Homogeneous Coordinates

- Represent 2D points by 3D vectors
- \((x,y)\rightarrow(x,y,1)\)
- Now a multitude of 3D points \((x,y,W)\) represent the same 2D point, \((x/W, y/W, 1)\)
- Represent 2D transforms with 3 by 3 matrices
- Can now do translations
- Homogenous coordinates have other uses/advantages (later)

2D Translation in H.C.

\[
P_{\text{new}} = P + T
\]
\[
(x', y') = (x, y) + (t_x, t_y)
\]
\[
M = \begin{bmatrix}
1 & 0 & t_x \\
0 & 1 & t_y \\
0 & 0 & 1
\end{bmatrix}
\]

2D Scale in H.C.

\[
M = \begin{bmatrix}
S_x & 0 & 0 \\
0 & S_y & 0 \\
0 & 0 & 1
\end{bmatrix}
\]

2D Rotation in H.C.

\[
M = \begin{bmatrix}
\cos \theta & -\sin \theta & 0 \\
\sin \theta & \cos \theta & 0 \\
0 & 0 & 1
\end{bmatrix}
\]

Composition of Transformations

- If we use one matrix, \(M_1\) for one transform and another matrix, \(M_2\) for a second transform, then the matrix for the first transform followed by the second transform is simply \(M_2M_1\)
- This generalizes to any number of transforms
- Computing the combined matrix \textbf{first} and then applying it to many objects, can save \textbf{lots} of computation

Composition Example

- Matrix for rotation about a point, \(P\)
- Problem--we only know how to rotate about the origin.
Composition Example

- Matrix for rotation about a point, P
- Problem—we only know how to rotate about the origin.
- Solution—translate to origin, rotate, and translate back

2D transformations (continued)

- The transformations discussed so far are invertable (why?). What are the inverses?

2D viewing

- Three coordinate systems are common in graphics
  - World coordinates or modeling coordinates - where the model is defined (meters, miles, etc.)
  - Normalized device coordinates; usually (0-1) in each variable.
  - Device coordinates: the actual coordinates of the pixels on the frame-buffer or the printer
- Need to construct transformations between coordinate systems
- Terminology:
  - window = region on drawing that will be displayed (rectangle)
  - viewport = region in NDC’s/DC’s where this rectangle is displayed (often simply entire screen).

Determining the transform

- **Plan A**: Consider this as a sequence of transformations in homogenous coords, then determine each element in closed form.
- **Plan B**: Compute numerically from point correspondences.

- write \((wx, wy)\) for coordinates of \(i\)’th point on window
- translation is:

\[
\begin{bmatrix}
1 & 0 & wx \\
0 & 1 & wy \\
0 & 0 & 1
\end{bmatrix}
\]

(overbar denotes average over vertices, i.e., 1,2,3,4)
The text in the image seems to be discussing geometric transformations, particularly in the context of 2D and 3D coordinate systems. The diagrams illustrate various transformations such as rotation, scaling, and translation. The text mentions the need to compute a theta for a rotation matrix, and it also discusses the multiplication of transformation matrices to get the overall transformation.

Plan B is highlighted, indicating a direct approach to solving the problem. The plan likely involves transforming the geometry within the window to the desired coordinates, followed by clipping and drawing.

The text appears to be focused on computer graphics or a related field, possibly discussing computer-aided design (CAD) or a similar application. The transformations are described using mathematical notation and geometric illustrations, which are typical in such contexts.
Plan B: Solve for the affine transformation directly.

- We know that this is an “affine” transform.
- In particular, the matrix we seek is:

\[
\begin{bmatrix}
  a & b & c \\
  d & e & f \\
  0 & 0 & 1
\end{bmatrix}
\]

More Details

Write

\[ax_1 + by_1 + c = u_1\]

As

\[x_1a + y_1b + 1 \cdot c + 0 \cdot d + 0 \cdot e + 0 \cdot f = u_1\]

Notice that this gives one equation in the six unknowns

More Details

- Consider the first mapping, \(M_{p_1q_1}\)
- \(p_1 = (x_1, y_1, 1)^T, \quad q_1 = (u_1, v_1, 1)^T\)

\[
\begin{bmatrix}
  a & b & c \\
  d & e & f \\
  0 & 0 & 1
\end{bmatrix}
\]

\[ax_1 + by_1 + c = u_1\]

\[dx_1 + ey_1 + f = v_1\]

More Details

Similarly, write

\[dx_1 + ey_1 + f = v_1\]

As

\[0 \cdot a + 0 \cdot b + 0 \cdot c + x_1 \cdot d + y_1 \cdot e + 1 \cdot f = u_1\]

Notice that this gives a second equation in the six unknowns

More Details

- \(M_{p_1q_1}\) gives first two rows
- \(p_1 = (x_1, y_1, 1)^T, \quad q_1 = (u_1, v_1, 1)^T\)

\[
\begin{bmatrix}
  a & b & c \\
  d & e & f \\
  0 & 0 & 1
\end{bmatrix}
\]

\[ax_1 + by_1 + c = u_1\]

\[dx_1 + ey_1 + f = v_1\]

More Details

\(M_{p_1q_1}, M_{p_2q_1}\) give other rows

Final representation of six equations in six unknowns

This can be solved using standard methods
Hierarchical modeling

- Consider constructing a complex 2D drawing, e.g., an animation showing the plan view of a building, where the doors swing open and shut.

Hierarchical modeling

Options:
- Specify everything in world coordinate frame, but then each room is different, and each door moves differently.
- Exploit similarities by using repeated copies of models in different places (instancing).

Hierarchical modeling

- Model form
  - Directed acyclic graph.
  - Each node consists of 0 or more objects (lines, polygons, etc).
  - Each edge is a transformation
- There can be many edges joining two nodes (e.g., in the case of the corridor - many copies of the same room model, each transformed differently).
- Every graphics API supports hierarchies - some directly (meaning you have to learn a language to express the model) some indirectly with a matrix stack

Hierarchical modeling

Write the transformation from door coordinates to room coordinates as:

\[ T_{\text{door}}^{\text{room}} \]

Then to render a door, use the transformation:

\[ T_{\text{world}}^{\text{device}} T_{\text{floor}}^{\text{corridor}} T_{\text{corridor}}^{\text{room}} T_{\text{door}}^{\text{room}} \]

To render a body, use the transformation:

\[ T_{\text{world}}^{\text{device}} T_{\text{floor}}^{\text{corridor}} T_{\text{corridor}}^{\text{room}} T_{\text{body}}^{\text{room}} \]

Matrix stacks and rendering

- Matrix stack:
  - Stack of matrices used for rendering
  - Applied in sequence.
  - Pops/remove last matrix
  - Pushes/append a new matrix
- In previous example, body-device transformation comes from door-device transformation by popping door-room and pushing body-room
Matrix stacks and rendering

- Algorithm for rendering a hierarchical model:
  - stack is $T_{\text{root}}$
  - render (root)

- Recursive definition of render (node)
  - if node has object, render it
  - for each child:
    - push transformation
    - render (child)
    - pop transformation

Now to render door on first room in first corridor, stack looks like: $W \cdot C1 \cdot R1 \cdot D1$

- For efficiency we would store “running” products, i.e., the stack contains: $W$, $W \cdot C1$, $W \cdot C1 \cdot R1$, $W \cdot C1 \cdot R1 \cdot D1$.

- We do not need two copies of corridor, or 16 copies of body; we render one copy using 16 different transformations. This is known as instancing

- Animation requires care; if $D1$ is a single function of time, all doors will swing open and closed at the same time.

- Stack is $W$
- render kneecap
- Stack is $W \cdot L$
- render ankle
- Stack is $W \cdot L \cdot F$
- render foot
- Stack is $W \cdot L \cdot S$
- render shin
- Stack is $W \cdot T$
- render thigh