (u \cdot v) v / |v|^2$



24

### Cross Product

$$\begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} \times \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} a_2 b_3 - a_3 b_2 \\ a_3 b_1 - a_1 b_3 \\ a_1 b_2 - a_2 b_1 \end{pmatrix}$$

- $|a \times b| = |a| |b| |\sin(\theta)|$
- Cross product is perpendicular to both  $a$  and  $b$
- Right-hand rule

Source: Wikipedia

### Plane

- Plane defined by point  $P_0$  and vectors  $u$  and  $v$
- $u$  and  $v$  should not be parallel
- Parametric form:  
 $T(\alpha, \beta) = P_0 + \alpha u + \beta v$   
 ( $\alpha$  and  $\beta$  are scalars)
- $n = u \times v / |u \times v|$  is the normal
- $n \cdot (P - P_0) = 0$  if and only if  $P$  lies in plane

26

### Coordinate Systems

- Let  $v_1, v_2, v_3$  be three linearly independent vectors in a 3-dimensional vector space
- Can write *any* vector  $w$  as  
 $w = \alpha_1 v_1 + \alpha_2 v_2 + \alpha_3 v_3$   
 for some scalars  $\alpha_1, \alpha_2, \alpha_3$

27

### Frames

- Frame = origin  $P_0$  + coordinate system
- Any point  $P = P_0 + \alpha_1 v_1 + \alpha_2 v_2 + \alpha_3 v_3$

28

### In Practice, Frames are Often Orthogonal

29

### Change of Coordinate System

- Bases  $\{u_1, u_2, u_3\}$  and  $\{v_1, v_2, v_3\}$
- Express basis vectors  $u_i$  in terms of  $v_j$

$$u_1 = \gamma_{11}v_1 + \gamma_{12}v_2 + \gamma_{13}v_3$$

$$u_2 = \gamma_{21}v_1 + \gamma_{22}v_2 + \gamma_{23}v_3$$

$$u_3 = \gamma_{31}v_1 + \gamma_{32}v_2 + \gamma_{33}v_3$$

- Represent in matrix form:

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} = M \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} \quad M = \begin{bmatrix} \gamma_{11} & \gamma_{12} & \gamma_{13} \\ \gamma_{21} & \gamma_{22} & \gamma_{23} \\ \gamma_{31} & \gamma_{32} & \gamma_{33} \end{bmatrix}$$

30

## Representing 3D transformations (and model-view matrices)

31

## Linear Transformations

- 3 x 3 matrices represent linear transformations  
 $\mathbf{a} = \mathbf{M}\mathbf{b}$
- Can represent rotation, scaling, and reflection
- Cannot represent translation

$$M = \begin{bmatrix} \gamma_{11} & \gamma_{12} & \gamma_{13} \\ \gamma_{21} & \gamma_{22} & \gamma_{23} \\ \gamma_{31} & \gamma_{32} & \gamma_{33} \end{bmatrix}$$

32

In order to represent rotations,  
scales AND translations:  
Homogeneous Coordinates

- Augment  $[\alpha_1 \ \alpha_2 \ \alpha_3]^T$  by adding a fourth component (1):

$$\mathbf{p} = [\alpha_1 \ \alpha_2 \ \alpha_3 \ 1]^T$$

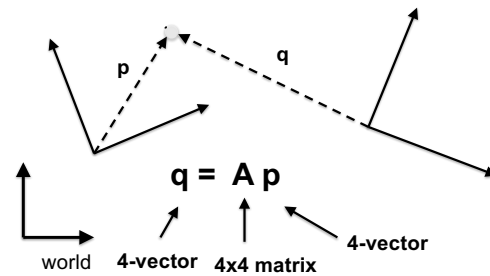
- Homogeneous property:

$$\mathbf{p} = [\alpha_1 \ \alpha_2 \ \alpha_3 \ 1]^T = [\beta\alpha_1 \ \beta\alpha_2 \ \beta\alpha_3 \ \beta]^T,$$

for any scalar  $\beta \neq 0$

33

Homogeneous coordinates  
are transformed by 4x4 matrices



34

## Affine Transformations (4x4 matrices)

- Translation
- Rotation
- Scaling
- Any composition of the above
- Later: projective (perspective) transformations
  - Also expressible as 4 x 4 matrices!

35

## Translation

- $\mathbf{q} = \mathbf{p} + \mathbf{d}$  where  $\mathbf{d} = [\alpha_x \ \alpha_y \ \alpha_z \ 0]^T$
- $\mathbf{p} = [x \ y \ z \ 1]^T$
- $\mathbf{q} = [x' \ y' \ z' \ 1]^T$
- Express in matrix form  $\mathbf{q} = \mathbf{T}\mathbf{p}$  and solve for  $\mathbf{T}$

$$T = \begin{bmatrix} 1 & 0 & 0 & \alpha_x \\ 0 & 1 & 0 & \alpha_y \\ 0 & 0 & 1 & \alpha_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

36

## Scaling

- $x' = \beta_x x$
- $y' = \beta_y y$
- $z' = \beta_z z$
- Express as  $\mathbf{q} = \mathbf{S} \mathbf{p}$  and solve for  $\mathbf{S}$

$$\mathbf{S} = \begin{bmatrix} \beta_x & 0 & 0 & 0 \\ 0 & \beta_y & 0 & 0 \\ 0 & 0 & \beta_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

37

## Rotation in 2 Dimensions

- Rotation by  $\theta$  about the origin
- $x' = x \cos \theta - y \sin \theta$
- $y' = x \sin \theta + y \cos \theta$
- Express in matrix form:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

- Note that the determinant is 1

38

## Rotation in 3 Dimensions

- Orthogonal matrices:

$$\mathbf{R}\mathbf{R}^T = \mathbf{R}^T\mathbf{R} = \mathbf{I}$$
$$\det(\mathbf{R}) = 1$$

- Affine transformation:

$$\mathbf{A} = \begin{bmatrix} R_{11} & R_{12} & R_{13} & 0 \\ R_{21} & R_{22} & R_{23} & 0 \\ R_{31} & R_{32} & R_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

39

## Affine Matrices are Composed by Matrix Multiplication

- $\mathbf{A} = \mathbf{A}_1 \mathbf{A}_2 \mathbf{A}_3$
- Applied from right to left
- $\mathbf{A} \mathbf{p} = (\mathbf{A}_1 \mathbf{A}_2 \mathbf{A}_3) \mathbf{p} = \mathbf{A}_1 (\mathbf{A}_2 (\mathbf{A}_3 \mathbf{p}))$
- Compatibility mode:  
When calling `glTranslatef`, `glRotatef`, or `glScalef`, OpenGL forms the corresponding 4x4 matrix, and multiplies the current modelview matrix with it.

40

## Summary

- OpenGL Transformation Matrices
- Vector Spaces
- Frames
- Homogeneous Coordinates
- Transformation Matrices

41