University of British Columbia

Geometric Transformations

- ***recap***
- ***composition of transformations***
- ***rendering pipeline overview***

News

University of British Columbia

- newsgroup is working
- labs start today, CICSR 011
  - *waitlisted: pick a lab, attend to hear TA talk*
  - *accounts: newsgroup answer on activate/reenable*
- readings
  - *Chap 1: graphics overview*
  - *Chap 2: graphics programming*
  - *Chap 4: transformations*

Transformations recap

University of British Columbia

Affine transformations

- linear transformation + translations
- can be expressed as a 3x3 matrix + 3 vector

$$\mathbf{x}' = \mathbf{M} \cdot \mathbf{x} + \mathbf{t}$$

Homogeneous coordinates

- Unified representation as 4-vector (in 3D) for
  - *Points*
  - *Vectors / directions*
- Affine transformations become 4x4 matrices
  - *Composing multiple affine transformations involves simply multiplying the matrices*

Homogeneous Coordinates

University of British Columbia

Homogeneous representation of points:

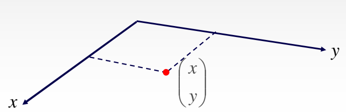
- Add an additional component $w=1$ to all *points*
- All multiples of this vector are considered to represent the same 3D point
- why bother? need unified representation

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \equiv \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \equiv \begin{bmatrix} x \cdot w \\ y \cdot w \\ z \cdot w \\ w \end{bmatrix} \forall w \neq 0$$

Geometrically In 2D

University of British Columbia

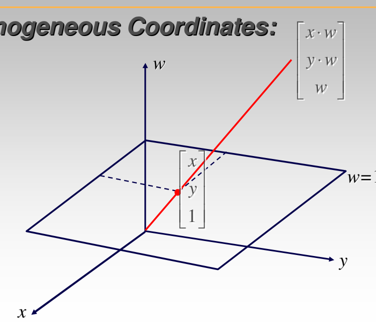
Cartesian Coordinates:



Geometrically In 2D

University of British Columbia

Homogeneous Coordinates:



Homogeneous Matrices

University of British Columbia

Affine Transformations

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} m_{1,1} & m_{1,2} & m_{1,3} & 0 \\ m_{2,1} & m_{2,2} & m_{2,3} & 0 \\ m_{3,1} & m_{3,2} & m_{3,3} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} m_{1,1} & m_{1,2} & m_{1,3} & 0 \\ m_{2,1} & m_{2,2} & m_{2,3} & 0 \\ m_{3,1} & m_{3,2} & m_{3,3} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & t_x \\ 0 & 0 & 0 & t_y \\ 0 & 0 & 0 & t_z \\ 0 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Homogeneous Matrices

University of British Columbia

Combining the two matrices into one:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} m_{1,1} & m_{1,2} & m_{1,3} & 0 \\ m_{2,1} & m_{2,2} & m_{2,3} & 0 \\ m_{3,1} & m_{3,2} & m_{3,3} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & t_x \\ 0 & 0 & 0 & t_y \\ 0 & 0 & 0 & t_z \\ 0 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} m_{1,1} & m_{1,2} & m_{1,3} & t_x \\ m_{2,1} & m_{2,2} & m_{2,3} & t_y \\ m_{3,1} & m_{3,2} & m_{3,3} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Homogeneous Matrices

University of British Columbia

Note:

- Multiplication of the matrix with a constant does not change the transformation!

$$\tilde{T}\left(\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}\right) = \begin{bmatrix} m_{1,1}\cdot k & m_{1,2}\cdot k & m_{1,3}\cdot k & t_x\cdot k \\ m_{2,1}\cdot k & m_{2,2}\cdot k & m_{2,3}\cdot k & t_y\cdot k \\ m_{3,1}\cdot k & m_{3,2}\cdot k & m_{3,3}\cdot k & t_z\cdot k \\ 0 & 0 & 0 & k \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x'\cdot k \\ y'\cdot k \\ z'\cdot k \\ k \end{bmatrix}$$

$$\equiv \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = T\left(\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}\right)$$

normalize: divide by w

Homogeneous Vectors

University of British Columbia

point: w >= 1

- What about vectors (directions)?
- What is the affine transformation of a vector?
  - *Rotation*
  - *Scaling*
  - *Translation*

Vectors are invariant under translation!

Homogeneous Vectors

University of British Columbia

Representing vectors/directions in homogeneous coordinates

- Need representation that is only affected by linear transformations, but not by translations
- Answer: $w=0$

$$T\left(\begin{bmatrix} x \\ y \\ z \\ 0 \end{bmatrix}\right) = \begin{bmatrix} m_{1,1} & m_{1,2} & m_{1,3} & t_x \\ m_{2,1} & m_{2,2} & m_{2,3} & t_y \\ m_{3,1} & m_{3,2} & m_{3,3} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 0 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \\ 0 \end{bmatrix}$$

Transformations

University of British Columbia

translate(a,b,c)

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & & & a \\ & 1 & & b \\ & & 1 & c \\ & & & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

scale(a,b,c)

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} a & & & \\ & b & & \\ & & c & \\ & & & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$\mathrm{Rotate}(x,\theta)$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & & & \\ & \cos\theta & -\sin\theta & \\ & \sin\theta & \cos\theta & \\ & & & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$\mathrm{Rotate}\ (y,\theta)$

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

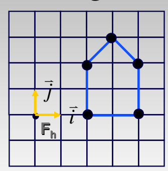
$\mathrm{Rotate}\ (z,\theta)$

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

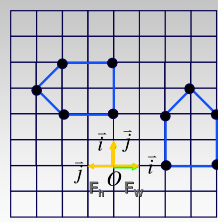
Transformations

University of British Columbia

Arriving at a transformation...



object defined in local coords

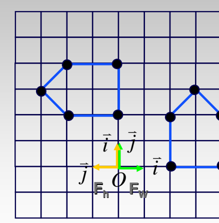


place object in world

Transformations

University of British Columbia

Arriving at a transformation...



$Rotate(z,90)$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & & \\ \sin\theta & \cos\theta & & \\ & & 1 & \\ & & & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

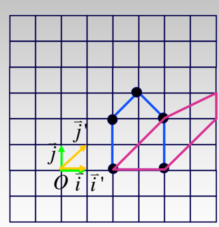
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix}_{\mathbf{w}} = \begin{bmatrix} 0 & -1 & & \\ 1 & 0 & & \\ & & 1 & \\ & & & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}_{\mathbf{h}}$$

$$P = O_h + x_h \vec{i}_h + y_h \vec{j}_h$$

Transformations

University of British Columbia

How about the following?



$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & & \\ 0 & 1 & & \\ & & 1 & \\ & & & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

shear

Composing Transformations

University of British Columbia

translation

$$T1 = T(dx_1, dy_1) = \begin{bmatrix} 1 & & dx_1 & a \\ & 1 & dy_1 & b \\ & & 1 & c \\ & & & 1 \end{bmatrix} \qquad T2 = T(dx_2, dy_2) = \begin{bmatrix} 1 & & dx_2 & a \\ & 1 & dy_2 & b \\ & & 1 & c \\ & & & 1 \end{bmatrix}$$

$$P'' = T2 \bullet P' = T2 \bullet [T1 \bullet P] = [T2 \bullet T1] \bullet P, \textit{ where}$$

$$T2 \bullet T1 = \begin{bmatrix} 1 & & dx_1 + dx_2 & a \\ & 1 & dy_1 + dy_2 & b \\ & & 1 & c \\ & & & 1 \end{bmatrix}$$

so translations add

Composing Transformations

University of British Columbia

scaling

$$S2 \bullet S1 = \begin{bmatrix} sx_1 * dx_2 & & & \\ & sy_1 * sy_2 & & \\ & & 1 & \\ & & & 1 \end{bmatrix}$$

so scales multiply

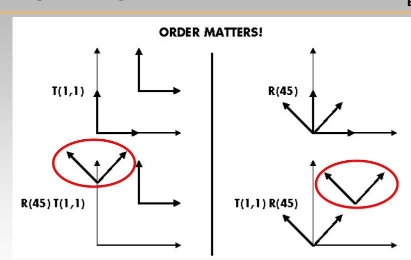
rotation

$$R2 \bullet R1 = \begin{bmatrix} \cos(\theta 1 + \theta 2) & -\sin(\theta 1 + \theta 2) & & \\ \sin(\theta 1 + \theta 2) & \cos(\theta 1 + \theta 2) & & \\ & & 1 & \\ & & & 1 \end{bmatrix}$$

so rotations add

Composing Transformations

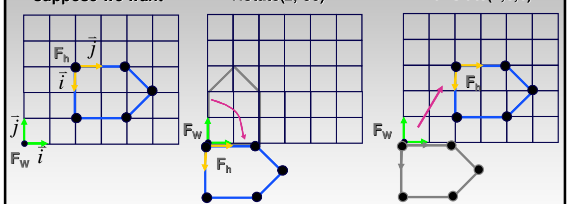
University of British Columbia



Ta Tb = Tb Ta, but Ra Rb != Rb Ra and Ta Rb != Rb Ta

Composing Transformations

University of British Columbia



$P_A = \boxed{Rot(z,-90)\, P_h}$ $\quad P_W = Trans(2,3,0)\, P_A$

$P_W = Trans(2,3,0)\, Rot(z,-90)\, P_h$

Composing Transformations

University of British Columbia

$P_W = Trans(2,3,0)\, Rot(z,-90)\, P_h$

Which direction to read?

- R-to-L: interpret operations wrt fixed coords [object]
- L-to-R: interpret operations wrt local coords [coord sys]
- OpenGL (L-to-R, local coords)

glTranslatef(2,3,0); $\quad M = M \cdot Trans(2,3,0)$

glRotatef(-90,0,0,1); $\quad M = M \cdot Rot(z,-90)$

updates current transformation matrix by postmultiplying

Composing Transformations

University of British Columbia

Undoing Transformations: inverses

$$Trans(x,y,z)^{-1} = Trans(-x,-y,-z)$$
$$Trans(x,y,z)\; Trans(-x,-y,-z) = I$$

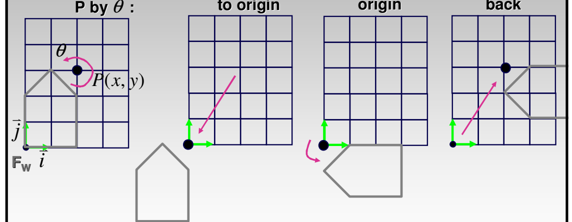
$$Rot(z,\theta)^{-1} = Rot(z,-\theta) = Rot^T(z,\theta)$$ **(R is orthogonal)**

$$Rot(z,\theta)\; Rot(z,-\theta) = I$$

$$Scale(sx,sy,sz)^{-1} = Scale(\frac{1}{sx},\frac{1}{sy},\frac{1}{sz})$$

Rotation about a point

University of British Columbia



$$Trans(x,y,z)\, Rot(z,\theta)\, Trans(-x,-y,-z)$$

Composing of Linear Transformations

University of British Columbia

In general:

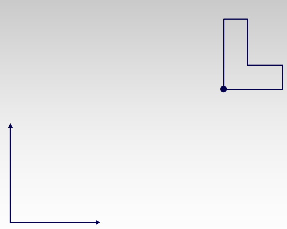
- Transformation of geometry into coordinate system where operation becomes simpler
- Perform operation
- Transform geometry back to original coordinate system

Also works for affine transformations

Composing of Affine Transformations

University of British Columbia

Example: Rotation around arbitrary center

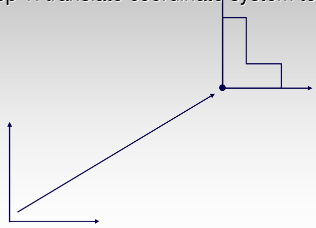


Composing of Affine Transformations

University of British Columbia

Example: Rotation around arbitrary center

- Step 1: translate coordinate system to rotation center

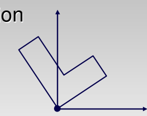


Composing of Affine Transformations

University of British Columbia

Example: Rotation around arbitrary center

- Step 2: perform rotation

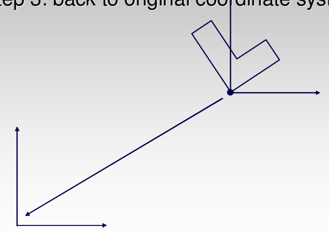


Composing of Affine Transformations

University of British Columbia

Example: Rotation around arbitrary center

- Step 3: back to original coordinate system



Rotation about an arbitrary axis

University of British Columbia

- axis defined by two points
- translate point to the origin
- rotate to align axis with z-axis (or x or y)
- perform rotation
- undo aligning rotations
- undo translation

University of British Columbia

The Rendering Pipeline – An Overview

3D Graphics

University of British Columbia

Modeling:

- Representing object properties
  - *Geometry: polygons, smooth surfaces etc.*
  - *Materials: reflection models etc.*

Rendering:

- Generation of images from models
  - *Interactive rendering*
  - *Ray-tracing*

Animation:

- Making geometric models move and deform

Rendering

University of British Columbia

Goal:
- Transform computer models into images
- May or may not be photo-realistic

Interactive rendering:
- Fast, but until recently low quality
- Roughly follows a fixed patterns of operations
  - ➢ Rendering Pipeline

Offline rendering:
- Ray-tracing
- Global illumination

Rendering

University of British Columbia

Tasks that need to be performed (in no particular order):
- Project all 3D geometry onto the image plane
  - *Geometric transformations*
- Determine which primitives or parts of primitives are visible
  - *Hidden surface removal*
- Determine which pixels a geometric primitive covers
  - *Scan conversion*
- Compute the color of every visible surface point
  - *Lighting, shading, texture mapping*

The Rendering Pipeline

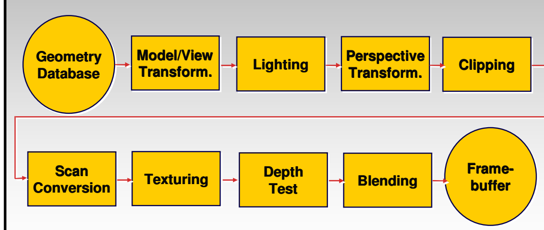
University of British Columbia

What is it? All of this:
- Abstract model for sequence of operations to transform a geometric model into a digital image
- An abstraction of the way graphics hardware works
- The underlying model for application programming interfaces (APIs) that allow the programming of graphics hardware
  - *OpenGL*
  - *Direct 3D*

Actual implementations of the rendering pipeline will vary in the details

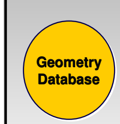
The Rendering Pipeline

University of British Columbia



The Rendering Pipeline
Geometry Database

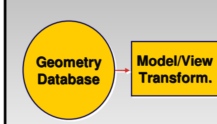
University of British Columbia



Geometry database:
- Application-specific data structure for holding geometric information
- Depends on specific needs of application
  - – *Independent triangles, connectivity information etc.*

The Rendering Pipeline
Model/View Transformation

University of British Columbia

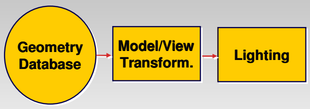


Modeling transformation:
- Map all geometric objects from a local coordinate system into a world coordinate system

Viewing transformation:
- Map all geometry from world coordinates into camera coordinates

The Rendering Pipeline
Lighting

University of British Columbia

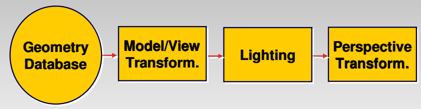


Lighting:

- Compute the brightness of every point based on its material properties (e.g. Lambertian diffuse) and the light position(s)
- Computation is performed *per-vertex*

The Rendering Pipeline
Perspective Transformation

University of British Columbia

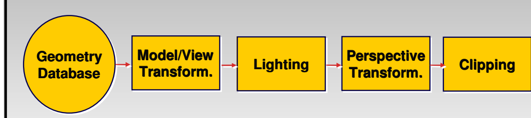


Perspective transformation

- Projecting the geometry onto the image plane
- Projective transformations and model/view transformations can all be expressed with 4x4 matrix operations

The Rendering Pipeline
Clipping

University of British Columbia

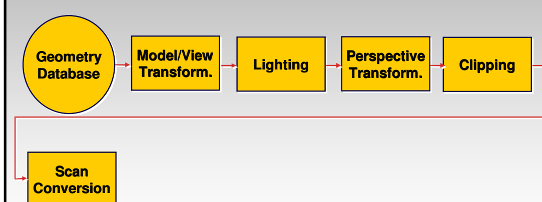


Clipping

- Removal of parts of the geometry that fall outside the visible screen or window region
- May require *re-tessellation* of geometry

The Rendering Pipeline
Scan Conversion

University of British Columbia



The Rendering Pipeline
Scan Conversion

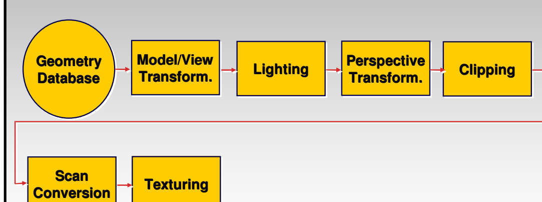
University of British Columbia

Scan conversion

- Turning 2D drawing primitives (lines, polygons etc.) into individual pixels (*discretizing/sampling*)
- Interpolation of colors across the geometric primitive
- This yields a *fragment* (pixel data associated with a particular location in the final image and color values, depth, and some additional information)

The Rendering Pipeline
Texture Mapping

University of British Columbia



The Rendering Pipeline
Texture Mapping

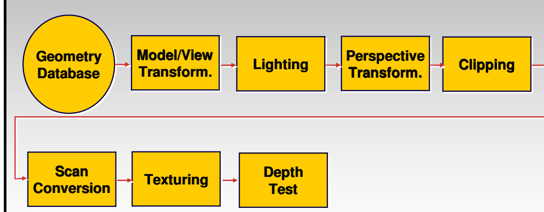
University of British Columbia

Texture mapping

- “gluing images onto geometry”
- The color of every fragment is altered by looking up a new color value from an image

The Rendering Pipeline
Depth Test

University of British Columbia



The Rendering Pipeline
Depth Test

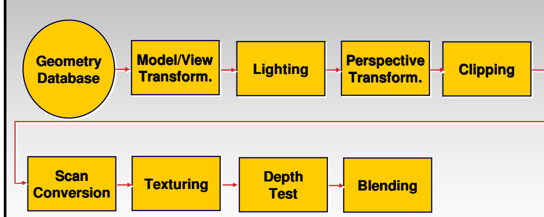
University of British Columbia

Depth test:

- Removes parts of the geometry that are hidden behind other geometry
- Test is performed on every individual fragment
  - *we will also discuss other approaches later*

The Rendering Pipeline
Blending

University of British Columbia



The Rendering Pipeline
Blending

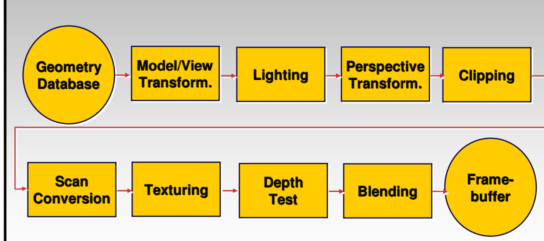
University of British Columbia

Blending:

- Fragments are written to pixels in the final image
- Rather than simply replacing the previous color value, the new and the old value can be combined with some arithmetic operations (blending)
- The video memory on the graphics board that holds the resulting image and is used to display it is called the *framebuffer*

The Rendering Pipeline

University of British Columbia



Discussion

University of British Columbia

Advantages of a pipeline structure

- Logical separation of the different components, modularity
- Easy to parallelize:
  - *Earlier stages can already work on new data while later stages still work with previous data*
  - *Similar to pipelining in modern CPUs*
  - *But much more aggressive parallelization possible (special purpose hardware!)*
  - *Important for hardware implementations!*
- Only local knowledge of the scene is necessary

Discussion

University of British Columbia

Disadvantages:

- Limited flexibility
- Some algorithms would require different ordering of pipeline stages
  - *Hard to achieve while still preserving compatibility*
- Only local knowledge of scene is available
  - *Shadows*
  - *Global illumination*