COMP 557 Study guide

Francis Piche

September 17, 2018

Contents

Part I
Introduction

1 Disclaimer

These notes are curated from Professor Paul Kry COMP557 lectures at McGill University. They are for study purposes only. They are not to be used for monetary gain.

2 About This Guide

I make my notes freely available for other students as a way to stay accountable for taking good notes. If you spot anything incorrect or unclear, don't hesitate to contact me via Facebook or e-mail at `http://francispiche.ca/contact/`

Part II
Transformations

3 Notation

3.1 Implicit Representation

$$(1)\{\vec{v}|f(\vec{v})=0\}$$

In English, this means the set of all vectors v, such that some function f of v gives zero. For example, if $f=\vec{v}\cdot\vec{u}+k$. Then the set of all vectors which satisfy (1) would be some line (since the dot product gives a scalar, and the k is a scalar. So we need to dot product to be $-k$). If k is 0, then the set is just the set of vectors orthogonal to $\vec{u}$, which is a line.

Another example:

$$\{\vec{v}|(\vec{v}-\vec{p})\cdot(\vec{v}-\vec{p})-\vec{r}^2=0\}$$

This is just a fancy way of expressing the equation of a circle centered at $\vec{p}$. (Plug some sample vectors in if you don't believe it)

3.2 Explicit (Parametric) Representation

A parameter is given with a specified domain to describe the equation. For example:

$$\{\vec{p}+r\begin{pmatrix}cos(t)\\sin(t)\end{pmatrix}|t\in[0,2\pi]\}$$

describes a circle.

4 Linear Transformations

A great video (and series) to visualize the linear algebra used in graphics is this from 3Blue1Brown on YouTube.

Transformations are like "functions" that operate on a set of points.

Parametric form of a mapping from one set to another using some transform T:

$$\{f(t)|t \in D\} \to \{T(f(t))|t \in D\}$$

Implicit form:

$$\{\vec{v}|f(\vec{v} = 0)\} \to \{T(\vec{v})|f(\vec{v}) = 0\}$$
$$= \{\vec{v}|f(T^{-1}(\vec{v})) = 0\}$$

To convince yourself of that last equality, try it on few examples.

Translation:

$$T(\vec{v}) = \vec{v} + \vec{u}$$

See the slides for visual representations of the common linear transformations: (translation, sheer, scale, rotation, reflection etc.) The video mentioned above is also nice for getting a feel of how it works.

4.1 Combining Linear Transforms with Translation

Could do it this way:

$$T(\vec{p}) = M\vec{p} + \vec{u}$$

for some matrix M, but if you try to do that with a composition:

$$T(\vec{p}) = M_T\vec{p} + \vec{u}_T$$

$$S(\vec{p}) = M_S\vec{p} + \vec{u}_S$$

then

$$(S \circ T) = M_S(M_T\vec{p} + \vec{u}_T) + \vec{u}_S$$

which is honestly pretty gross to look at. We can do better using *homogeneous coordinates*.

4.1.1 Homogeneous Coordinates

This is the use of a 3x3 matrix to perform translation with a linear transformation. We add an extra component w, to our 2x2 vectors, and an extra row $(0, 0, w)$ and column $\begin{pmatrix} 0 \\ 0 \\ w \end{pmatrix}$ For points in an affine space, $w = 1$.

Linear transformations:

$$\begin{pmatrix} a & b & 0 \\ c & d & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} ax+by \\ cx+dy \\ 1 \end{pmatrix}$$

Translation (uses the extra column):

$$\begin{pmatrix} 1 & 0 & t \\ 0 & 1 & s \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} x+t \\ y+s \\ 1 \end{pmatrix}$$

If we now do composition, we can do it like this (using block notation):

$$\begin{pmatrix} M_S & \vec{u_S} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} M_T & \vec{u_T} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} p \\ 1 \end{pmatrix}$$

$$= \begin{pmatrix} M_S M_T & M_S \vec{u_T} + \vec{u_S} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} p \\ 1 \end{pmatrix}$$

Which is essentially the same, but cleaner and will be more useful later.

4.2 Affine Transformations

These are transformations in which lines that were straight, and lines that were parallel to each other are still straight and parallel to each other. Also, the ratios of lengths along lines are preserved.

Common transforms:

Translation:

$$\begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix}$$

is a translation of t_x in the x direction and t_y in the y.

Scale:

$$\begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

to scale s_x in the x-axis and s_y in the y-axis.

4.3 Rigid Motions

A transformation made up of only translation and rotation is a rigid motion.

Note that for rotations, the inverse is the transpose (rotation matrices are orthogonal), and so the inverse of a rigid motion E is:

$$E = \begin{pmatrix} R & \vec{u} \\ 0 & 1 \end{pmatrix}$$

$$E^{-1} = \begin{pmatrix} R^T & -R^T\vec{u} \\ 0 & 1 \end{pmatrix}$$

4.4 Composing to change axes

To rotate about a point other than the origin, first we translate, then rotate, and translate back.

$$M = T^{-1}RT$$

To scale along a particular axis and point, you would move it to the point, rotate so that the axis lines up with the x or y axis, scale, then undo all the operations.

$$M = T^{-1}R^{-1}SRT$$

4.5 Points vs. Vectors

Points and vectors are NOT the same. Points are locations in space, whereas vectors can be thought of displacements in space, or a tuple of distance and direction between points.

In homogeneous coordinates, vectors have $w = 0$.

Translations do not affect vectors:

$$\begin{pmatrix} M & t \\ 0 & 1 \end{pmatrix} \begin{pmatrix} p \\ 1 \end{pmatrix} = \begin{pmatrix} Mp + t \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} M & t \\ 0 & 1 \end{pmatrix} \begin{pmatrix} v \\ 0 \end{pmatrix} = \begin{pmatrix} Mv \\ 0 \end{pmatrix}$$

5 Change of Coordinates

This is similar to the idea of moving an object to the origin to apply transformations and moving it back, but more formulaic. (A computer could do it!)

$$T_e = FT_F F^{-1}$$

Where T_e is the transform expressed with respect to the canonical basis (The form we're used to). T_F is the transformation expressed in the *natural* frame. F is the transform which takes us from the new basis to the canonical one. It's form is: $\begin{pmatrix} \vec{u} & \vec{v} & \vec{p} \end{pmatrix}$ where u, v are the axes and p is the new origin.

6 Aside: Classes of transforms

There is a sort of "hierarchy" of transforms. The classes are as follows:

- Homographies (Lines remain lines)
- Affine (preserve parallel lines)
- Conformal (Also preserve angles)
- Rigid (also preserve lengths)

where:

$$Rigid \subseteq Conformal \subseteq Affine \subseteq Homographies$$

7 3D Transformations

3D Transformations are essentially the same as 2D. (where we add the extra row/column and put a 1 or 0) We simply use a 4x4 matrix instead of a 3x3. For example:

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Would be a translation by t_x ,t_y and t_z.

Rotations get a little weird. We not only have to specify an angle, but now also an axis of rotation. So:

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} cos(\theta) & -sin(\theta) & 0 & 0 \\ sin(\theta) & cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Would be a rotation about the z axis.

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(\theta) & -sin(\theta) & 0 \\ 0 & sin(\theta & cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Would be a rotation about the x axis.

As you can see, this is pretty messy to come up with for an arbitrary axis. It would be nice to represent rotations in more intuitive ways.

A matrix for a reflection in the plane which passes through the origin, with normal n would be derived as follows. First, we need to project the point p onto the normal n, using the dot product

$$(n \cdot p)n$$

. (You can imagine the dot was moved onto the normal line) Then multiplying by -2 to move the dot through the plane backwards:

$$-2(n \cdot p)n$$

We now move the dot to where it should float in space by adding p.

$$-2(n \cdot p) + p$$

To now express this as a matrix, we use n and p as vectors, and add Ip at the end instead of p.

$$p' = -2(\begin{pmatrix} n_x \\ n_y \\ n_z \end{pmatrix} \begin{pmatrix} n_x & n_y & n_z \end{pmatrix}) + \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix}$$

7.1 Rotations in 3D

In general, 3D scaling, rotation and translation do not commute. Think of a rotation followed by a non-uniform scale. This will appear to mutate the object. See (`https://www.youtube.com/watch?v=ThD1h-0_noE` for an illustration) The reason is that the axis of scaling changes before the scale is applied.

Similarly, a translation followed by a rotation does not commute. The axis of rotation will be relative to the origin, but we're no longer at the origin, so the object would move in a wide arc around the origin rather than rotate around it's center as we'd expect.

Note that 3D rotations are defined as:

$$R \in \mathbb{R}^{3x3} : R^T = R^{-1}, det(R) = 1$$

Also note that we have 3 more degrees of freedom in 3D, since we can rotate about any axis. So knowing the angle, and the axis direction is all the info required to describe a rotation.

7.2 Euler Angles

Coming up with general rotation matrices is kind of hard, so it can be useful to break them down into rotations about the canonical axes. This is possible with Euler Angles.

Any rotation can be broken down into 3 rotations about the x, y, or z axis.

The first rotation is rotating in the canonical frame, while the others are rotating with respect to the object frame.

It takes exactly 3 rotations, with no two successive rotations being along the same axis to be able to express ANY rotation. (Euler's Theorem)

7.2.1 Gimbal Lock

This is a problem that can occur with using Euler Angles if the first rotation "squashes" an axis. That is, the rotation places the new axis directly on top on an existing axis, removing a possible axis of rotation.

For example a rotation of 90 degrees about the z axis would leave the z on top of the x, removing the possibility of rotating about the z and x axes independently.

This video that Prof. Kry put in the slides is super useful for understanding Euler angles and Gimbal lock. `https://www.youtube.com/watch?v=zc8b2Jo7mno`

7.2.2 Interpolation of Euler Angles

If you simply interpolate each rotation one after the other, you'll get different paths for different orders of rotation. For example if you rotate xyx you'd get a different path than xyz even if the end rotation is the same. Changing the order of rotations can also be leveraged to avoid Gimbal lock.

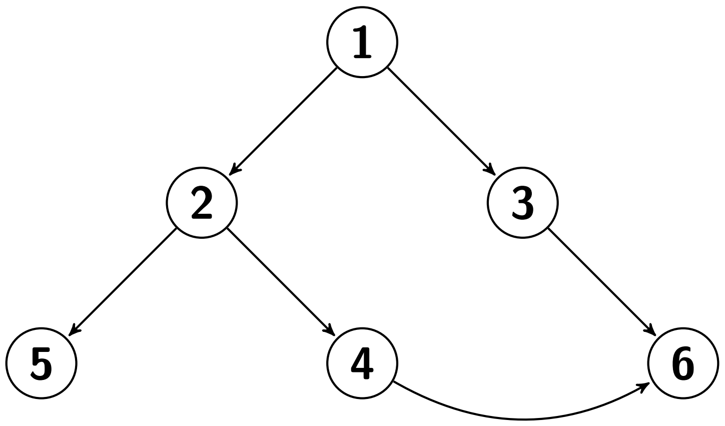
8 Scene Graphs

When you have multiple objects in a scene, and need them to have working relationships in terms of how they move, we use a scene graph.

This allows for the grouping of objects in a way that makes sense. For example, you can (as in the assignment) make a character with limbs that can move independently, or together depending on where you apply transformations in the graph.

In general, we use a "group" node to keep sets of objects together. Interior nodes in the graph are groups, and the leaves are objects.

However, we can also have relations to multiple "parents" so we actually have a directed acyclic graph, not a tree (DAG).



So here, tranforming 1 will apply the same transform to all the nodes, whereas transforming 3 or 4 will affect 6, and transforming 6 will affect only 6.

Having multiple parents is called "instancing".

Part III