Lecture 5:

Transforms II

Computer Graphics and Imaging
UC Berkeley CS184/284A
3D Transforms
3D Transformations

Use homogeneous coordinates again:

- 3D point $= (x, y, z, 1)^T$
- 3D vector $= (x, y, z, 0)^T$

Use 4×4 matrices for affine transformations

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} a & b & c & t_x \\ d & e & f & t_y \\ g & h & i & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$
3D Transformations

Scale

\[
S(s_x, s_y, s_z) = \begin{pmatrix}
    s_x & 0 & 0 & 0 \\
    0 & s_y & 0 & 0 \\
    0 & 0 & s_z & 0 \\
    0 & 0 & 0 & 1
\end{pmatrix}
\]

Translation

\[
T(t_x, t_y, t_z) = \begin{pmatrix}
    1 & 0 & 0 & t_x \\
    0 & 1 & 0 & t_y \\
    0 & 0 & 1 & t_z \\
    0 & 0 & 0 & 1
\end{pmatrix}
\]

Coordinate Change (Frame-to-world)

\[
F(u, v, w, o) = \begin{bmatrix}
    u & v & w & o \\
    0 & 0 & 0 & 1
\end{bmatrix}
\]
3D Transformations

Rotation around x-, y-, or z-axis

\[ R_x(\alpha) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & \cos \alpha & -\sin \alpha & 0 & 0 \\ 0 & \sin \alpha & \cos \alpha & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \]

\[ R_y(\alpha) = \begin{pmatrix} \cos \alpha & 0 & \sin \alpha & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ -\sin \alpha & 0 & \cos \alpha & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \]

\[ R_z(\alpha) = \begin{pmatrix} \cos \alpha & -\sin \alpha & 0 & 0 & 0 \\ \sin \alpha & \cos \alpha & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \]
3D Rotations

Compose any 3D rotation from $R_x$, $R_y$, $R_z$?

$$R_{xyz}(\alpha, \beta, \gamma) = R_x(\alpha) R_y(\beta) R_z(\gamma)$$

• So-called Euler angles
• Often used in flight simulators: roll, pitch, yaw
3D Rotations

Compose any 3D rotation from $R_x$, $R_y$, $R_z$?

$$R_{xyz}(\alpha, \beta, \gamma) = R_x(\alpha) R_y(\beta) R_z(\gamma)$$

- So-called Euler angles
- Often used in flight simulators: roll, pitch, yaw
3D Rotation Around Arbitrary Axis

Construct orthonormal frame transformation $F$ with $p, u, v, w$, where $p$ and $w$ match the rotation axis.

Apply the transform $(F R_z(\theta) F^{-1})$.

Interpretations (both valid):
- Move to Z axis, rotate, then move back.
- Cast w-axis rotation in new coordinate frame.
Rodrigues’ Rotation Formula

Rotation by angle $\alpha$ around axis $n$

$$ R(n, \alpha) = \cos(\alpha) I + (1 - \cos(\alpha)) nn^T + \sin(\alpha) \begin{pmatrix} 0 & n_z & -n_y \\ -n_z & 0 & n_x \\ n_y & -n_x & 0 \end{pmatrix} $$

How to prove this magic formula?

- Matrix $N$ computes a cross-product: $N x = n \times x$
- Assume orthonormal system $e_1, e_2, n$

$$ Rn = n $$
$$ Re_1 = \cos \alpha e_1 + \sin \alpha e_2 $$
$$ Re_2 = -\sin \alpha e_1 + \cos \alpha e_2 $$
Many Other Representations of Rotations

Quaternions

Exponential map

...
Hierarchical Transforms
Skeleton - Linear Representation

head
torso
right upper arm
right lower arm
right hand
left upper arm
left lower arm
left hand
right upper leg
right lower leg
right foot
left upper leg
left lower leg
left foot
Linear Representation

Each shape associated with its own transform

A single edit can require updating many transforms

- E.g. raising arm requires updating transforms for all arm parts
Skeleton - Hierarchical Representation

torso
  head
  right arm
    upper arm
    lower arm
    hand
  left arm
    upper arm
    lower arm
    hand
  right leg
    upper leg
    lower leg
    foot
  left leg
    upper leg
    lower leg
    foot
Hierarchical Representation

Grouped representation (tree)

- Each group contains subgroups and/or shapes
- Each group is associated with a transform relative to parent group
- Transform on leaf-node shape is concatenation of all transforms on path from root node to leaf
- Changing a group’s transform affects all parts
  - Allows high level editing by changing only one node
  - E.g. raising left arm requires changing only one transform for that group
translate(0, 10);
drawTorso();

pushmatrix(); // push a copy of transform onto stack
translate(0, 5); // right-multiply onto current transform
rotate(headRotation); // right-multiply onto current transform
drawHead();
popmatrix(); // pop current transform off stack

pushmatrix();
translate(-2, 3);
rotate(rightShoulderRotation);
drawUpperArm();
pushmatrix();
translate(0, -3);
rotate(elbowRotation);
drawLowerArm();
pushmatrix();
translate(0, -3);
rotate(wristRotation);
drawHand();
popmatrix();
popmatrix();
translate(0, 10);
drawTorso();

pushmatrix(); // push a copy of transform onto stack
translate(0, 5); // right-multiply onto current transform
rotate(headRotation); // right-multiply onto current transform
drawHead();
popmatrix(); // pop current transform off stack
pushmatrix();
translate(-2, 3);
rotate(rightShoulderRotation);
drawUpperArm();

pushmatrix();
translate(0, -3);
rotate(elbowRotation);
drawLowerArm();

pushmatrix();
translate(0, -3);
rotate(wristRotation);
drawHand();

popmatrix();
popmatrix();
popmatrix();
popmatrix();

....
Viewing and Perspective
Viewing and Perspective Transforms

Scene modeling
3D world coordinates

Rasterization
2D screen coordinates
Camera Space
“Standard” Camera Space

We will use this convention for “standard” camera coordinates:

- camera located at the origin
- looking down negative z-axis
- vertical vector is y-axis
- (x-axis) orthogonal to y & z
“Standard” Camera Coordinates

Resulting image
(z-axis pointing away from scene)
Consider A Camera Pointing in The World
Consider A Camera Pointing in The World

- $u =$ up vector
- $v =$ view direction
- $e =$ eye point (position of camera)
Camera “Look-At” Transformation

Input: e, u & v given in world space coordinates
Output: transform matrix from world space to standard camera space

u = up vector
v = view direction
e = eye point (position of camera)
Camera “Look-At” Transformation

Inverse: Matrix from standard camera to world space
(Why? This is a coordinate frame transform to (e,r,u,-v))

\[
\begin{pmatrix}
  r_x & u_x & -v_x & e_x \\
  r_y & u_y & -v_y & e_y \\
  r_z & u_z & -v_z & e_z \\
  0 & 0 & 0 & 1
\end{pmatrix}
\]

“Look-at” transform is the inverse of above matrix:

\[
\begin{pmatrix}
  r_x & u_x & -v_x & e_x \\
  r_y & u_y & -v_y & e_y \\
  r_z & u_z & -v_z & e_z \\
  0 & 0 & 0 & 1
\end{pmatrix}^{-1}
= \begin{pmatrix}
  r_x & r_y & r_z & 0 \\
  u_x & u_y & u_z & 0 \\
  -v_x & -v_y & -v_z & 0 \\
  0 & 0 & 0 & 1
\end{pmatrix} \cdot \begin{pmatrix}
  1 & 0 & 0 & -e_x \\
  0 & 1 & 0 & -e_y \\
  0 & 0 & 1 & -e_z \\
  0 & 0 & 0 & 1
\end{pmatrix}
\]
How to transform from 3D camera space to 2D image plane?

- One option: orthographic projection (just delete $z$)
- Useful, e.g. for engineering drawings
- But is this the whole story?
Perspective in Art

Berlinghieri 1235
Perspective in Art

Giotto 1290
Perspective in Art

Giotto 1290
Perspective in Art

Brunelleschi experiment c. 1413
Perspective in Art

Delivery of the Keys (Sistine Chapel), Perugino, 1482
Perspective in Art

The Last Supper, Leonardo da Vinci, 1499
Could An Orthographic Projection Model Leonardo’s Last Supper?
Pinhole Camera Model
Pinhole Camera

Fig. 131.—How Light and a Pinhole Form an Image.
Projective Transform

Inverted image (as in real pinhole camera)
Pinhole Camera Projective Transform

Let \((x, y, z)^T\) be a point in the scene. The projection of this point onto the image plane is an upright image point located at coordinates \((x', y')\). The pinhole camera model maps the scene point to the image point through a projective transform, which takes into account the camera's intrinsic parameters and extrinsic orientation.

In the diagram:
- **Scene point**: \((x, y, z)^T\)
- **Image point**: \((x', y')\)
- **Pinhole C**: The center of the camera's optical axis.
- **Distance d**: The distance from the pinhole to the image plane, which is equal to the camera's focal length.

The projective transform relates the scene coordinates to the image coordinates, taking into account the camera's perspective and properties.
Projective Transforms

Standard perspective projection

- Center of projection: \((0, 0, 0)^T\)
- Image plane at \(z = d\)

\[
\begin{pmatrix}
x \\
y \\
z
\end{pmatrix}
\mapsto
\begin{pmatrix}
x \cdot d/z \\
y \cdot d/z \\
d
\end{pmatrix}
\]
Projective Transforms

Standard perspective projection

- Center of projection: $(0, 0, 0)^T$
- Image plane at $z = d$

$$
\begin{pmatrix}
  x \\
  y \\
  z
\end{pmatrix}
\rightarrow
\begin{pmatrix}
  x \cdot d/z \\
  y \cdot d/z \\
  d
\end{pmatrix}
$$

Perspective foreshortening

- Need division by $z$
- Matrix representation?
  ➞ Homogeneous coordinates!
Homogenous Coordinates (3D)

\[
p = \begin{pmatrix} wx \\ wy \\ wz \\ w \end{pmatrix} \quad \leftrightarrow \quad \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}
\]

\[
M = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{pmatrix}
\]

Note non-zero term in final row.
First time we have seen this.

\[
q = M \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} xd/z \\ yd/z \\ d \\ 1 \end{pmatrix}
\]
Pinhole Camera Model

This mathematical model produces all linear perspective effects!

- Converging lines + vanishing points
- Closer objects appear larger in images
- ...

CS184/284A

Ren Ng
Specifying Real Camera Perspectives
Perspective Composition = Camera Position + Angle of View

In this sequence, angle of view decreases as distance from subject increases, to size of human subject in image.

Notice the dramatic change in background perspective.

From Canon EF Lens Work III
Perspective Composition

Up close and zoomed wide with short focal length

16 mm (110°)
Perspective Composition

Walk back and zoom in with long focal length

200 mm (12°)
Specifying Perspective Projection

From Angel and Shreiner, Interactive Computer Graphics
Specifying Perspective Viewing Volume

From Angel and Shreiner, Interactive Computer Graphics
Specifying Perspective Viewing Volume

Parameterized by

- **fovy** : vertical angular field of view
- **aspect ratio** : width / height of field of view
- **near** : depth of near clipping plane
- **far** : depth of far clipping plane

Derived quantities

- **top** = near * tan (fovy)
- **bottom** = – top
- **right** = top * aspect
- **left** = – right
Perspective Projection Implementation
Perspective Projection Transform

Later we will “flatten” NDC to get screen coordinates.
Perspective Projection Transform

Notes:

• Need not be symmetric about z-axis, but for simplicity here we assume so

• This transform will preserve depth information (ordering) in NDC
Perspective Transform Matrix

\[ P = \begin{bmatrix}
  \text{near} & 0 & 0 & 0 \\
  \text{right} & 0 & 0 & 0 \\
  0 & \text{near} & -\frac{\text{far}+\text{near}}{\text{far}-\text{near}} & -2\frac{\text{far}\times\text{near}}{\text{far}-\text{near}} \\
  0 & 0 & -1 & 0
\end{bmatrix} \]
Perspective Transform Matrix Example

\[
P = \begin{bmatrix}
\frac{\text{near}}{\text{right}} & 0 & 0 & 0 \\
0 & \frac{\text{near}}{\text{top}} & 0 & 0 \\
0 & 0 & -\frac{\text{far}+\text{near}}{\text{far}-\text{near}} & -2\frac{\text{far}\cdot\text{near}}{\text{far}-\text{near}} \\
0 & 0 & -1 & 0 \\
\end{bmatrix}
\begin{bmatrix}
\frac{\text{right}}{\text{top}} \\
-\text{near} \\
1 \\
\end{bmatrix}
\]

\[
= \begin{bmatrix}
\frac{\text{near}}{\text{near}} \\
\frac{\text{near}}{\text{near}} \\
\frac{\text{far}+\text{near}}{\text{far}-\text{near}} - 2\frac{\text{far}\cdot\text{near}}{\text{far}-\text{near}} \\
\frac{\text{far}+\text{near}}{\text{far}-\text{near}} \\
\end{bmatrix}
\begin{bmatrix}
1 \\
1 \\
-\frac{\text{far}+\text{near}}{\text{far}-\text{near}} \\
1 \\
\end{bmatrix}
\]

\[
= \begin{bmatrix}
1 \\
1 \\
-1 \\
1 \\
\end{bmatrix}
\]

View volume

Camera Coordinates

Normalized Device Coords “NDC”
Transforms Recap
Transforms Recap

Coordinate Systems

• Object coordinates
  • Apply modeling transforms...

• World (scene) coordinates
  • Apply viewing transform...

• Camera (eye) coordinates
  • Apply perspective transform + homog. divide...

• Normalized device coordinates
  • Apply 2D screen transform...

• Screen coordinates
Transforms Recap

Object coords  \rightarrow  Modeling transforms  \rightarrow  World coords
Transforms Recap

World coords

Viewing transform

Camera coords
Transforms Recap

Camera coords

Perspective projection and homogeneous divide

NDC

(1, 1, 1)

(-1, -1, -1)
Transforms Recap

NDC

(1, 1, 1)

(-1, -1, -1)

Screen transform

Screen coords

(0, 0)

(w, h)
Transforms Recap

Screen coords

Rasterization
Things to Remember

Transform uses

• Basic transforms: rotate, scale, translate, ...
• Modeling, viewing, projection, perspective
• Change in coordinate system
• Hierarchical scene descriptions by push/pop

Implementing transforms

• Linear transforms = matrices
• Transform composition = matrix multiplication
• Homogeneous coordinates for translation, projection
Acknowledgments

Thanks to Pat Hanrahan, Kayvon Fatahalian, Mark Pauly for presentation resources.