Geometric Transformations

- recap
- composition of transformations
- rendering pipeline overview

Transformations recap

Affine transformations
- linear transformation + translations
- can be expressed as a 3x3 matrix + 3 vector
  \[ x' = M \cdot x + t \]

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 Coordinatess

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

\[
\begin{pmatrix}
x \\
y \\
z \\
w
\end{pmatrix} = \begin{pmatrix}
x - w \\
y - w \\
z - w \\
w
\end{pmatrix} \quad \forall w \neq 0
\]

Geometrically In 2D

Cartesian Coordinates:

Homogeneous Coordinates:

\[
\begin{pmatrix}
x \\
y \\
z \\
w
\end{pmatrix} = \begin{pmatrix}
x - w \\
y - w \\
z - w \\
w
\end{pmatrix}
\]

\( w=1 \)
Homogeneous Matrices

**Affine Transformations**

\[
\begin{bmatrix}
  x' \\
  y' \\
  z'
\end{bmatrix} =
\begin{bmatrix}
  m_{11} & m_{12} & m_{13} & t_x \\
  m_{21} & m_{22} & m_{23} & t_y \\
  m_{31} & m_{32} & m_{33} & t_z \\
  0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
  x \\
  y \\
  z \\
  1
\end{bmatrix}
\]

Homogeneous Matrices

**Note:**

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

\[
\begin{bmatrix}
  x' \\
  y' \\
  z' \\
  1
\end{bmatrix} =
\begin{bmatrix}
  m_{11} & m_{12} & m_{13} & t_x \\
  m_{21} & m_{22} & m_{23} & t_y \\
  m_{31} & m_{32} & m_{33} & t_z \\
  0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
  x \\
  y \\
  z \\
  1
\end{bmatrix}
\]

Homogeneous Matrices

**Combining the two matrices into one:**

\[
\begin{bmatrix}
  x' \\
  y' \\
  z'
\end{bmatrix} =
\begin{bmatrix}
  m_{11} & m_{12} & m_{13} & t_x \\
  m_{21} & m_{22} & m_{23} & t_y \\
  m_{31} & m_{32} & m_{33} & t_z \\
  0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
  x \\
  y \\
  z \\
  1
\end{bmatrix}
\]

Homogeneous Vectors

**Point: w \geq 1**

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

**Vectors are invariant under translation!**

Homogeneous Vectors

**Representing vectors/directions in homogeneous coordinates**

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

\[
\begin{bmatrix}
  x' \\
  y' \\
  z'
\end{bmatrix} =
\begin{bmatrix}
  m_{11} & m_{12} & m_{13} & t_x \\
  m_{21} & m_{22} & m_{23} & t_y \\
  m_{31} & m_{32} & m_{33} & t_z \\
  0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
  x \\
  y \\
  z \\
  1
\end{bmatrix}
\]

Transformations

- **Translate(a,b,c)**

- **Scale(a,b,c)**

- **Rotate (x, \theta)**

- **Rotate (y, \theta)**

- **Rotate (z, \theta)**
Transformations

Arriving at a transformation...

- Object defined in local coords
- Place object in world

How about the following?

Shear matrix:

\[
\begin{pmatrix}
1 & 1 \\
0 & 1 \\
0 & 0
\end{pmatrix}
\]

Composing Transformations

**Scaling**

\[ S_2 \cdot S_1 = \begin{pmatrix}
dx_1 & dx_2 \\
dy_1 & dy_2
\end{pmatrix}
\]

So scales multiply

**Rotation**

\[ R_2 \cdot R_1 = \begin{pmatrix}
cos(\theta_1 + \theta_2) & -sin(\theta_1 + \theta_2) \\
sin(\theta_1 + \theta_2) & cos(\theta_1 + \theta_2)
\end{pmatrix}
\]

So rotations add

Composing Transformations

**Translation**

\[ T_1 \cdot (dx_1, dy_1) = \begin{pmatrix}
1 & dx_1 \\
0 & 1 \\
0 & 0
\end{pmatrix}
\]

\[ T_2 \cdot (dx_2, dy_2) = \begin{pmatrix}
1 & dx_2 \\
0 & 1 \\
0 & 0
\end{pmatrix}
\]

\[ P'' = T_2 \cdot T_1 \cdot P = [T_2 \cdot T_1] \cdot P, \text{ where} \]

\[ T_2 \cdot T_1 = \begin{pmatrix}
dx_1 & dx_2 \\
dy_1 & dy_2 \\
1 & 1
\end{pmatrix}
\]

So translations add

**Order Matters**

- \( T_a \cdot T_b = T_b \cdot T_a \), but \( R_a \cdot R_b \neq R_a \cdot R_b \) and \( T_a \cdot R_b \neq R_b \cdot T_a \)

Composing Transformations
Composing Transformations

**Suppose we want**

\[ P_t = \text{Rot}(z, -90) \cdot P
\]

**Translate(2,3,0)**

\[ P_o = \text{Trans}(2,3,0) \cdot P
\]

**Rotation about a point**

\[ P' = \text{Rot}(z, \theta) \cdot P \]

\[ P'' = \text{Trans}(x, y, z) \cdot P'
\]

**Composing of Linear Transformations**

**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 Transformations**

\[ P_w = \text{Rot}(z, -90) \cdot P_t
\]

**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)

\[ \text{glTranslate}(2,3,0); \quad \text{M} = \text{M} \cdot \text{Trans}(2,3,0)
\]

\[ \text{glRotate}(-90,0,0,1); \quad \text{M} = \text{M} \cdot \text{Rot}(z, -90)
\]

updates current transformation matrix by postmultiplying

**Undoing Transformations: inverses**

\[ \text{Trans}(x, y, z)^{-1} = \text{Trans}(-x, -y, -z)
\]

\[ \text{Trans}(x, y, z) \cdot \text{Trans}(-x, -y, -z) = I
\]

\[ \text{Rot}(z, \theta)^{-1} = \text{Rot}(z, \theta) \quad \text{(R is orthogonal)}
\]

\[ \text{Rot}(z, \theta) \cdot \text{Rot}(z, \theta) = I
\]

\[ \text{Scale}(x, y, z)^{-1} = \text{Scale}(1/x, 1/y, 1/z)
\]
Composing of Affine Transformations

**Example: Rotation around arbitrary center**
- Step 1: translate coordinate system to rotation center

Composing of Affine Transformations

**Rotation about an arbitrary axis**
- 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

The Rendering Pipeline – An Overview

3D Graphics

**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

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

The Rendering Pipeline

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
  - Direct3D

Actual implementations of the rendering pipeline will vary in the details

The Rendering Pipeline

Geometry Database

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

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:**
- 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

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

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

**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)

**Texture Mapping**
The Rendering Pipeline

Texture Mapping

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

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

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
**Discussion**

**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

**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