UiO : University of Oslo

INF3320: Computer Graphics and Discrete Geometry

Transforms (part I)

Christopher Dyken and Martin Reimers
Corrections and additions by André R. Brodtkorb

September 3, 2014

Transforms (part I)

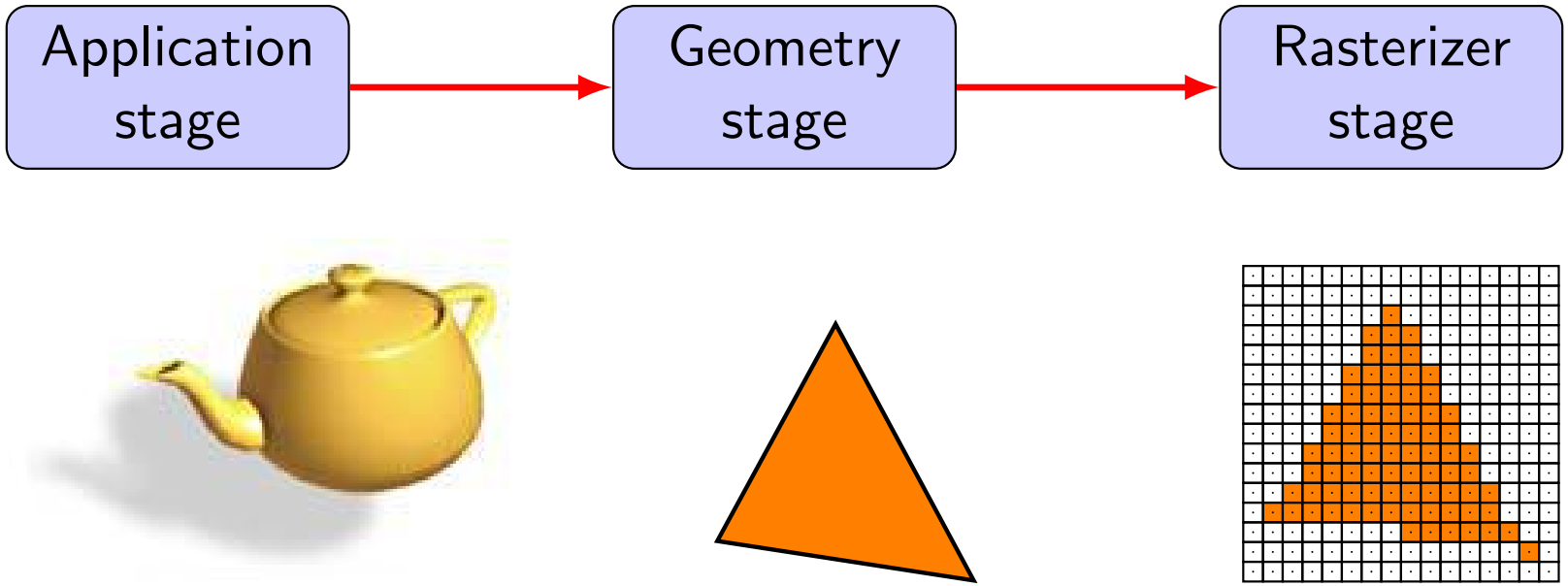
Real Time Rendering book:

- Some linear algebra (Appendix A)
- Basic transforms (Chapter 4.1)

The Red Book:

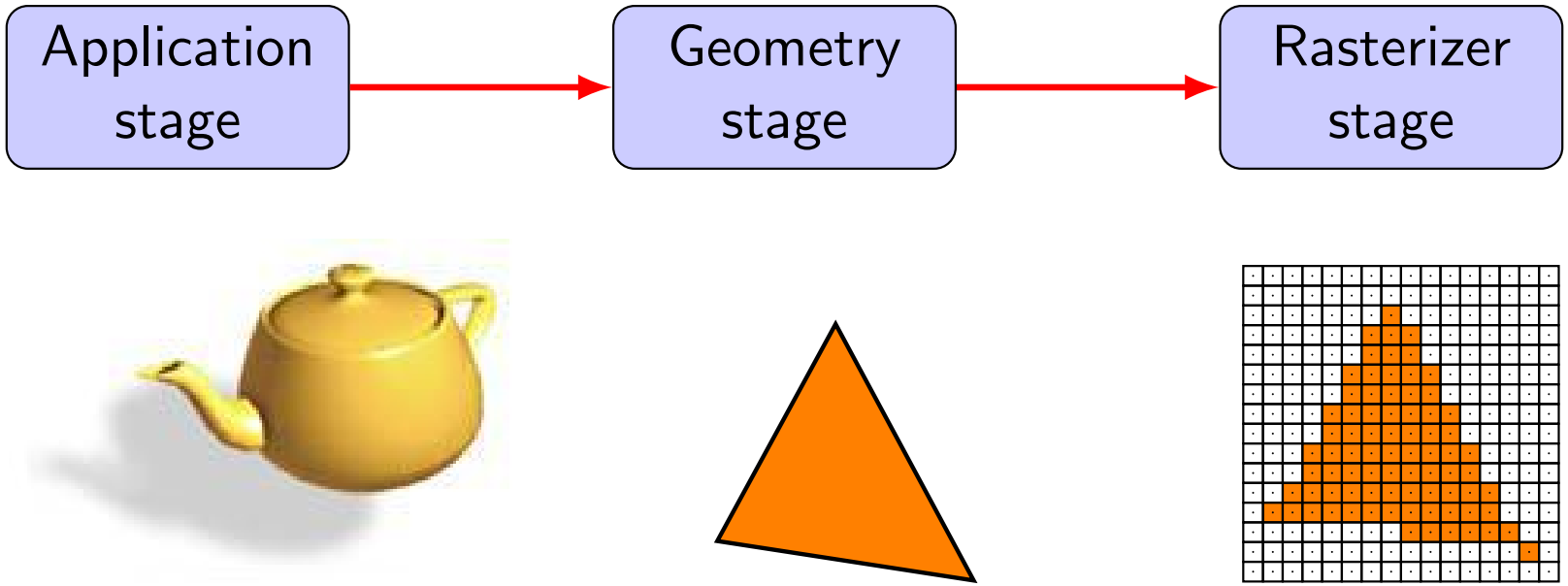
- Viewing (Chapter 3)
- Homogeneous Coordinates ... (Appendix F)

Recap and preview



Last lecture: drawing vertex based geometry in OpenGL

Recap and preview



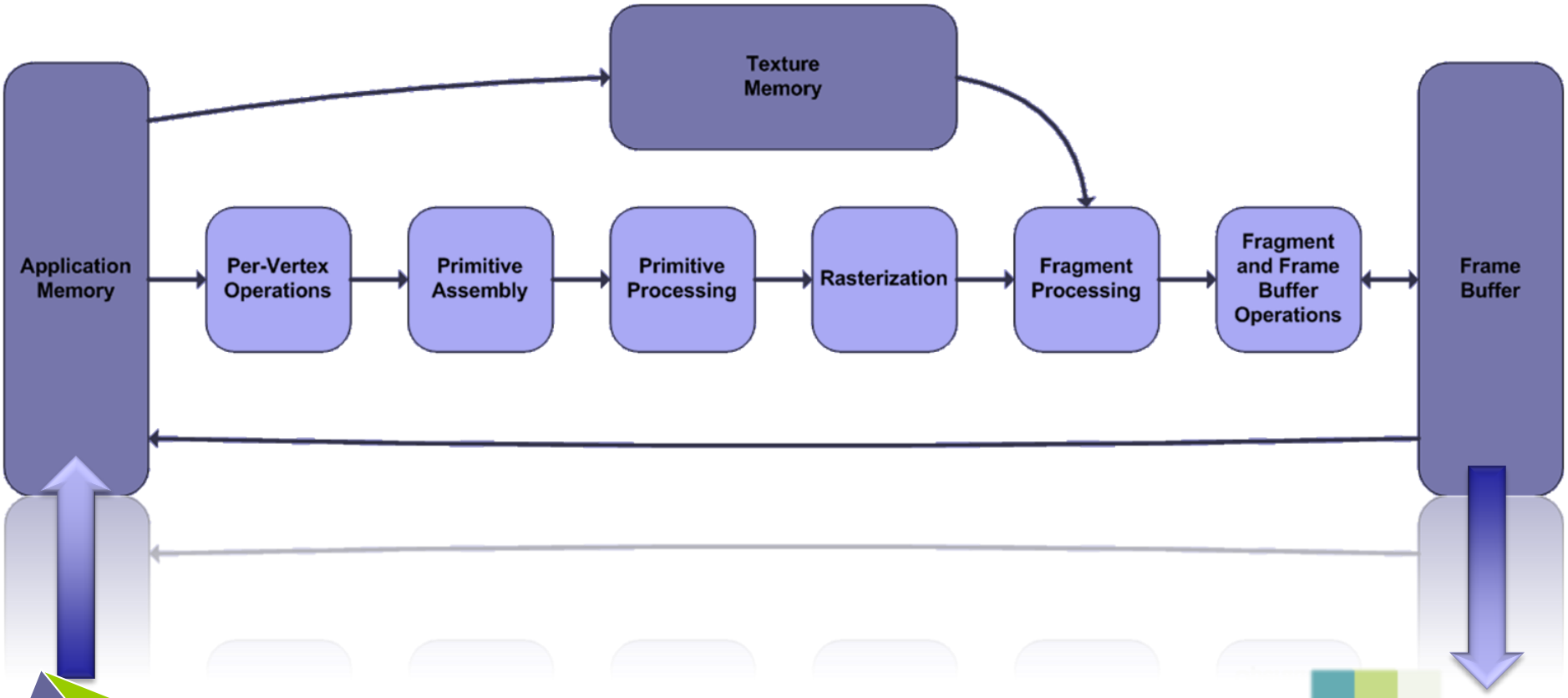
Last lecture: drawing vertex based geometry in OpenGL

Today: Vertex processing, transformations

In : Geometry described by vertices

Out : Transformed geometry (vertices)

The OpenGL Pipeline

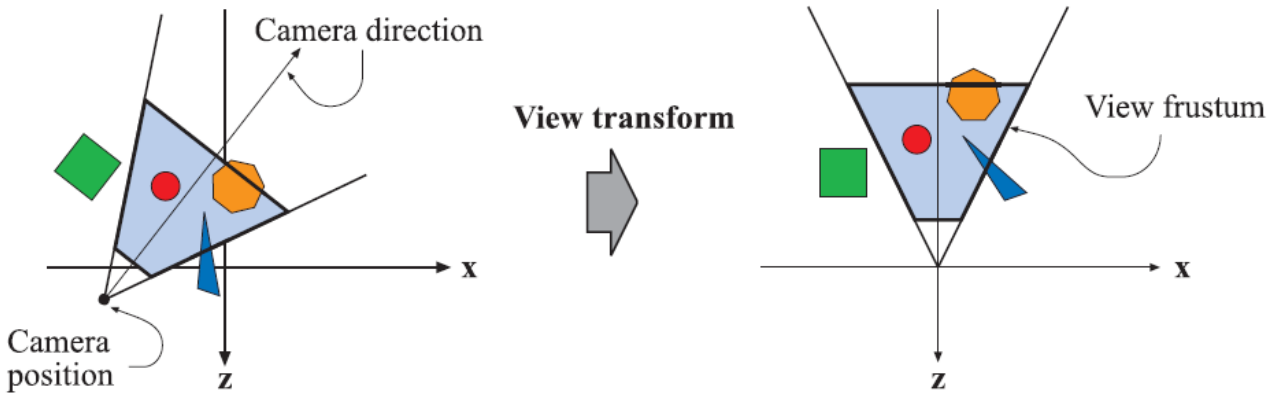


Linear Coordinate Systems and Transforms

Motivation: place geometry in eye-space

We have a "world" coordinate-system with objects, and a "camera"

Need to describe the objects in camera coordinates (eye space)



This is pure linear algebra!

Linear spaces and bases

A *linear vector space* is a set of elements (vectors), closed under addition and scalar multiplication

A basis for a linear space is a set of *linearly independent* vectors that *span* the space: *any* vector in the space can be expressed as a unique linear combination of the basis vectors.

Linearly independent

The vectors $\mathbf{b}_1, ..., \mathbf{b}_n$ are linearly independent if

$$\sum a_i \mathbf{b}_i = \mathbf{0} \implies a_i = 0 \quad \forall i$$

Basis matrix

Cardinal basis for $\mathbb{R}^3$

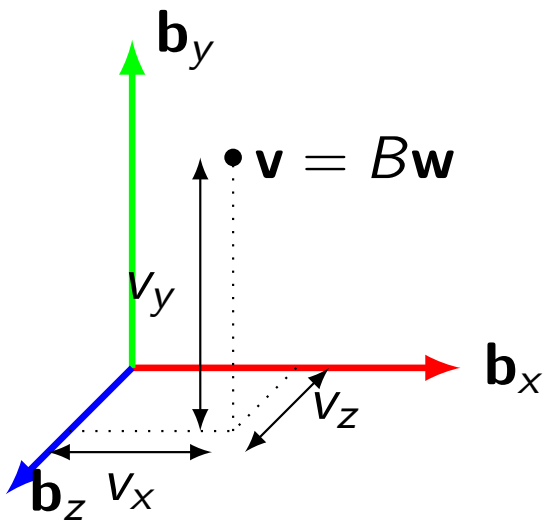
the set of the three unit vectors along the x, y, and z axes.

Basis matrix: The matrix with the basis vectors as columns

Basis matrix for the cardinal basis

$$B = \begin{bmatrix} \mathbf{b}_x & \mathbf{b}_y & \mathbf{b}_z \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

Expressing a vector in a basis



Vectors are expressed as

linear combinations of basis

$$\mathbf{v} = \begin{bmatrix} \mathbf{b}_x & \mathbf{b}_y & \mathbf{b}_z \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix}$$

$$= v_x\mathbf{b}_x + v_y\mathbf{b}_y + v_z\mathbf{b}_z$$

$$= B\mathbf{w}$$

Coordinates

$\mathbf{w} = (v_x, v_y, v_z)$ are the *coordinates* of $\mathbf{v}$ wrt $(\mathbf{b}_i)$

To find coordinates: solve $B\mathbf{w} = \mathbf{v}$, i.e. $\mathbf{w} = B^{-1}\mathbf{v}$,

Orthonormal bases

A basis $(\mathbf{b}_i)$ is *orthonormal* if

$$\mathbf{b}_i\mathbf{b}_j = 0, \qquad i \neq j, \qquad ||\mathbf{b}_i|| = 1$$

If $(\mathbf{b}_i)$ is orthonormal, B is a *orthogonal matrix*

$$B^{-1} = B^T$$

This is the usual case in computer graphics!

Finding coordinates is simple:

$$B\mathbf{w} = \mathbf{v} \implies \mathbf{w} = B^T\mathbf{v}$$

Change of basis

Given a vector in one basis, we can use a *change-of-basis-matrix* to express it in another basis.

Change of basis: Two representations $\mathbf{v} = B_1\mathbf{v}_1 = B_2\mathbf{v}_2$

$$\mathbf{v}_1 = \underbrace{B_1^{-1}B_2}_{M}\mathbf{v}_2 = M\mathbf{v}_2$$

M transforms coordinates wrt B_2 to coordinates wrt. B_1.

M^{-1} transforms coordinates wrt B_1 to coordinates wrt. B_2.

If B_1 is the cardinal basis ($B_1 = I$),
the change-of-basis-matrix simplifies to $M = B_2$.

Linear transformations

Linear transformation:

$$f(\mathbf{x}+\mathbf{y}) = f(\mathbf{x}) + f(\mathbf{y}), \qquad f(\alpha\mathbf{y}) = \alpha f(\mathbf{y})$$

Linear transforms $f : \mathbb{R}^n \to \mathbb{R}^n$:

- can be expressed as matrix-vector products: $f(\mathbf{x}) = A\mathbf{x}$
- can be concatenated: $A(B\mathbf{x}) = (AB)\mathbf{x}$
- does not commute: $AB\mathbf{x} \neq BA\mathbf{x}$
- can be decomposed into basic transforms:

$$\underbrace{\begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}}_{\text{Scaling}} \quad \underbrace{\begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix}}_{\text{Rotation}} \quad \underbrace{\begin{bmatrix} 1 & s \\ 0 & 1 \end{bmatrix}}_{\text{Shear}} \quad \underbrace{\begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}}_{\text{Projection}}$$

Note: $f(x) = ax + b$ is not a linear transform (unless $b = 0$)!

Affine transformations

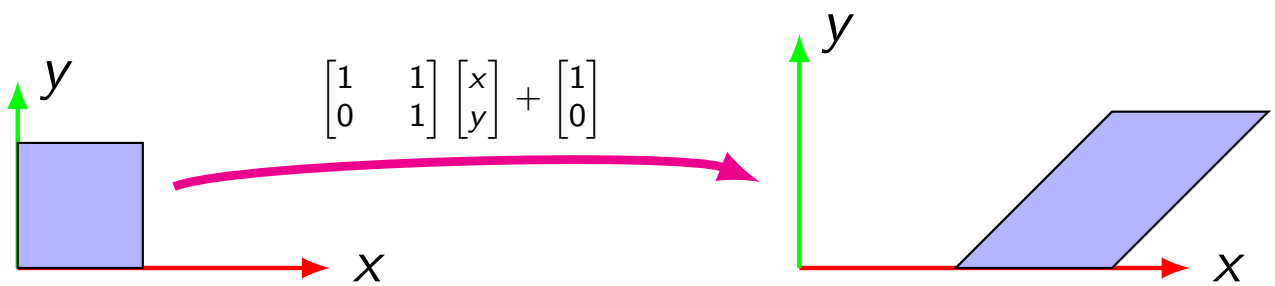
Affine transformation: $f : \mathbb{R}^n \to \mathbb{R}^n$

$$f(\alpha\mathbf{x} + \beta\mathbf{y}) = \alpha f(\mathbf{x}) + \beta f(\mathbf{y}), \quad \alpha + \beta = 1$$

In $\mathbb{R}^n$: The combination of a linear transform and a translation

$$f(\mathbf{x}) = A\mathbf{x} + \mathbf{t}$$

Preserves linearity: lines $\mapsto$ lines, triangles $\mapsto$ triangles etc.



Therefore: *linear geometry can be transformed by its vertices*
$\Longrightarrow$ Affine mappings are extremely important in gfx!

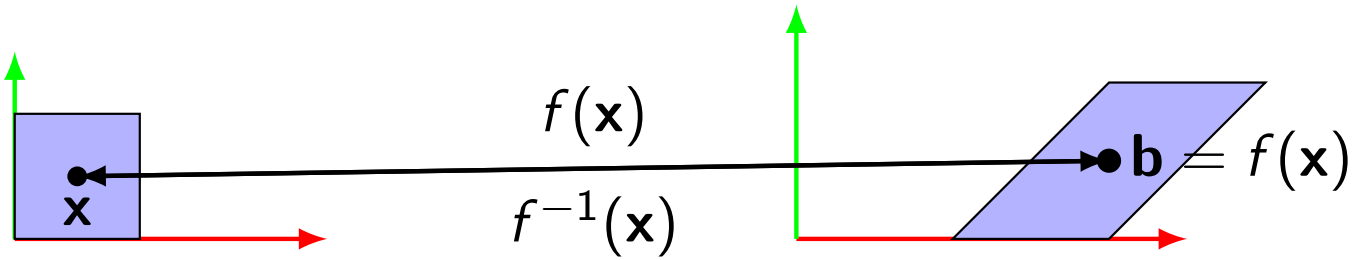
Inverse affine transformations

The inverse of an invertible affine transformation is an affine transformation: if B is nonsingular and

$$\mathbf{b} = f(\mathbf{x}) = B\mathbf{x} + \mathbf{t}$$

Then

$$\mathbf{x} = f^{-1}(\mathbf{b}) = B^{-1}(\mathbf{b} - \mathbf{t}) = B^{-1}\mathbf{b} - B^{-1}\mathbf{t}$$

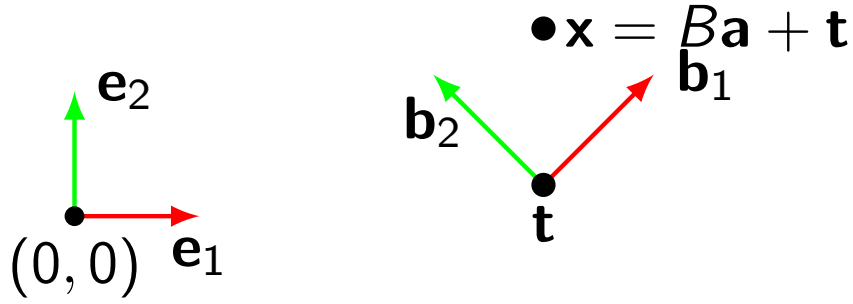


Change of coordinate system/frame

Coordinate system (frame): an "origin" and a basis for $\mathbb{R}^n$

$\mathbf{x} \in \mathbb{R}^n$ is expressed in coordinates wrt a basis, relative to an origin

$$\mathbf{x} = x\mathbf{e}_1 + y\mathbf{e}_2 = \mathbf{t} + a_1\mathbf{b}_1 + a_2\mathbf{b}_2 = B\mathbf{a} + \mathbf{t}$$



Coordinates wrt the other coordinate system: affine transformation

$$\mathbf{a} = B^{-1}(\mathbf{x} - \mathbf{t}) = B^{-1}\mathbf{x} - B^{-1}\mathbf{t}$$

Important example in gfx: model to camera/eye space.

Homogeneous Coordinates

Using homogeneous coordinates for affine transforms

The homogeneous representation of points in $\mathbb{R}^3$

have one dimension more then the Euclidean representation,

$$\mathbf{p} = \begin{bmatrix} x & y & z & w \end{bmatrix}^T$$

(we have added w, the fourth coordinate)

Why?

- Linear, affine, & projective transforms
  can be represented as matrix multiplications

$\Longrightarrow$ simplifies transform stage of computer graphics considerably!

Homogeneous points

Homogeneous points from Euclidean points

From the Euclidean point $\begin{bmatrix} x & y & z \end{bmatrix}^T$, we add $w = 1$,

$$\mathbf{p} = \begin{bmatrix} x & y & z & 1 \end{bmatrix}^T$$

A Euclidean point is in fact *a line* in homogeneous coordinates

$$\mathbf{p} = \begin{bmatrix} \alpha x & \alpha y & \alpha z & \alpha \end{bmatrix}^T$$

is exactly the same Euclidean point for any nonzero real α.

Euclidean representation from homogeneous representation

the Euclidean point is found by scaling s.t. $w = 1$,

$$\mathbf{p} = \begin{bmatrix} x/w & y/w & z/w \end{bmatrix}^T$$

What happens when w goes to zero?

Where does $\lim_{w \to 0} \begin{bmatrix} 1 & 2 & 0 & w \end{bmatrix}^T$ end up?

homogeneous representation	Euclidean representation
$\begin{bmatrix} 1 & 2 & 0 & 1 \end{bmatrix}^T$	$\begin{bmatrix} 1 & 2 & 0 \end{bmatrix}^T$
$\begin{bmatrix} 1 & 2 & 0 & 0.1 \end{bmatrix}^T$	$\begin{bmatrix} 10 & 20 & 0 \end{bmatrix}^T$
$\begin{bmatrix} 1 & 2 & 0 & 0.01 \end{bmatrix}^T$	$\begin{bmatrix} 100 & 200 & 0 \end{bmatrix}^T$
$\begin{bmatrix} 1 & 2 & 0 & 0.001 \end{bmatrix}^T$	$\begin{bmatrix} 1000 & 2000 & 0 \end{bmatrix}^T$
$\vdots$	$\vdots$

$\Longrightarrow$ rapidly moves towards infinity in the direction of $\begin{bmatrix} 1 & 2 & 0 \end{bmatrix}^T$

Think of (x, y, z, w) as a point and $(x, y, z, 0)$ as a vector

Affine Transformation Matrices

Affine transformations

Linear transforms (3×3-matrices) cannot handle translations

Affine transforms: a linear transform plus a translation

$$\mathbf{p}' = A\mathbf{p} + \mathbf{t}$$

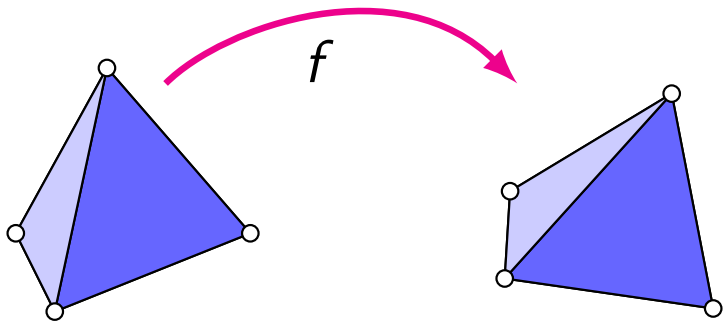
Affine transforms preserves line segments

If $f : \mathbb{R}^n \to \mathbb{R}^n$ is an affine transform, then

$$f\big((1-\alpha)\mathbf{p}_0 + \alpha\mathbf{p}_1\big) = (1-\alpha)f(\mathbf{p}_0) + \alpha f(\mathbf{p}_1), \quad \alpha \in [0,1].$$

$\Longrightarrow$ maps triangles to triangles, tetrahedra to tetrahedra, etc.

$\Longrightarrow$ transform OpenGL primitives by transforming the vertices only!



Homogeneous transformation matrices

Affine transforms represented as homogeneous matrices

$$M = \begin{bmatrix} a_{11} & a_{12} & a_{13} & t_1 \\ a_{21} & a_{22} & a_{23} & t_2 \\ a_{31} & a_{32} & a_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where

- 3×3 matrix (red, green, and blue) encodes linear transform.
- vector (magenta elements) encodes the translation.

Carrying out the matrix multiplication, we see that

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & t_1 \\ a_{21} & a_{22} & a_{23} & t_2 \\ a_{31} & a_{32} & a_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} xa_{11} + ya_{12} + za_{13} + t_1 \\ xa_{21} + ya_{22} + za_{23} + t_2 \\ xa_{31} + ya_{32} + za_{33} + t_3 \\ 1 \end{bmatrix}$$

is equivalent to $\mathbf{p}' = A\mathbf{p} + \mathbf{t}$!

Concatenation of transforms

Assume $\mathbf{p}'$ is $\mathbf{p}$ transformed by M_1 and $\mathbf{p}''$ is $\mathbf{p}'$ transformed by M_2:

$$\mathbf{p}' = M_1\mathbf{p} \qquad \text{and} \qquad \mathbf{p}'' = M_2\mathbf{p}'.$$

We can then concatenate the transforms:

$$\mathbf{p}'' = M_2\mathbf{p}' = M_2(M_1\mathbf{p}) = \underbrace{M_2M_1}_{Q}\mathbf{p} = Q\mathbf{p}$$

If M_1 and M_2 are affine transforms then Q is an affine transform.

Any number of transforms $M_1, \ldots, M_n$ can be concatenated.

M_1 applied to $\mathbf{p}$, then M_2 applied to the result, and so on

$$M\mathbf{p} = \left(M_nM_{n-1}\cdots M_2M_1\right)\mathbf{p}$$

Block notation

Write M on block form

$$M = \begin{bmatrix} a_{11} & a_{12} & a_{13} & t_1 \\ a_{21} & a_{22} & a_{23} & t_2 \\ a_{31} & a_{32} & a_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} A & \mathbf{t} \\ \mathbf{0}^t & 1 \end{bmatrix}$$

where

- A is a 3×3 matrix, e.g. rotation and scaling
- $\mathbf{t}$ is a vector representing translation.

Affine transforms applied to points reads

$$\begin{bmatrix} A & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix} = \begin{bmatrix} A\mathbf{x} + \mathbf{t} \\ 1 \end{bmatrix}$$

Affine transforms applied to points reads

$$\begin{bmatrix} A & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix} = \begin{bmatrix} A\mathbf{x} + \mathbf{t} \\ 1 \end{bmatrix}$$

We can then treat affine transforms as block-matrices

$$\begin{aligned} M_2 M_1 &= \begin{bmatrix} A_2 & \mathbf{t}_2 \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} A_1 & \mathbf{t}_1 \\ \mathbf{0}^T & 1 \end{bmatrix} \\ &= \begin{bmatrix} \overbrace{A_2 A_1}^{3\times 3} + \overbrace{\mathbf{t}_2 \mathbf{0}^T}^{3\times 3} & \overbrace{A_2 \mathbf{t}_1}^{3\times 1} + \overbrace{\mathbf{t}_2 \cdot 1}^{3\times 1} \\ \underbrace{\mathbf{0}^T A_1}_{1\times 3} + \underbrace{\mathbf{1} \cdot \mathbf{0}^T}_{1\times 3} & \underbrace{\mathbf{0}^t \mathbf{t}_1}_{1\times 1} + \underbrace{\mathbf{1} \cdot \mathbf{1}}_{1\times 1} \end{bmatrix} \\ &= \begin{bmatrix} A_2 A_1 & A_2 \mathbf{t}_1 + \mathbf{t}_2 \\ \mathbf{0}^T & \mathbf{1} \end{bmatrix} \end{aligned}$$

Inverse transforms

Inverse transforms in homogeneous coordinates:

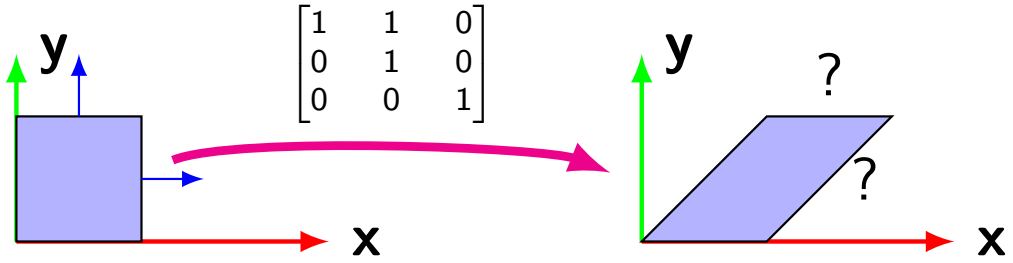
$$M^{-1} = \begin{bmatrix} A & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix}^{-1} = \begin{bmatrix} A^{-1} & -A^{-1}\mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix}$$

Check:

$$M^{-1}M = \begin{bmatrix} A & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix}^{-1} \begin{bmatrix} A & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} = \begin{bmatrix} A^{-1}A & A^{-1}\mathbf{t} - A^{-1}\mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix}$$
$$= I$$

Normal transformation

Transform the unit square with the following transformation...

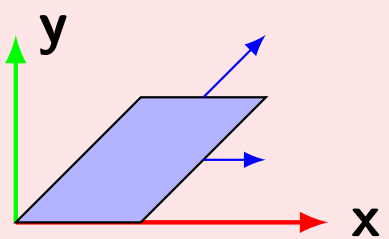


...what should we do with the normal vectors?

WRONG: using homogeneous normal vectors

$$\begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

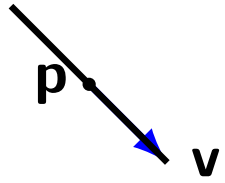
...the transformed normals are
not normal to the surface anymore!



Lines and planes

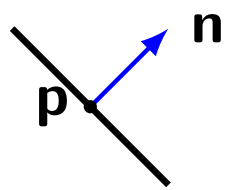
A *line* in $\mathbb{R}^2$ can be expressed explicitly by a point and a vector

$$\mathbf{L}(t) = t\mathbf{v} + \mathbf{p}$$



or implicitly by a point and a normal vector, as

$$f(\mathbf{x}) = (\mathbf{x} - \mathbf{p}) \cdot \mathbf{n} = 0$$



Similarly, a *plane* can be represented explicitly (parametrically)

$$\mathbf{P}(u, v) = u\mathbf{e}_1 + v\mathbf{e}_2 + \mathbf{p}$$

with a point and two tangent vectors, or implicitly as

$$f(\mathbf{x}) = (\mathbf{x} - \mathbf{p}) \cdot \mathbf{n} = ax + by + cz + d = 0$$

A *half-space*: $\mathbf{x}$ s.t. $f(\mathbf{x}) < 0$ or $f(\mathbf{x}) > 0$

Homogeneous planes

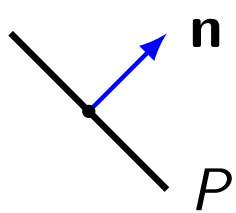
We use *row* vectors $\mathbf{n} = \begin{bmatrix} a & b & c & d \end{bmatrix}$ to represent planes.

- $\mathbf{v} = [a, b, c]$ is the normal vector of the plane
- d encodes the distance from the plane to the origin
- $\alpha\mathbf{n}$ and $\mathbf{n}$ represents the same plane (for $\alpha \neq 0$)

Implicit equation of a homogenous plane

The hom. point $\mathbf{p}$ is *on* the hom. plane $\mathbf{n}$ if and only if

$$\mathbf{n} \cdot \mathbf{p} = a\ x + b\ y + c\ z + d\ w = 0$$



$[a, b, c, 0]$ **encodes a plane through 0, with normal** $[a, b, c]$

The plane $\mathbf{n} = \begin{bmatrix} 0 & 0 & 0 & d \end{bmatrix}$ represents a plane at infinity

Transforming normal vectors

How to transform a homogeneous plane

Let $\mathbf{p}$ be a homogeneous point on a homogeneous plane $\mathbf{n}$

Then, for a non-singular homogeneous transformation matrix M...

$$\mathbf{n}\,\mathbf{p} = \mathbf{n}\,I\,\mathbf{p} = \mathbf{n}\,\overbrace{M^{-1}\,M}^{I}\,\mathbf{p} = 0$$

...the point $\mathbf{p}' = M\mathbf{p}$ lies on the plane $\mathbf{n}' = \mathbf{n}M^{-1}$!

We transform a plane by right-multiplying $\mathbf{n}$ with M^{-1}

Transforming normal vectors

Representing normal vectors as homogeneous planes through the origin (d=0), we have a rule to transform them ...

...transformed normal vectors are normal to transformed planes

We can write $\mathbf{n}' = \mathbf{n}M^{-1}$ on the form

$$\begin{bmatrix} \mathbf{v}^T & 0 \end{bmatrix} \begin{bmatrix} A^{-1} & \mathbf{w} \\ \mathbf{0}^T & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{v}^T A^{-1} & \mathbf{v}^T \mathbf{w} \end{bmatrix}$$

The transformed Euclidean normal vector is thus $\mathbf{v}' = (A^{-1})^T \mathbf{v}$

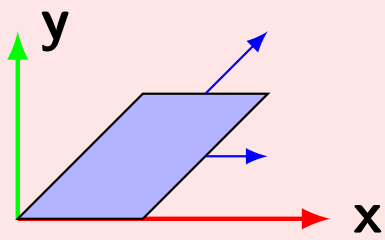
Note: If A is orthogonal, $A^{-1} = A^T$ and $\mathbf{v}' = A\mathbf{v}$

Back to our example

WRONG: using homogeneous normal vectors

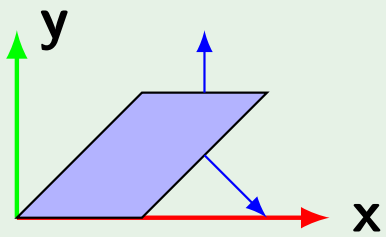
$$\begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

. . . the transformed normals are *not normal* to the surface anymore!



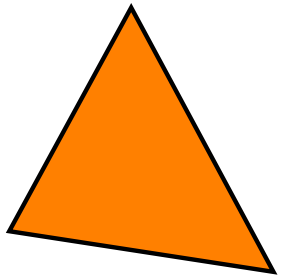
Right: Using M^{-1} to transform normals

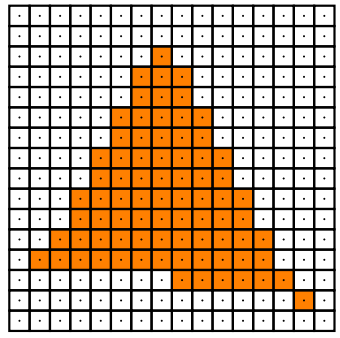
$$\begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix}$$



. . . normal vectors are *normal* to the planes!

Affine Transformations in OpenGL

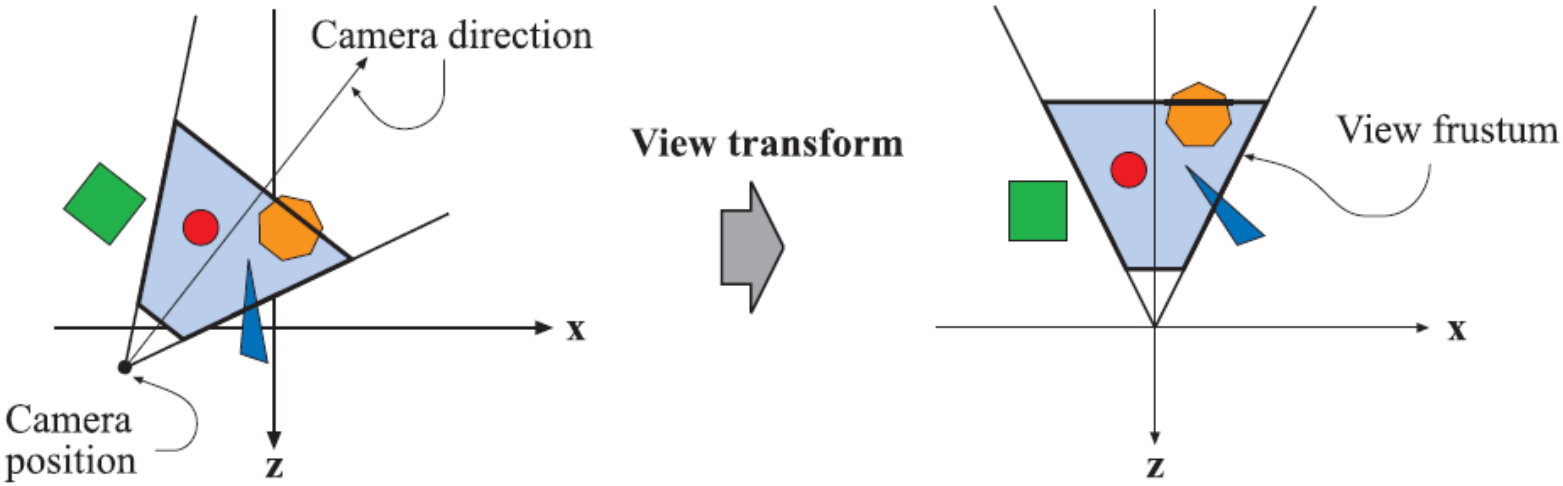




2. *The geometry stage*: Vertex processing
   - Hardware and software ("vertex shaders")
   - Further divided into five substages:

2.1 *Model and view transform*

- The *model transform* defines how *objects* are positioned w.r.t the global coordinate system.
- The *view transform* defines how the *camera* is positioned w.r.t the global coordinate system.



- Combined, the *modelview transform* transform the input geometry to the local coordinate system of the camera, the *eye/camera space*

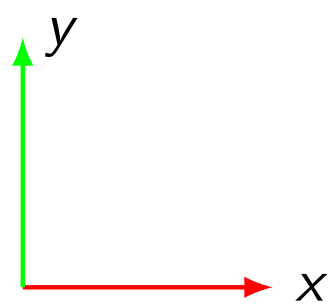
OpenGL transformations

OpenGL needs affine transforms, projections etc.

The first transformation is from model space to camera space: affine transformation called *modelview transformation*

Transformations are done using homogeneous coordinates

The default camera coordinate system looks like



with the positive z axis towards you

OpenGL transformation matrices

OpenGL (<3.2) has 2 vertex transformation matrices (state vars.):

- `GL_MODELVIEW`, mapping from model to eye-space
- `GL_PROJECTION`, mapping from eye- to clip-space

OpenGL matrices - organized in two stacks

Select which matrix stack to work on using `glMatrixMode`

Push and pop matrices using `glPushMatrix` and `glPopMatrix`.
$\Longrightarrow$ Always the top of the stack that is used and modified.

Clear using `glLoadIdentity`

Set using `glLoadMatrix`

Right-multiply with an arbitrary matrix using `glMultMatrix`

Specify a sequence of transformations in OpenGL

The OpenGL code for $M\mathbf{p} = \left(M_n M_{n-1} \cdots M_2 M_1\right)\mathbf{p}$ is

`glMatrixMode(GL_MODELVIEW);`	switch to `MODELVIEW`
`glLoadIdentity();`	`MODELVIEW` matrix M is I
`glMultMatrix*(`M_n`)`	$M = M_n$
`glMultMatrix*(`M_{n-1}`)`	$M = M_n M_{n-1}$
$\vdots$	$\vdots$
`glMultMatrix*(`M_2`)`	$M = M_n M_{n-1} \cdots M_2$
`glMultMatrix*(`M_1`)`	$M = M_n M_{n-1} \cdots M_2 M_1$
`glVertex*(`$\mathbf{p}$`)`	$\mathbf{p}' = M_n M_{n-1} \cdots M_2 M_1 \mathbf{p}_1$

$\mathbf{p}'$ is then the $\mathbf{p}$ transformed to the camera coordinate system

Model and view transformations (Modelview transform)

The modelview transform maps geometry into camera-space

It is useful to divide the transform into two components

- Model transforms for drawing geometry
- View transforms for setting up camera

Both manipulate the GL_MODELVIEW matrix

Occur in opposite order in the code

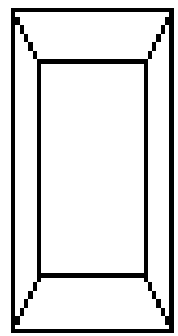
Modelview transform example

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity ();                  /* clear the matrix */

// viewing transformation
gluLookAt (0.0, 0.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);

// Equivalent transformation
// glTranslatef (0.0, 0.0, -5.0);

// modeling transformation
glScalef (1.0, 2.0, 1.0);
glutWireCube (1.0);
```



Model transforms

Position geometry, using translations, scaling, rotations
Two ways to *think* about model transform TR

We have a grand, fixed world frame

- matrix mults change position and orientation of your model.
- first rotate model with R and then translate model with T
- operations appear in *opposite order* as in the code.

We're moving a local "drawing" frame

- matrix ops move a "current" frame relative to the origin/camera.
- first translate the frame with T and then rotate it with R,
- operations appear in the *same order* as in the code.

(Move camera or move models - Pick your favorite)

View transform

Used to position the camera in world space

Main operations: translations and rotations

Very useful function: `gluLookAt(`**eye-pos,ref-point,up-vector)**

NB: Camera operations are inverse of model operations

- Moving the camera from the model is equivalent to moving the model from the camera (inverse translation)
- Camera rotations are equivalent to inverse model rotations

The last slide

Model and View transformations are important but difficult

Check out Nate Robins tutorials

`www.xmission.com/~nate/tutors.html`

Check also out the examples in Chapter 3 in the Red Book

Next lecture: Transformations part II